RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CSD 334 MINI PROJECT REPORT

Department Stock Management System

Submitted by

Adwaith Jayan(Reg. No: KTE22CS006)
Anand Sankar P T(Reg. No: KTE22CS018)
Arjun Sabu(Reg. No: KTE22CS022)
Niranjan M Varma(Reg. No: KTE22CS057)

APJ ABDUL KALAM TECHNOLOGICAL UNIVERSITY
THIRUVANANTHAPURAM

APRIL 2025

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

Vision of the Department

To be a center of excellence for nurturing young minds to become innovative computing professionals for the empowerment of society.

Mission of the Department

- To offer a solid foundation in computing and technology for crafting competent professionals.
- To promote innovative and entrepreneurial skills of students by exposing them to the forefront of developments in the field of computing.
- To inculcate strong ethical values in the young minds to work with commitment for the progress of the nation.

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

GOVERNMENT ENGINEERING COLLEGE

KOTTAYAM-686 501

DEPARTMENT OF
COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that this report entitled ***Department Stock Management System-A Web-Based Stock Management System*** *is an authentic report of the project done by the team consisting of* ***Adwaith Jayan (Reg. No: KTE22CS006), Anand Sankar P T (Reg. No: KTE22CS018), Arjun Sabu (Reg. No: KTE22CS022) and Niranjan M Varma (Reg. No: KTE22CS057)*** *during the academic year* ***2024-25****, in partial fulfillment of the requirements for the award of the Degree of Bachelor of Technology in Computer Science and Engineering of APJ Abdul Kalam Technological University, Thiruvananthapuram.*

GUIDE COORDINATOR

HEAD OF THE DEPARTMENT

Acknowledgement

Every successful project is the outcome of hard work and strong support and guidance given by several people. We sincerely appreciate the inspiration, support, and guidance of all those people who have been instrumental in making this project a success.

We express our sincere gratitude to **Dr. Prince A., Principal, RIT Kottayam** for making the resources available at the right time, without which this project would not have been a success.

We take this opportunity to express a deep sense of gratitude to **Prof. Kavitha N., Associate Professor & Head, Department of Computer Science and Engineering**.

We also take this opportunity to express our profound gratitude and deep regards to our guide, **Prof. Anil Kumar S, Assistant Professor** for his exemplary guidance, monitoring, and constant encouragement throughout this project. The blessing, help, and guidance given by him from time to time shall carry us a long way in the journey of life on which we are about to embark.

We also express our gratitude to Project Coordinators **Prof. Anu Bonia Francis, Assistant Professor** and **gDr. Raji R. Pillai, Assistant Professor** for the cordial support, valuable information and guidance, which helped us in completing this task through various stages.

Last, but not the least, we thank **almighty, our parents** and **friends** for their constant encouragement without which this project would not have been possible.

Adwaith Jayan
Anand Sankar P T
Arjun Sabu
Niranjan M Varma

Declaration

We, the undersigned hereby declare that the project report entitled **Department Stock Management System-** ***A Web-Based Stock Management System***, submitted for partial fulfillment of the requirements for the award of the degree of Bachelor of Technology of the **APJ Abdul Kalam Technological University, Kerala** is a bonafide work done by our team under the supervision of **Prof. Anil Kumar S**. This submission represents our idea in our own words where ideas or words of others have not been included; we have adequately and accurately cited and referenced the original sources. We have adhered to the ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea in our submission. We understand that any violation of the above can result in disciplinary actions from the institute and/ or University and can evoke penal action from the sources which have not been properly cited or from whom proper permission has not been obtained. This report has not been previously formed as the basis for awarding any degree, diploma, or similar title of any other university.

Adwaith Jayan (Reg. No: KTE22CS006)

Anand Sankar P T (Reg. No: KTE22CS018)

Arjun Sabu (Reg. No: KTE22CS022)

Niranjan M Varma (Reg. No: KTE22CS057)

Batch: 2022 - 2026 April 1, 2025

Abstract

Department Stock Management System is a web-based application designed to streamline inventory management within the Computer Science and Engineering department. Managing stock efficiently is crucial for ensuring that laboratory equipment, furniture, and other assets are properly tracked, allocated, and maintained. Often, the lack of a systematic approach leads to stock misplacement, delays in transfers, and inaccurate records. Department Stock Management System provides a structured workflow where Principals, HODs, Stock-in-Charge, Custodians and Verifiers can collaborate to manage inventory effectively. With a user-friendly interface, the Department Stock Management System allows real-time stock updates, real-time stock status updates and complaint registration, verification of items, seamless stock transfers between premises, Real-time Notifications and automated report generation. Role-based access control ensures that only authorized personnel can update or approve stock changes, improving transparency and accountability. Department Stock Management System not only enhances inventory tracking but also optimizes resource utilization, reducing administrative overhead and making stock management more reliable and efficient.

Contents

List of Figures

List of Algorithms

List of Symbols and Abbreviations

ER	Entity Relationship
SRS	Software Requirement Specification
UML	Unified Modelling Language
UI	User Interface
API	Application Programming Interface
HTTP	HyperText Transfer Protocol
HOD	Head of Department
TSK	Technical Stock Keeper
CSE	Computer Science and Engineering
RIT	Rajiv Gandhi Institute of Technology
RBAC	Role Based Access Control

Chapter 1

Introduction

Effective stock management is crucial for the seamless functioning of any organization, especially in an academic setting where assets like laboratory equipment, furniture, and computing resources must be tracked, allocated, and maintained. Without a structured system, inventory mismanagement can result in misplaced stock, transfer delays, and inaccurate records, ultimately impacting resource utilization.
To overcome these challenges, we developed an integrated web application designed to streamline stock operations.Our project aims to eliminate inefficiencies, reduce administrative burdens, and enhance transparency, ensuring a more efficient and organized inventory management system for the CSE department at RIT, Kottayam.

1.1 Objectives

This project aims to achieve the following objectives.

1. To develop a web-based stock management system that streamlines inventory tracking, allocation, and maintenance within the department.
2. To establish a structured platform for effective collaboration between Principal, TSK, HOD, Stock-in-Charge, and Custodians in managing stock-related tasks.
3. To ensure transparency and accountability by implementing stock verification, approval workflows, and role-based access control.
4. To digitalize stock management processes such as stock allocation, verification, transfer, and clearance, minimizing manual efforts and errors.
5. To enable seamless reporting and analysis through automated report generation, providing valuable insights for decision-making.
6. To enhance communication and responsiveness by integrating real-time notifications for stock updates, approvals, and pending actions.

1.2 Motivation

The key factors that motivated the development of this project are as follows:

1. *Challenges in Manual Stock Management* – Manual stock handling leads to misplaced inventory, delays, and inaccurate records due to the absence of a centralized tracking system. A web-based system ensures efficient tracking, structured workflows, and transparency.
2. *Lack of Transparency and Accountability* – Without a proper verification and approval mechanism, stock updates and transfers may result in mismanagement. Implementing role-based access control ensures accountability among Principals, HODs, Stock-in-Charge, and Custodians.

3. *Delays in Stock Allocation and Transfer* – Traditional stock transfers involve manual approvals, paperwork, and communication delays. An digitalized allocation module with real-time notifications enhances efficiency.
4. *Difficulty in Stock Condition Tracking and Maintenance* – Tracking the working, damaged, or under-maintenance status of assets like lab equipment and furniture is challenging. The system allows real-time condition verification and timely maintenance updates.
5. *Need for Automated Reporting and Insights* – Decision-making is hindered by manual logging and report generation, which can be time-consuming and error-prone. This project introduces automated reporting, providing organized stock records, transfers, and verification insights.

1.3 Scope of the Project

- *Need*: The absence of a centralized system for managing inventory in academic departments leads to inefficiencies in stock tracking, verification, and transfers. A structured system is required to ensure transparency, accountability, and real-time updates.
- *Deliverables*: Through this web application, we intend to provide a web-based stock management system that enables Principals, HODs, Stock-in-Charge, and Custodians to efficiently track, allocate, transfer, and verify stock, with features like real-time notifications, digitalized reporting, and role-based access control.
- *Exclusions*:
  - External integrations with third-party inventory software are not included.
- *Assumptions*:
  - Users managing stock (Principals, HODs, Stock-in-Charge, and Custodians) act responsibly and accurately update inventory records.

1.4 Prerequisites for the Reader

To fully comprehend the contents of this report, the reader is expected to have the following prerequisites:

1. A fundamental understanding of software development concepts, including system analysis, design, implementation, testing, and deployment.
2. Basic knowledge of UML diagrams to effectively interpret the system architecture and workflows discussed in Chapter 3.
3. Familiarity with web development technologies, including React.js, Node.js, Express.js, and MongoDB, for a better grasp of the implementation details.
4. An understanding of role-based access control (RBAC) and its importance in managing stock authorization and approvals.

1.5 Organization of the Report

The remainder of the report is organized into six chapters. In Chapter 2, we detail the system study and the requirement engineering process. This includes the details of the existing systems (section 2.1), gap analysis (section 2.2), the proposed system (section 2.3), and those of the requirements engineering process (section 2.4), including feasibility study (section 2.4.1), requirements elicitation (section 2.4.2), requirements specification (section 2.4.4), and requirements validation (section 2.4.5). In Chapter 3, we detail the design of the proposed system, by giving a detailed discussion on system design (section 3.1) and the detailed design (section 3.2). Chapter 4 is intended to give the details of the implementation of the proposed system by providing an account of what tools are used for the implementation (section 4.1) and how each module is implemented using these tools (section 4.3). In Chapter 5, we discuss the different testing strategies used in this project (section 5.1) and the results of applying these strategies to the developed system (section 5.2). Chapter 6 is dedicated to presenting the results obtained (section 6.1) along with a result analysis (section 6.2). We close the report with chapter 7 which is a summary and recommendations for future works (section 7.1).

Chapter 2

System Study and Requirement Engineering

System study focuses on understanding the current system. This is the initial phase of system development where the current system, is thoroughly analyzed to identify problems, inefficiencies, and areas for improvement.

Requirement engineering focuses on defining the needs and constraints of stakeholders and translating them into unambiguous requirements for the new system. Both processes are essential for the successful development and implementation of software system

2.1 Existing Systems

Existing systems are those systems that are currently being used to perform the task that comes under the objectives of this project. In this phase, we study these systems to see how they work and what problems they have. This analysis helps us identify features to be included in our system to make it more efficient. By studying these systems, a better system can be implemented.

2.1.1 Manual Stock Register System

The manual stock register system is a traditional method used by institutions to keep track of department assets like computers, furniture, and lab equipment. It primarily involves maintaining records in physical logbooks or spreadsheets. The following are the key aspects of this system

- Stock details are recorded manually, making the process tedious and inefficient.
- Data retrieval requires searching through physical registers or scattered files, leading to delays.
- Updates to stock records must be made manually, increasing the chances of inconsistencies.
- Tracking stock movement between locations relies on manual logs, making verification cumbersome.
- Periodic audits involve cross-checking paper records, making stock verification time-consuming.
- Notifications regarding stock updates or shortages are not automated, requiring manual communication.

2.1.2 Literature Review

A literature survey for a project involves thoroughly researching and reviewing existing literature, such as scholarly articles, books, research papers, and other relevant sources, that pertain to the topic of the project. It helps the researcher to gain a comprehensive understanding of the existing theories, methodologies, and practices related to the project's subject matter.

The study by Hemant Kumar [1] explores the impact of Inventory Management Systems (IMS) in educational institutions, emphasizing their role in minimizing errors in stock reporting and streamlining inventory transactions. The research highlights how traditional inventory methods often lead to inconsistencies and inefficiencies, making it difficult for institutions to track stock levels accurately. By implementing an automated IMS, educational organizations can benefit from real-time stock updates, improved resource allocation, and enhanced operational efficiency. The findings reinforce the need for digitized inventory tracking, aligning with the objectives of our Department Stock Management System, which seeks to optimize inventory accuracy, stock movement, and accountability within academic environments.

A case study on Sparrow Academy [2] highlights the impact of implementing an inventory and asset tracking system in an educational institution. The system provided a centralized platform to manage both inventory and fixed assets, significantly improving operational efficiency by automating stock tracking, reducing manual errors, and optimizing resource allocation. By offering configurable features tailored to the academy's needs, the system enabled real-time visibility into stock levels, automated process workflows, and enhanced decision-making based on data insights. The study demonstrates how digital inventory solutions can transform stock management in academic environments, aligning with our Department Stock Management System, which aims to improve inventory transparency, accountability, and streamlined operations through technological advancements.

The study by Soegoto and Palalungan [3] presents a web-based online inventory information system designed to maximize human performance in organizational processes. The research emphasizes the role of digital systems in streamlining inventory management tasks, reducing manual workload, and enhancing data accuracy. By adopting such a system, organizations can achieve more efficient inventory control, timely reporting, and improved decision-making capabilities. These findings are relevant to our Department Stock Management System, which aims to leverage web-based technologies to optimize inventory management and operational efficiency.

2.2 Gap Analysis

Some potential drawbacks were identified in each of the above-mentioned existing systems. They are discussed in the following.

1. *Prone to errors and inconsistencies*:- Manual record-keeping and spreadsheet-based systems often lead to data entry mistakes, duplication, and inconsistencies, affecting the reliability of stock records.

2. *Lack of real-time stock updates*:- Existing methods do not provide instant stock updates, leading to delays in tracking available inventory and making informed decisions.

3. *Difficulty in tracking stock movement*:- Traditional systems fail to efficiently monitor the transfer of stock between different locations, increasing the risk of misplaced or unaccounted items.

4. *No automated notifications or alerts*:- The existing system lacks automated notifications for stock allocation, verification assignments, and transfer approvals, requiring manual follow-ups and increasing delays.

5. *Time-consuming verification and reporting*:- Annual stock verification and report generation require extensive manual effort, leading to delays and inaccuracies in inventory audits.

6. *Limited role-based access control*:- Many existing systems lack proper access control mechanisms, allowing unauthorized modifications and reducing accountability in stock management.

2.3 Proposed System

The proposed system, Department Stock Management System facilitates efficient tracking, allocation, and verification of inventory within educational institutions. It prioritizes accuracy, accessibility, and security, ensuring seamless stock management through a centralized and automated platform. With role-based access control, real-time updates, and automated reporting, the system enhances accountability and transparency in inventory handling. By streamlining stock transactions and verification processes, the Department Stock Management System aims to eliminate manual errors, reduce inefficiencies, and optimize resource utilization, creating a more structured and reliable inventory management framework.

2.3.1 Problem Statement

To develop a web-based stock management system for the Computer Science and Engineering department of RIT Kottayam.

2.3.2 System Model

The model of the proposed system is shown in Figure 2.1. The system consists of six main user groups:

1. *Principal*: The principal has the highest level of authority in the system. They can approve stock verification reports, assign faculty for stock verification, and review overall inventory status.

2. *Head of Department (HOD)*: The HOD controls all stock-related activities within the department, including assigning faculty members, creating new stock systems, allocating stock to specific premises, and managing user access permissions.

3. *Stock-In-Charge*: Faculty members responsible for managing stock within their assigned premises. They can add new stock, update stock details, remove items from inventory, transfer stock and generate reports.

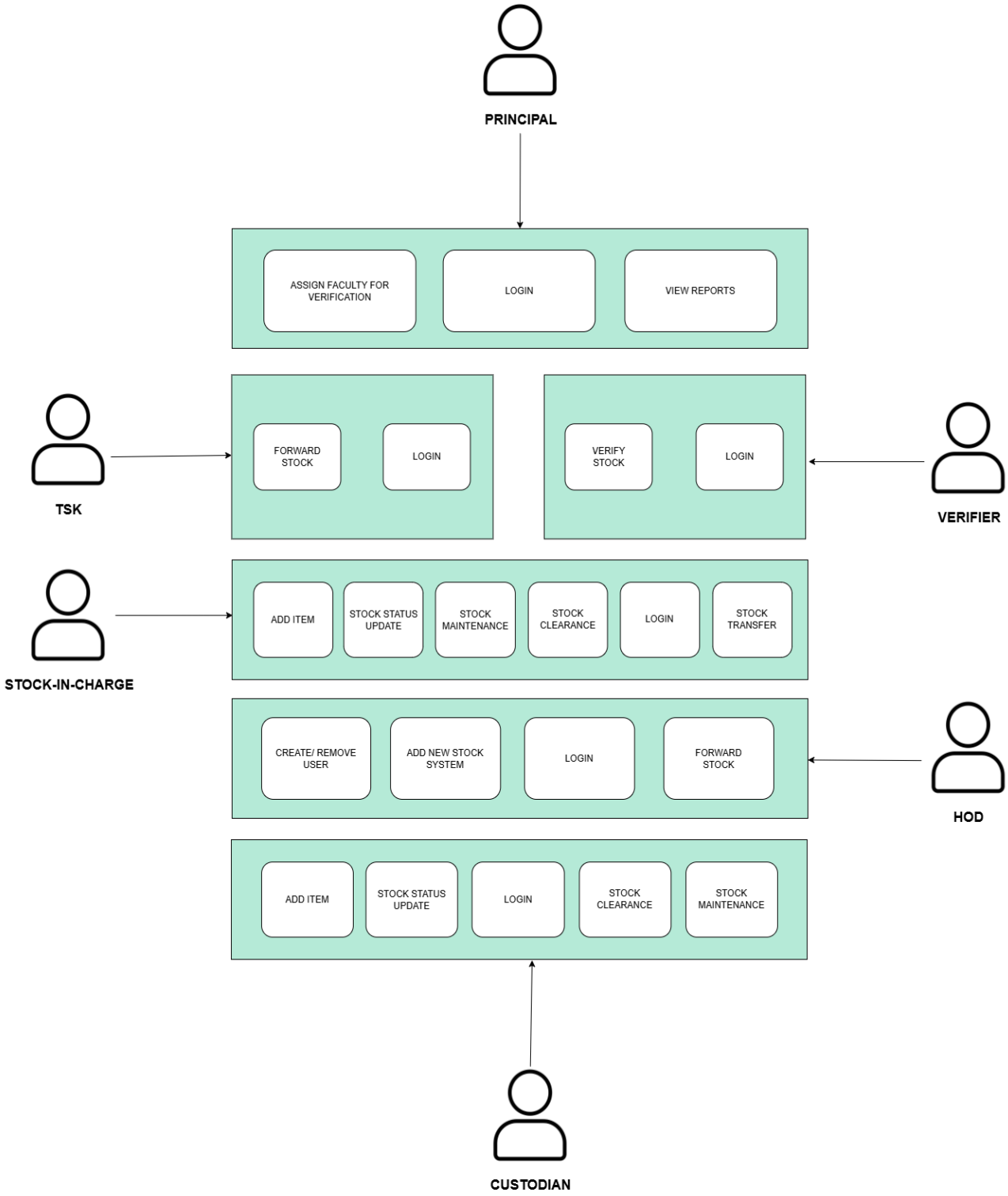


Figure 2.1: System Model

4. *Custodian*: Custodians are responsible for handling and maintaining stock in their assigned department. They assist the Stock-In-Charge in inventory management tasks, register complaints for maintenance, and report damaged stock.

5. *Technical Stock Keeper (TSK)*: This role is responsible for initiating stock allocation process, managing stock transfers, and keeping records of stock movements. The TSK also ensures that inventory is properly maintained and allocated to the correct premises.

6. *Faculty (Verifier)*: Faculty members assigned temporarily to verify inventory. They are responsible for checking stock availability, ensuring records match physical inventory, and submitting verification reports to the Principal.

2.4 Requirements Engineering

Collecting software requirements from clients, analyzing, and documenting them is termed *Requirements Engineering*. Its objective is to create and sustain a detailed *System Requirements Specification* document. This process primarily comprises the following five key activities.

1. Feasibility Study
2. Requirements Elicitation
3. Requirements Analysis
4. Requirements Specification
5. Requirements Validation

2.4.1 Feasibility Study

A Feasibility Study is an evaluation conducted to assess the viability of a proposed project or system. It involves an analysis of the practicality and benefits of developing the chosen software. The key considerations involved in the feasibility analysis are

1. Economic Feasibility
2. Technical Feasibility
3. Behavioral Feasibility

2.4.1.1 Economic Feasibility

In an economic feasibility study, the total cost of developing the proposed system including expenses for hardware, software, design, development, and operations is evaluated to determine its alignment with the project's financial constraints. The proposed system is a web-based application, and while most development resources, such as frameworks and tools, are open-source or freely available, there are significant costs associated with hosting the application and managing the database on MongoDB Atlas. However, by opting for cost-effective hosting and database plans, the team has ensured that expenses remain manageable and within budget. Therefore, the proposed system is economically feasible.

2.4.1.2 Technical Feasibility

In a technical feasibility study, the available resources and technologies are assessed to determine if they are sufficient for developing the proposed system. It also evaluates the technical skills of the development team and analyzes whether maintenance and future upgrades can be efficiently managed with the chosen technology stack.

The proposed system is a web-based application developed using the MERN stack (MongoDB, Express.js, React, and Node.js). These technologies are widely used, well-documented, and supported by a large developer community, ensuring smooth development and long-term maintainability. While hosting and database management require technical expertise, the team possesses the necessary skills to handle deployment, security, and scaling. Therefore, the proposed system is technically feasible.

2.4.1.3 Behavioural Feasibility

In a behavioral feasibility study, the focus is on assessing whether new users can smoothly adopt and effectively use the proposed system. This involves evaluating how well the system meets user requirements, its ease of use and maintenance, and overall user acceptance. Additionally, it considers usability factors and whether the recommended implementation approach aligns with user expectations.

Since the proposed system is a web-based application with a user-friendly interface, users can quickly adapt to it without requiring extensive training. Basic functionalities like stock management, verification, and tracking are designed to be intuitive and accessible for faculty and staff. Moreover, the system streamlines existing manual processes, making adoption easier. Therefore, the proposed system is behaviorally feasible.

2.4.2 Requirements Elicitation

Requirements Elicitation is the process of gathering and defining the requirements for a software system. The following methods were used to collect requirements for our project.

1. *Studying existing system*:- The existing system was studied in detail and the advantages of the system were identified. A gap analysis was conducted to identify the common disadvantages of the system and potential shortcomings. These details are made available in section 2.1 and section 2.2 of this report.

2. *Field study*: Field Study Method involves gathering information through direct interactions and observations within the relevant environment. Requirements were gathered through direct interactions with faculty members responsible for stock management in the department. These discussions provided valuable insights into their workflow, challenges, and specific inventory needs. Additionally, our project guide, who has hands-on experience with stock duties, contributed critical input, ensuring the system aligns with the department's operational requirements and feasibility.

2.4.3 Requirements Analysis

The data collected from the field study and the analysis of the existing system were thoroughly examined, and the essential information was carefully shortlisted. Based on this analysis, the following requirements were finalized for the development of the application.

1. *User Authentication*:- Ensures secure system access by verifying user credentials based on their roles. Failed login attempts trigger alerts after a certain threshold.
2. *Stock Allocation*:- Facilitates the allocation of inventory items to specific department premises. The system notifies the relevant personnel upon successful allocation.
3. *Stock Verification and Remarks*:- Enables faculty to verify assigned inventory items and submit annual stock verification reports. The system updates item conditions and logs remarks for future reference.
4. *Stock Transfer*:- Allows stock-in-charge to transfer items from one premise to another. The destination stock-in-charge is notified to approve or reject the transfer, and stock records are updated accordingly.
5. *Stock Maintenance*:- Ensures proper logging of maintenance details for inventory items. Stock-in-charge can notify relevant personnel about items needing maintenance, and updates are recorded after repairs.
6. *Stock Clearance*:- Enables custodians to remove damaged or obsolete items from the inventory. Cleared items are logged in the system, and notifications are sent to relevant stakeholders.
7. *Stock Handover*:- Allows faculty members to hand over stock responsibilities to another faculty. The system generates notifications and ensures a seamless transition.
8. *User-Based Notifications*:- Keeps users informed about assigned tasks, approvals, stock movements, and verification updates via dashboard notifications.
9. *Report Generation*:- Provides users with detailed reports based on filters like item type, condition, and location. Reports can be exported in PDF or Excel format.
10. *Add a Stock System*:- Enables the HOD to create a new stock database for a newly added department premise. The system assigns a stock-in-charge and custodian for managing the new inventory.

2.4.4 Requirements Specification

A well-defined set of requirements is the foundation for any successful software development project. Requirements specifications serve as the blueprint for the system, outlining its functionalities, behavior, and performance expectations. Requirements are divided into *functional requirements* and *non-functional requirements* [4, 5]. Functional requirements specify the system's actions, like data handling, and user interactions. Non-functional requirements [4, 5] describe how quickly the system should respond when its functionality is used and address broader qualities such as performance, security, usability, and scalability. The SRS [4, 5] document is given in the Appendix A.

2.4.4.1 Functional Requirements

1. *User Authentication*
    - Ensures secure access by validating user credentials (email id and password).

- Triggers alerts after multiple failed login attempts.

2. *Stock Allocation*

   - The HOD must be able to forward the notification send by the TSK to stock-in-charge of respective premises.
   - The stock-in-charge must be able to forward the notification send by the HOD to the respective custodian.
   - The custodian accepts the notification from the stock in charge and inserts the items into the respective stock inventory.
   - The stock notifications must be send to stock in charge, HOD and TSK to notify that the inventory items are successfully allocated to the respected premises.

3. *Stock Verification and Remarks*

   - Principal must be able to assign a faculty for verifying the stock and send a notification to the respective persons.
   - The HOD must be able to create a temporary login credentials for each faculty who were assigned stock verification duty by the principal and to send these details to the respective faculty.
   - Each Faculty who was assigned the stock verification duty must be able to view the items of the respective premises and must be able to mark the present status of the items (working, damaged, or under maintenance). Further, on completion of the status entries, the faculty must be able to submit the inferences to the principal.
   - The Principal must be able to access the inferences sent by the faculty who were assigned stock verification duty as separate reports and approve them.
   - The HOD must be able to delete the temporary user credentials on completion of the respective stock verification and the approval of the principal.

4. *Stock Transfer*

   - The stock-in-charge must be able to select the destination premise and specify the item to be transferred.
   - The stock-in-charge of the destination premise must receive the notification to accept or reject the item.
   - A provision to update the stock of both the premises if the stock-in-charge of the respective premise accepts the notification.
   - A provision to send notifications regarding the transfer to the custodians of both the premises.

5. *Stock Maintenance*

   - There must be a provision to send an email to the service provider.
   - Completed task should update the item status and maintenance details in inventory database.

6. *Stock Clearance*
    - The stock-in-charge of the respective premise must be able to identify and select items that are eligible for clearance.
    - Once cleared, the system should automatically update the respective stock records to reflect the removed items.
    - A provision to send notifications to relevant parties.
    - A data storage to keep up the records of all the cleared items.
7. *Stock Handover*
    - The party who wants to transfer the responsibility must be able to send a notification to HOD.
    - The HOD must be able to appoint a new faculty as stock-in-charge.
    - The HOD must be able to create a temporary login credentials for the new faculty.
    - The HOD must be able to remove the login credentials of the previous stock-in-charge.
    - Notifications regarding the stock handover must be sent to both the parties.
8. *User-Based Notifications*
    - Notifications should be available on the user dashboard.
    - Alerts must include task details, user assignment updates, and approval updates.
9. *Report Generation*
    - The Users must be able to select various filter options including the item type, condition (working, damaged, under maintenance), location and other specifications.
    - Reports should support export in formats like PDF and Excel for offline use.
10. *Add a Stock System*
    - The system must allow the creation of a new data storage for recording the details of the new stock system.
    - The HOD must be able to assign stock-in-charge and the custodian for the new stock system.

2.4.4.2 Non-functional requirements

1. Performance Requirements
    - *Response time*: The system must provide low response time, ensuring that web server requests are processed efficiently for a seamless user experience.
    - *Efficient data processing*: Database operations should be optimized for quick retrieval and updates, especially for stock verification, allocation, and reporting.
    - *Cross-platform compatibility*: The system must function smoothly on different web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge.

2. Safety Requirements

   - *Protection of data against unauthorized access*: Implement robust authentication and authorization mechanisms, ensuring that only authorized personnel can access stock records.
   - *Secure data storage*: Ensure that inventory records and user data are stored securely with regular backups to prevent loss due to system failures or cyber threats.
   - *System integrity*: Implement access logs to monitor system activities and detect unauthorized modifications to stock records.

3. Security Requirements

   - *Access Control*: Restrict system operations based on user roles (Principal, HOD, Faculty In-Charge, Custodian, etc.) to prevent unauthorized modifications.
   - *Data Encryption*: Store sensitive data, such as passwords, in a hashed and salted format to enhance security.
   - *Audit Logs*: Maintain a record of all system activities, including stock modifications, transfers, and verifications, to ensure accountability.
   - *Secure Communication*: Ensure encrypted data transmission between the client and server to prevent unauthorized access during transit.

4. Software Quality Attributes

   - *Scalability*: The system should be designed to accommodate increased users and services.
   - *Maintainability*: Ensure the system facilitates easy maintenance and updates.
   - *Usability*: Provide a user-friendly interface to enhance user experience.
   - *Accessibility*: Ensure the application is accessible to users from anywhere, anytime.
   - *Reliability*: The system should be dependable and available under all circumstances.

5. Other Requirements

   - *Adherence to institutional policies*: The system must comply with the stock management policies and guidelines set by the institution to ensure proper inventory handling.

2.4.5 Requirements Validation

Requirement validation ensures the specified requirements are thorough and consistent with user needs. It verifies that the requirement specifications adhere to defined quality standards. The process involves a systematic manual analysis of the project's needs.

Considering the functional requirements collected and framed, this system can be implemented to meet the objectives of the project. The non-functional requirements are also verified to meet the defined quality standards.

2.5 Summary

Chapter 2 analyzes existing stock management systems, highlighting inefficiencies in tracking and transparency. The proposed system enhances stock allocation, verification, and transfers based on faculty input. Key features include user authentication, stock tracking, reporting, and role-based access. The chapter also outlines security, scalability, and maintainability considerations.The next chapter, *System Design and Methodology*, details the architectural and functional aspects of the system.

Chapter 3

System Design and Methodology

The design phase is where the project's vision takes shape, defining its direction, structure, and the steps needed to bring it to existence. This phase involves planning, brainstorming, and creating a detailed blueprint that guides the entire development process. It acts as a roadmap ensuring the final product fulfills its intended purpose.

The design process unfolds in two key stages, each with a distinct focus:

- *System Design*: This stage sets the big picture, outlining the overall structure and components of the project.
- *Detailed Design*: Here, the focus narrows down to the specifics, creating a comprehensive blueprint with all the intricate details needed for successful execution.

3.1 System Design

There are 3 modules in the proposed system,

1. *Front-end Module* :- This module provides the UI for the users to interact with the web app.
2. *Database Module* :- This module provides data for each functionality in the web app.
3. *Connection Module* :- This module connects the Front-end module and the database module.

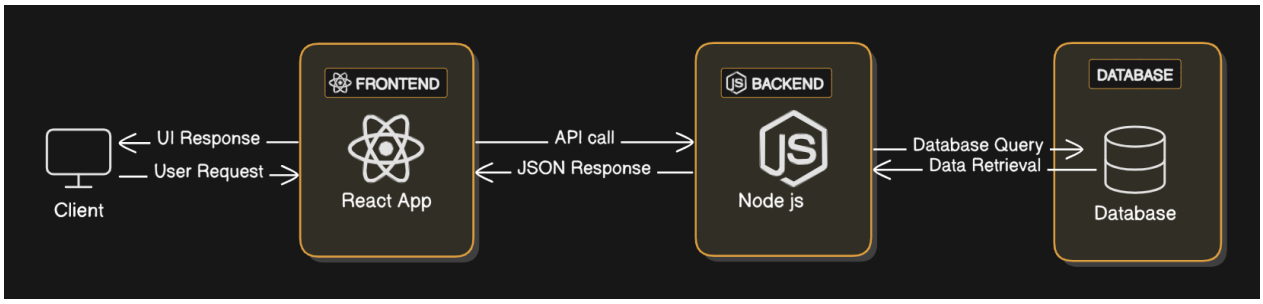


Figure 3.1: Common Architecture Diagram

3.1.1 Module 1: Front-end Module

The Front-end module includes the following pages for user interaction.

1. *Log-in page*: The entry point for users, where TSK,Principals, HOD, Stock-in-Charge, Custodians, and Verifiers can log in with their credentials for secure access.
2. *Register page*: This page is designed to input user details for registration of faculties by HOD.

3. *Dashboard Page*: A role-based dashboard providing personalized access to features based on user roles (e.g., stock approvals by TSK, stock verification for Verifiers, stock management for Stock-in-Charge and Custodians).

4. *Stock Details Page*: Displays a comprehensive list of all inventory items, including their current status,location, Specification and condition.

5. *Stock Status Update Page*: Allows Stock-in-Charge and Custodians to update stock conditions (e.g., working, damaged, under maintenance) and add remarks for verification.

6. *Stock Transfer Page*: Enables Stock-in-Charge to transfer stocks between different premises.

7. *Stock Warranty Page*: Used by Stock-in-Charge and Custodians to track warranty details of equipment and request warranty-based repairs or replacements.

8. *Add Stocks Page*: Allows authorized users (Stock-in-Charge) to add new stock items, specifying details such as item type, quantity, purchase date.

9. *Faculty Assign page*: Allows principal to assign faculty for verification of stocks for the department.

10. *Verification page*: Allows verifier to verify stock of the assigned premise.

11. *Maintenance Page* – Logs repair and maintenance requests for faulty inventory items. Stock-in-Charge and Custodians can register complaint to service provider, update maintenance status, ensuring timely repairs and tracking of maintenance history.

12. *Stock Clearance Page* – Used for managing stock disposal or clearance of outdated, broken, or unusable items. Stock-in-Charge can clear the items approved by verifier at any time .

13. *Send Email Page* – Facilitates email notifications for stock-related requests, ensuring smooth communication between users.

14. *Notification Page* – Displays real-time alerts related to stock verification, approvals, transfers, and maintenance updates, ensuring all users stay informed of critical actions.

3.1.2 Module 2: Database Module

The database module includes the following Entities and Relations

1. *User*: This entity includes attributes user ID, username, password, phone no, location, Email, and Document. The user ID and username together are identified as the key for this entity.

2. *Admin*: This entity includes attributes admin ID and name. Used to store the details of the administrators.

3. *Service*: This entity includes attributes service ID and service name. An admin can add services.

4. *Post*: This entity includes attributes post ID, description, document, date, and image. Each post corresponds to a service and is posted by some user.

5. *Committed*: This entity is a relation from one user to another user. Used to denote if two users are committed.

6. *Feedback*: This entity is a relation from a user to another user. Used to store the feedback details about one user from another user.

7. *Complaint*: This entity is a relation from a user to another user. Uses the store complaint details about one user from another user.

3.1.3 Module 3: Connection Module

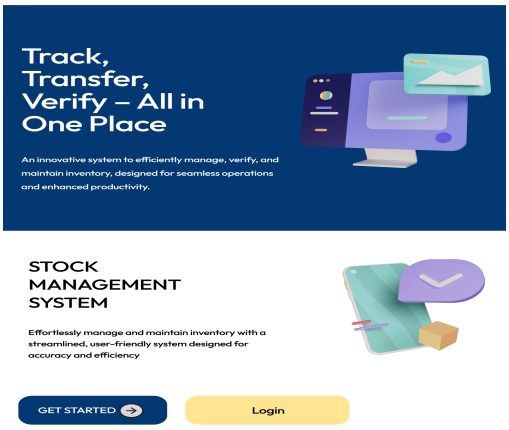
The connectivity module acts like a bridge, allowing the user-facing front end and the data-handling backend to communicate to each other. This module focuses on three aspects: sending data efficiently, keeping it safe, and enabling smooth communication. It specifies exactly how data will be formatted and sent (APIs), how users will be verified (authentication), and how to fix any problems that might arise (error handling).

3.2 Detailed Design

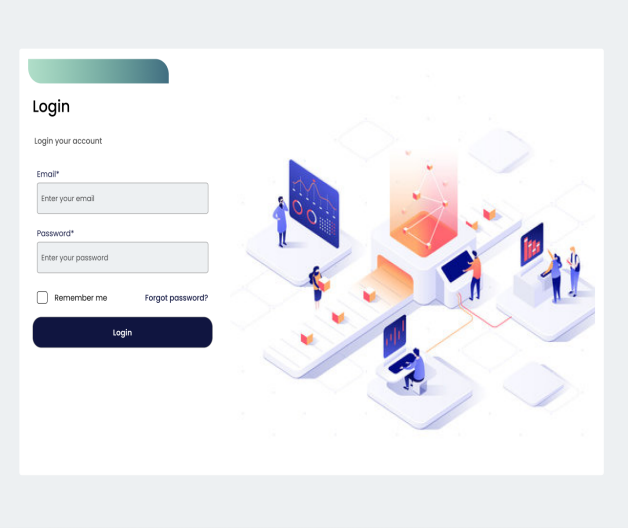
In this section, we present the detailed design, where we specify the internal details of each of the three modules. To specify the detailed design of the Connection module, we have used the State Chart Diagram for each of the functionalities in the module. Furthermore, we have given the respective algorithm for each of these functionalities from the implementation perspective.

3.2.1 Detailed design of Front-end module

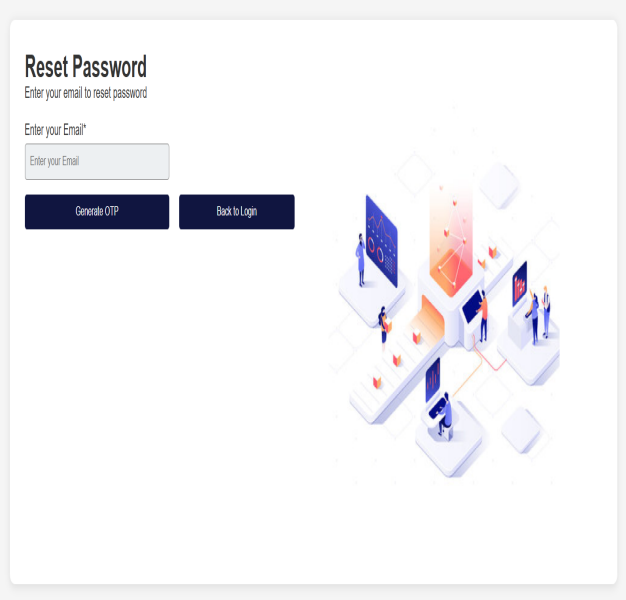
The following are the design of the various pages mentioned in section 3.1.1.



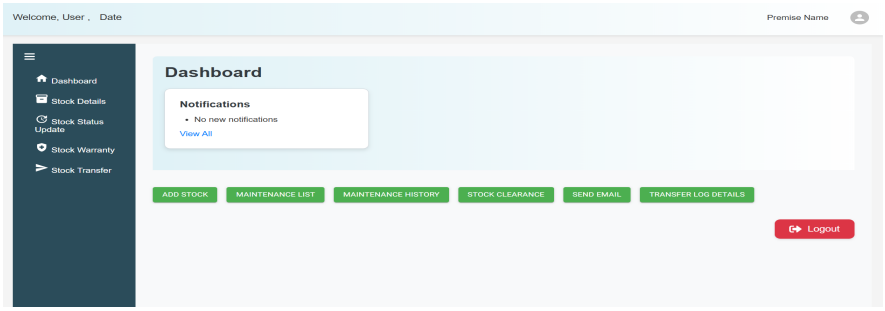
(a) Home page



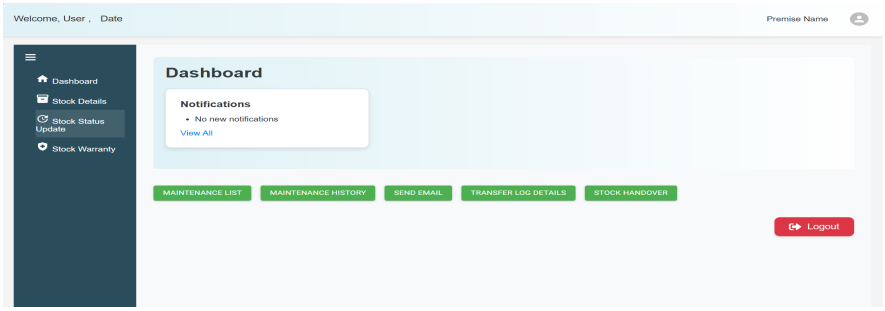
(a) User Login page



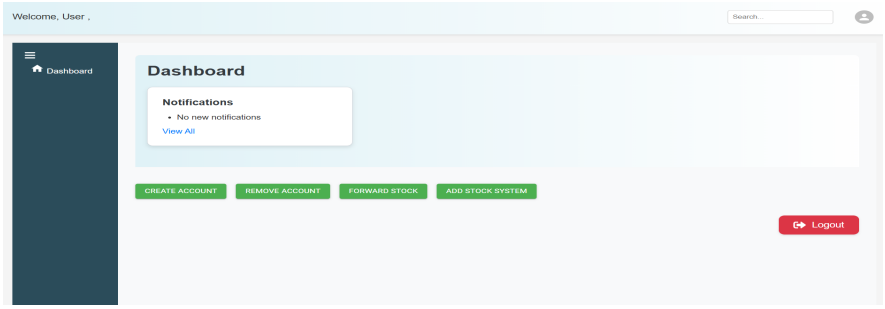
(b) Reset Password Page



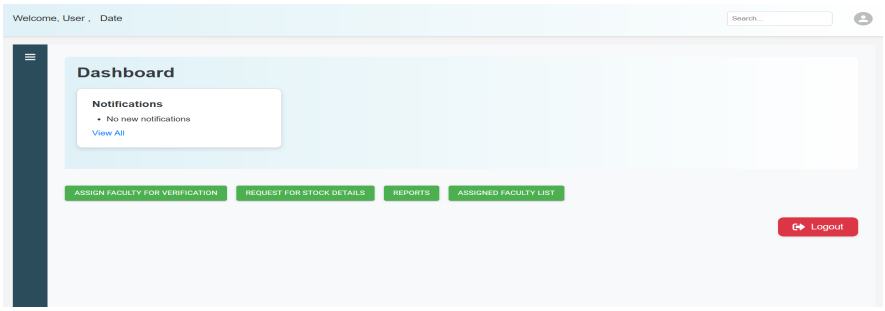
(a) Dashboard for Stock-In-Charge



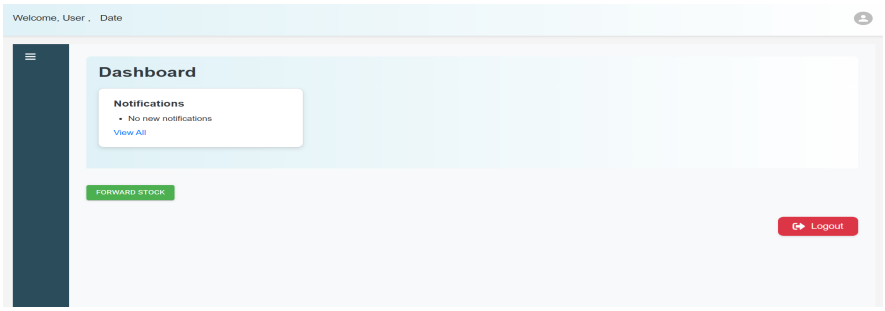
(b) Dashboard For Custodian



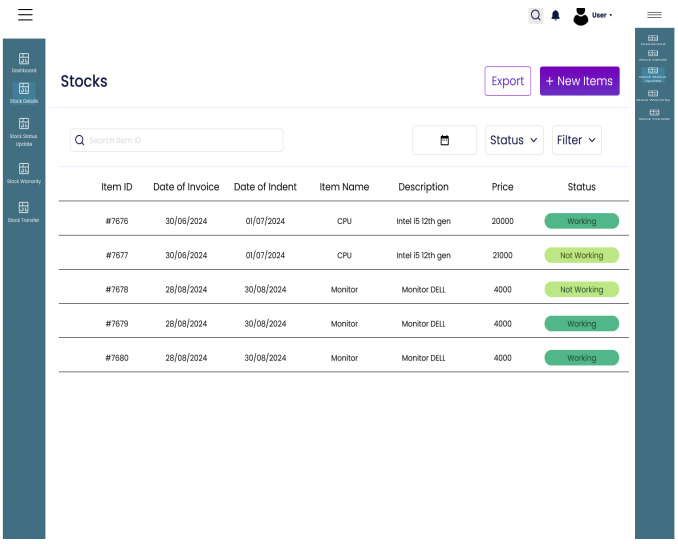
(c) Dashboard For HOD



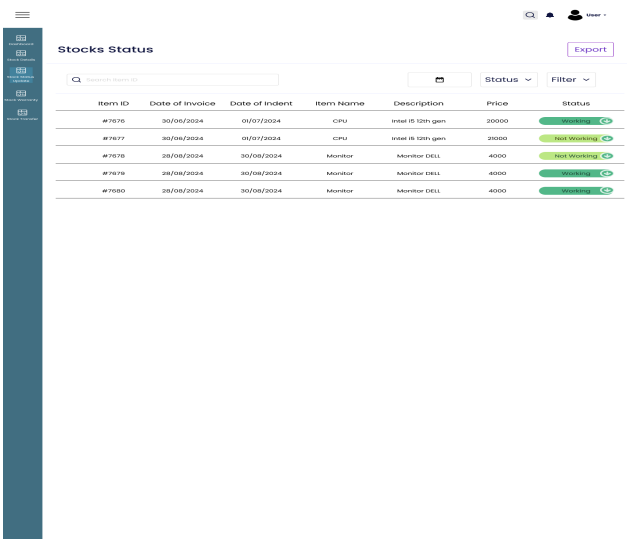
(d) Dashboard For Principal



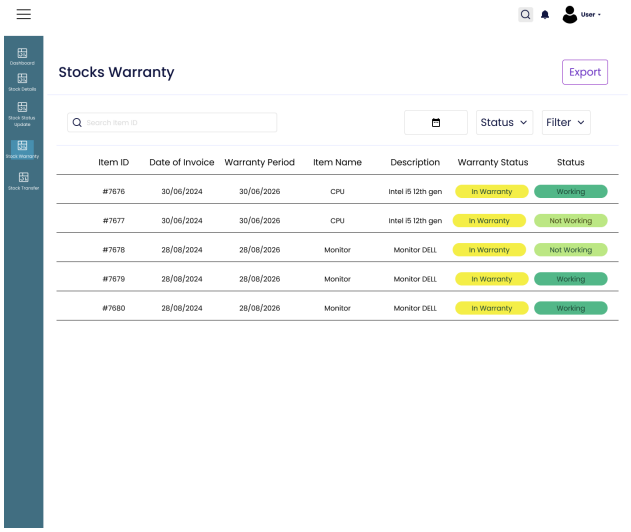
(e) Dashboard For TSK



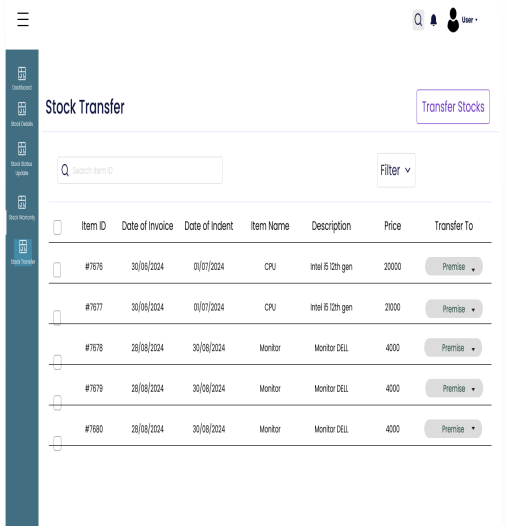
(a) Stock Details page



(b) Stock Status page

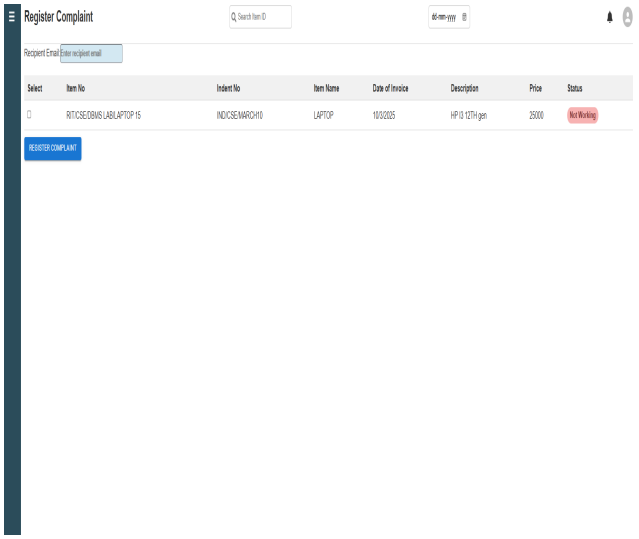


(c) Stock Warranty page

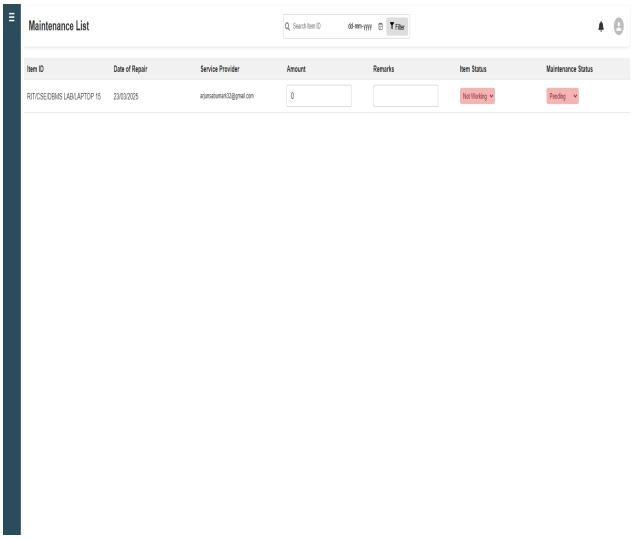


(d) Stock Transfer page for Stock-In-Charge

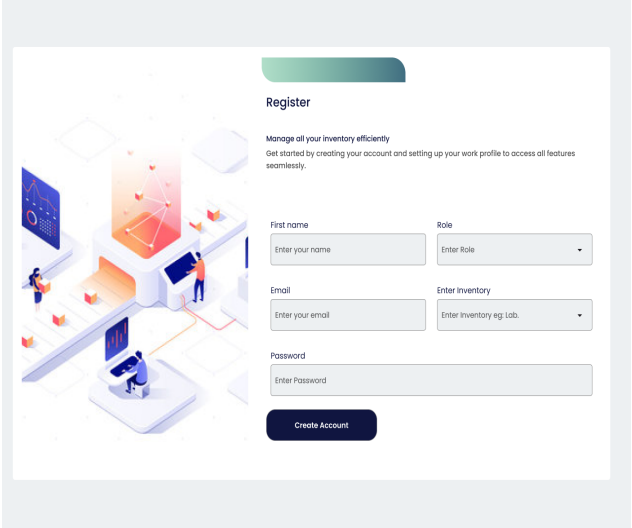
(a) Add stock page for Stock-In-Charge



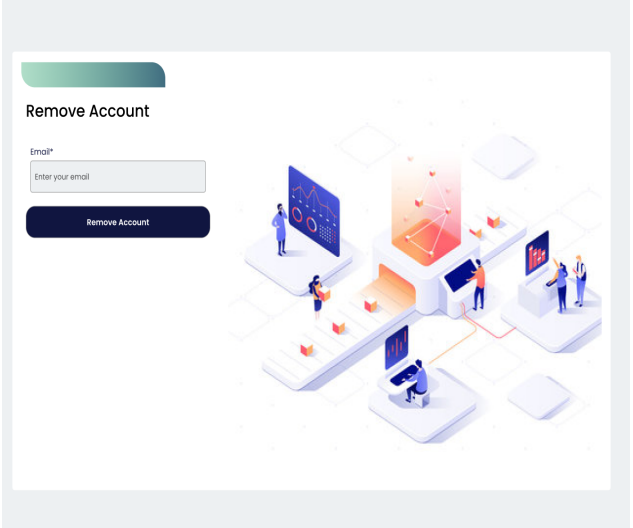
(a) Complaint Register page



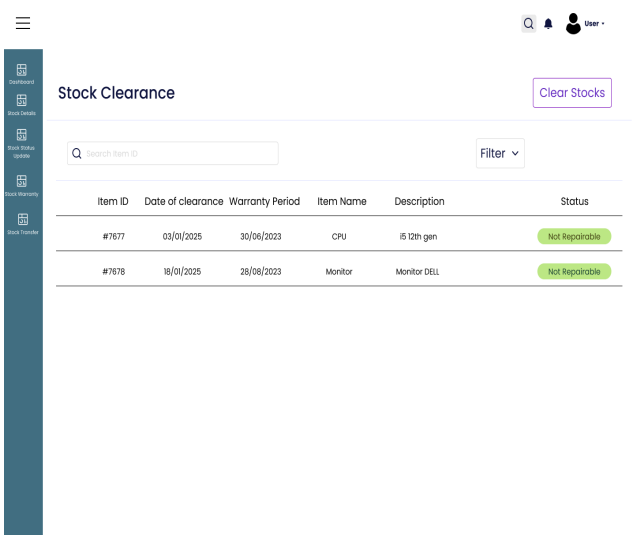
(b) Maintenance List page



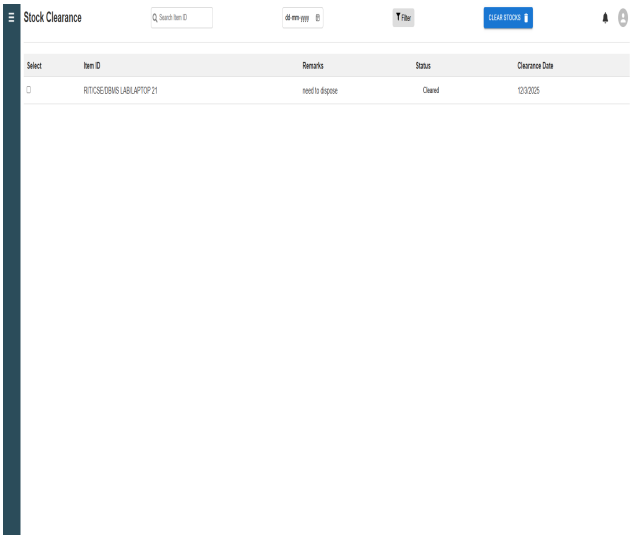
(a) Register Faculty page for HOD



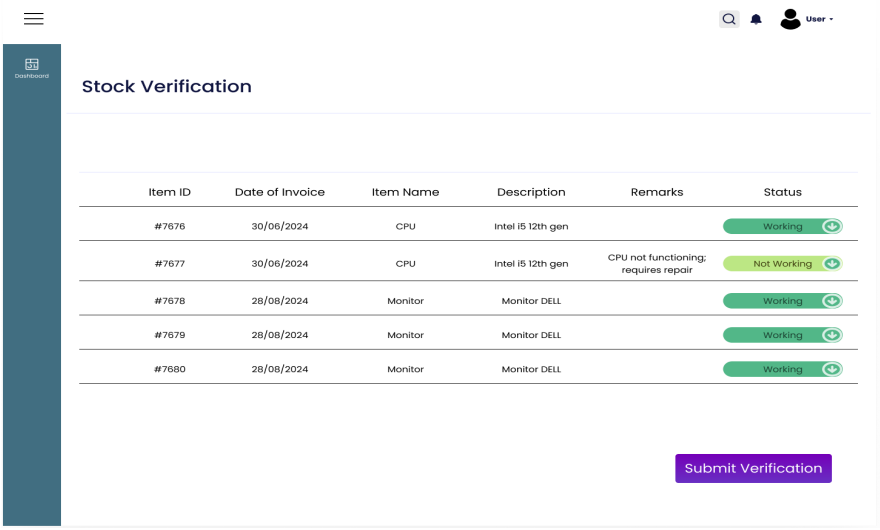
(b) Remove Account page for HOD



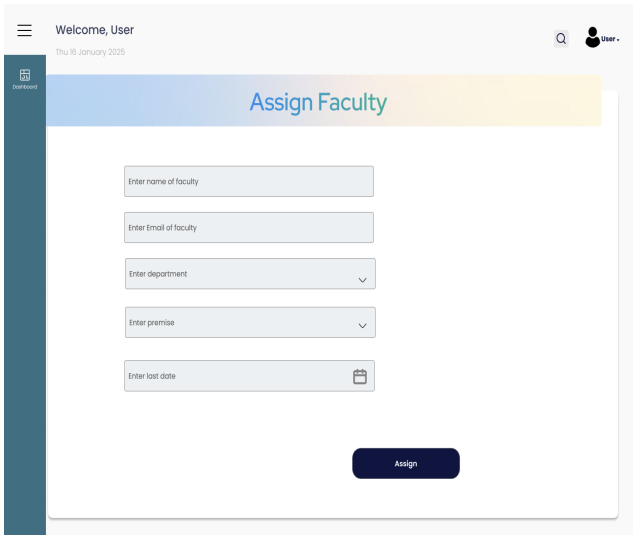
(a) Clearance page for Stock-In-Charge



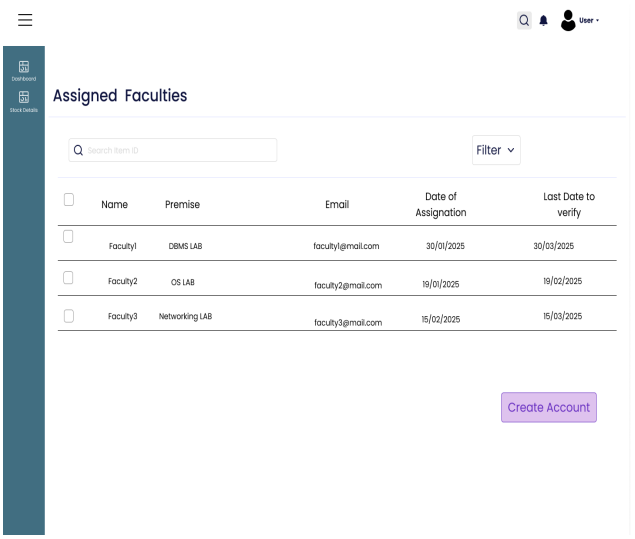
(b) Clearance List page for Stock-In-Charge



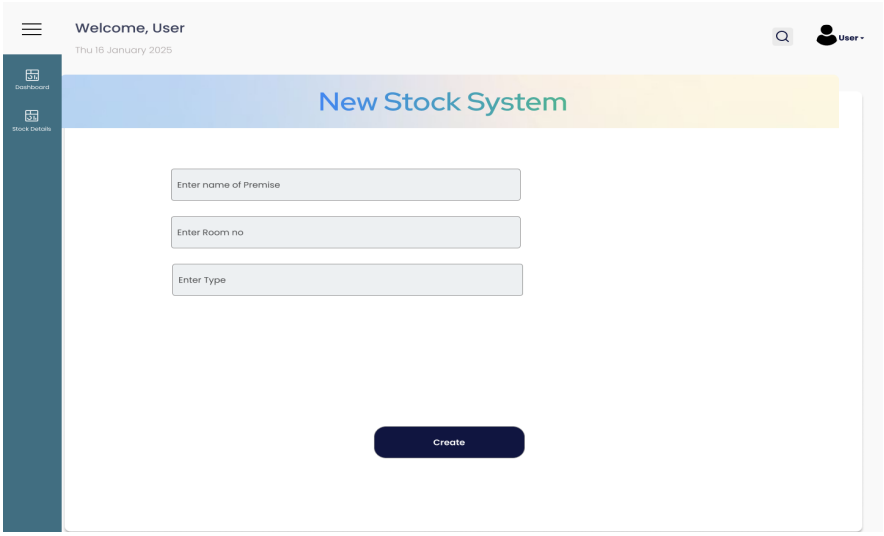
(a) Stock Verification page



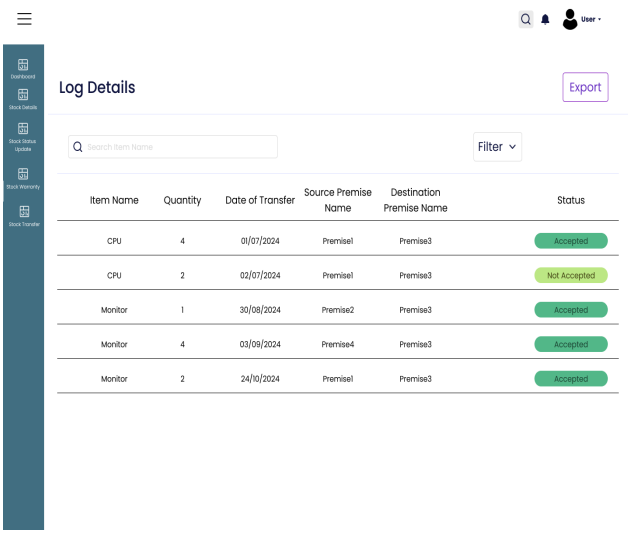
(a) Assign Faculty page for Principal



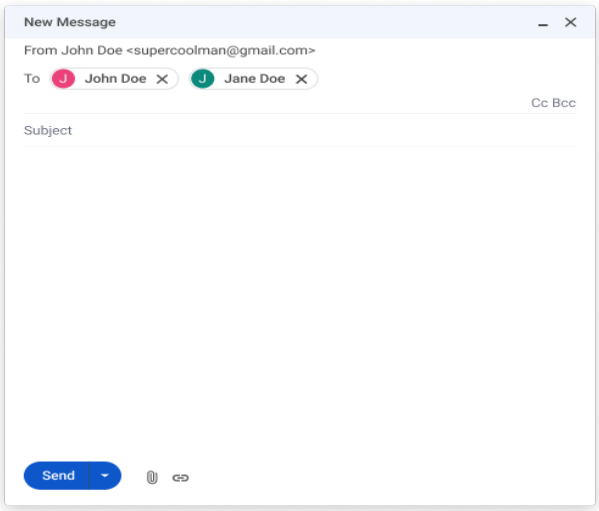
(b) Assigned Faculty List



(a) Create new stock system page for HOD



(a) Transferred Stock List



(a) Email page

3.2.2 Detailed design of Database module

The Detailed Design of the Database module is given in the ER Diagram shown in Figure 3.15. It consists of the different entities along with the attributes specified in section 3.1.2.

3.2.3 Detailed design of Connection module

In this section, we describe the detailed design of the connection module by specifying the state chart diagram and the algorithm for each functional requirement.

3.2.3.1 State chart diagram and algorithm of User Authentication

The State Chart Diagram for User Authentication is given in figure 3.16.

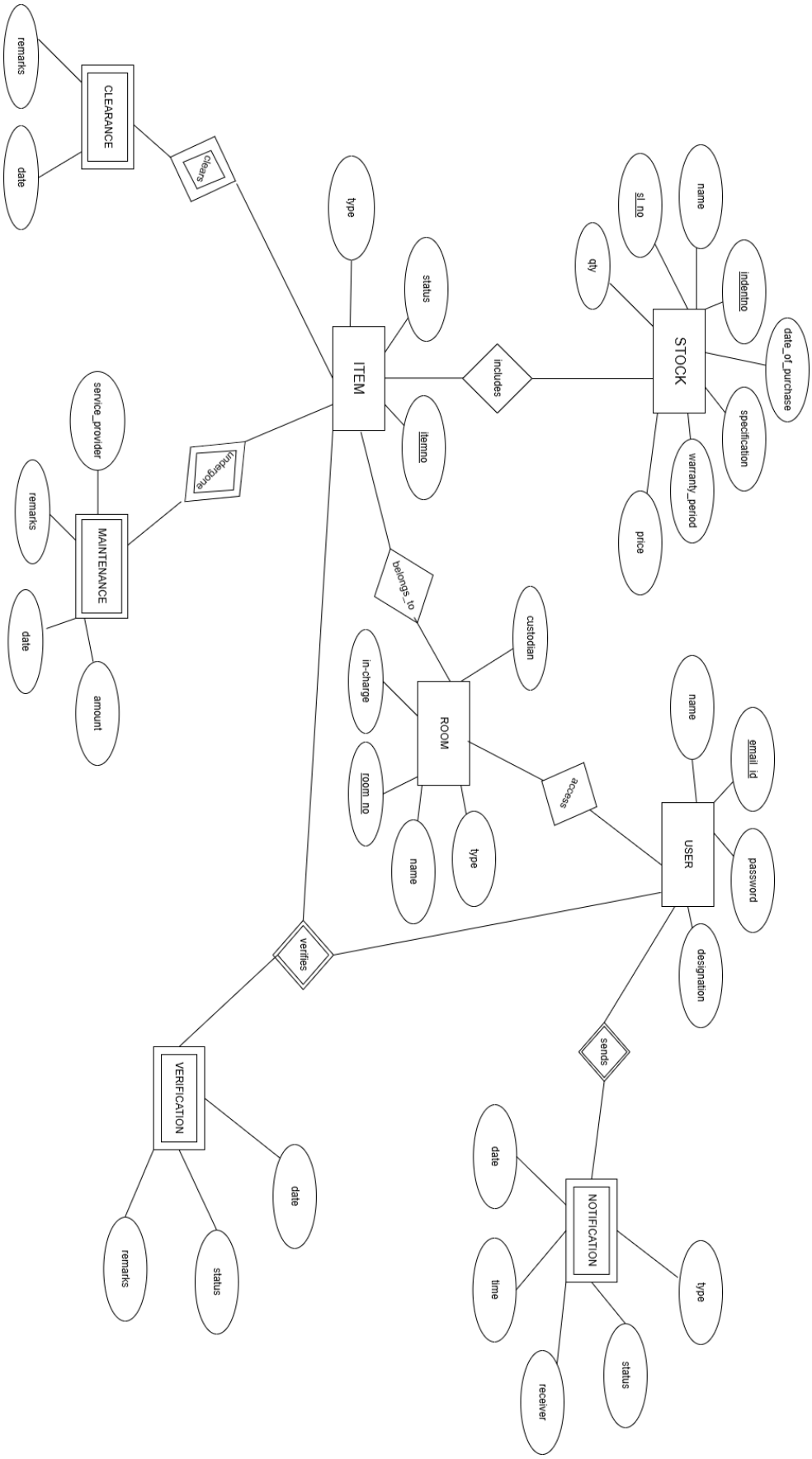


Figure 3.15: ER diagram

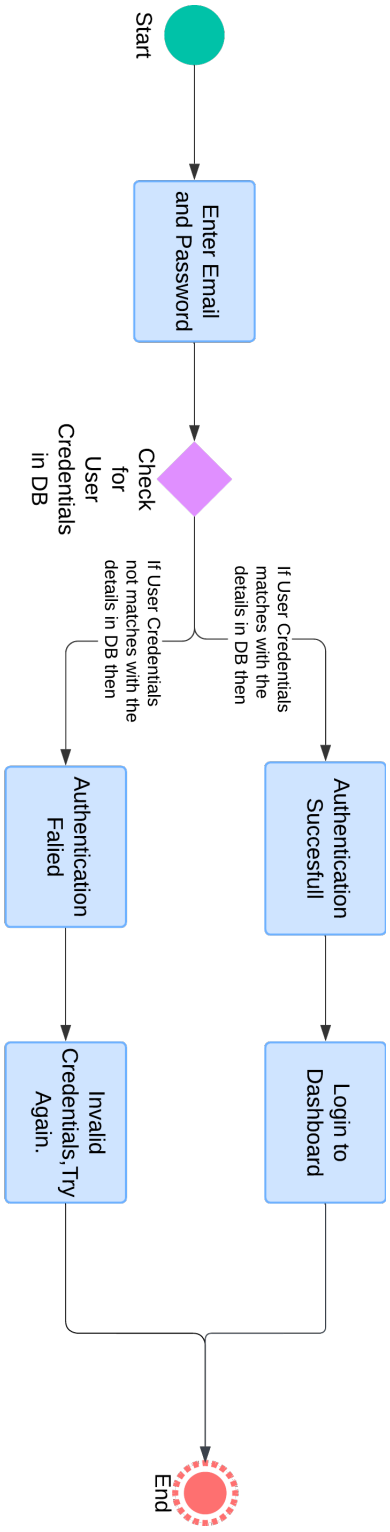


Figure 3.16: State-Chart Diagram for User Login

The algorithm for User Authentication is given in Algorithm 1.

Algorithm 1: Algorithm for User Authentication

Input: Login credentials
Result: Authentication status (Success/Failure message with token if successful) and access to dashboard

```
Initial Screen: /* Display options for login                    */
Input the Email and Password from the user.
Search for the Email in the user database.
if the user with the Email is found then
    Validate the Password.
    if the Password matches then
        Generate an Authentication Token.
        Grant access to the system.
        Display "Authentication Successful"
    end
    else
        Display "Invalid Password,Try Again!"
    end
end
else
    Display "User Not Found."
end
```

3.2.3.2 State chart diagram and algorithm of stock allocation

The State Chart Diagram for stock allocation is given in figure 3.17

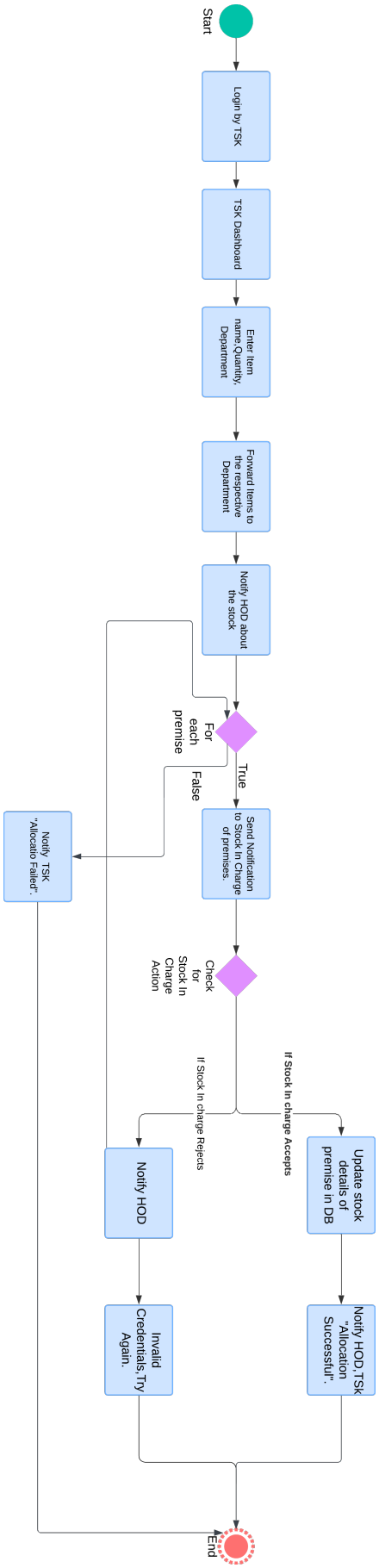


Figure 3.17: State Chart diagram for stock allocation

The algorithm for Stock Allocation is given in Algorithm 2.

Algorithm 2: Algorithm for Stock Allocation

Input: Item Name, Quantity, Department (from TSK)
Result: Allocation Successful: Notifications sent to Stock-In-Charge, HOD, and TSK
Allocation Failed: Notification sent to TSK

```
1  Initial Screen: /* Stock Allocation Page                    */
2  Input Item Name, Quantity, and Department (from TSK).
3  Forward the items to the selected department by TSK.
4  Notify the HOD about the stock allocation request.
5  if HOD forward items to the specific Premise. then
6  |  Send Notification to the Stock-In-Charge of the premise with the allocation
   |   request.
7  |  Wait for Response from the Stock-In-Charge.
8  |  if Stock-In-Charge Accepts then
9  |  |  Update the stock for the premise and update database data.
10 |  |  Notify HOD, Custodian, and TSK: "Allocation Successful".
11 |  end
12 |  if Stock-In-Charge Rejects then
13 |  |  Notify the HOD about the rejection.
14 |  |  Notify TSK: "Allocation Failed".
15 |  end
16 end
```

3.2.3.3 State Chart diagram and algorithm of Stock Transfer

The State Chart Diagram for Stock Transfer is given in figure 3.18

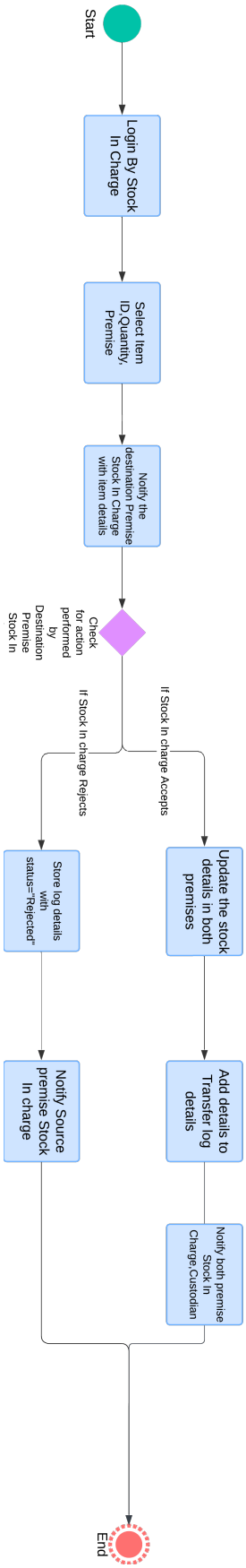


Figure 3.18: State-Chart Diagram for Stock Transfer

The algorithm for Stock Transfer is given in Algorithm 3.

Algorithm 3: Algorithm for Stock Transfer

Input: Item name,Destination premise name by Stock in charge
Result: Transfer Successful: Notification sent to Destination,Source Premise's Stock-In-Charge,Custodian.
Transfer Failed: Notification sent to Destination,Source Premise's Stock-In-Charge.

1 **Initial Screen:** `/* Stock Transfer page */`
2 Select Item ID,and Destination Premise by the Stock-In-Charge of the source premise.
3 Send Notification to the Stock-In-Charge of the selected Destination Premise with transfer details:Item name,Quantity,Source Premise name.
4 **if** *the Stock-In-Charge of the Destination Premise accepts the transfer request* **then**
5 Update the stock database.
6 Deduct the specified Stock from the Source Premise's stock.
7 Add the specified Stock to the Destination Premise's stock.
8 Store the transfer Log details : Source Premise Name, Destination Premise Name, Item ID, Date of Transfer, Status: Accepted.
9 Notify the sender,receiver Stock-In-Charge,Custodian with a successful transfer message.
10 **else**
11 **if** *the Stock-In-Charge of the Destination Premise rejects the transfer request* **then**
12 Store the transfer Log details : Source Premise Name, Destination Premise Name, Item ID, Date of Transfer, Status: Rejected.
13 Notify the sender Stock-In-Charge with an unsuccessful transfer message.
14 **end**
15 **end**

3.2.3.4 State Chart diagram and algorithm of Stock Clearance

The State Chart Diagram for Stock Clearance is given in figure 3.19

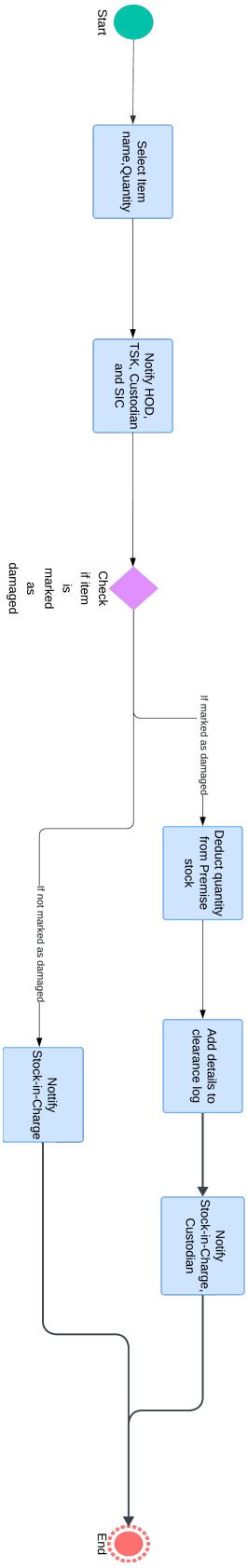


Figure 3.19: State-Chart Diagram for Stock Clearance

The algorithm for the Stock Clearance is given in Algorithm 4.

Algorithm 4: Algorithm for Stock Clearance

Input: Item name,Department by Stock in charge
Result: Clearance Successful: Updates the inventory,Notification sent to HOD, TSK, Custodian.
Transfer Failed: Notification sent to Stock-In-Charge.

1 **Initial Screen:** `/* Stock Clearance page */`
2 Select Item ID by the Stock-In-Charge.
3 Send Notification to HOD, TSK, Custodian, the Stock-In-Charge with clearance details:Item ID,name.
4 **if** *the Verifier marks the damaged items for clearance* **then**
5 Update the stock database.
6 Deduct the specified Item from the Premise's stock.
7 Store the Clearance Log details : Premise Name, Item Name, Date of Clearance.
8 Notify the Stock-In-Charge,Custodian with a successful transfer message.
9 **else**
10 Notify the Stock-In-Charge with an unsuccessful clearance message.
11 **end**

3.2.3.5 State Chart diagram and algorithm of Stock Maintenance

The State chart diagram of Stock Maintenance is given in figure 3.20.

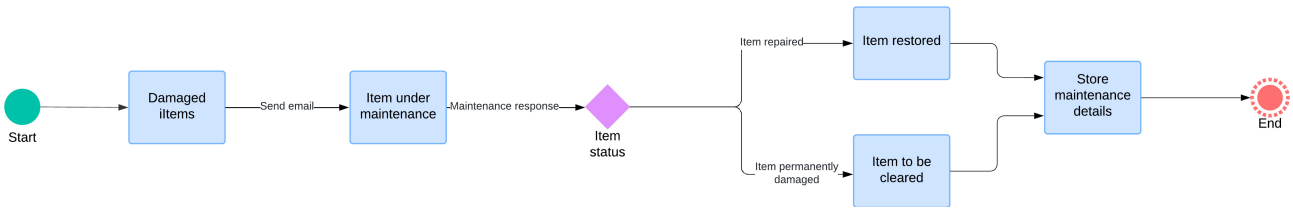


Figure 3.20: State-Chart Diagram for Stock Maintenance

The algorithm for Stock Maintenance is given in Algorithm 5.

Algorithm 5: Algorithm for Stock Maintenance

Input: Sends a Email notification to service provider by Stock in charge containing details of item requiring repair.

Result: After repair: Update item status, Notification sent to Stock-In-Charge, Custodian.

```
1  Initial Screen: /* Maintenance page                                    */
2  if User clicks on connections icon then
3  |  Display list of connections;
4  |  if User selects another user's profile icon then
5  |  |  Display the selected user's profile;
6  |  end
7  end
8  Send an email to the service provider by the Stock-In-Charge with details:Item
     name, Specification, and remark regarding the issue.
9  if the email was successfully send then
10 |  Update the item status to under maintenance.
11 |  Notify the Stock-In-Charge, Custodian with a successful email message.
12 else
13 |  Notify the Stock-In-Charge with an unsuccessful email message.
14 end
15 Wait for confirmation of task completion from the service provider.
16 if the item is repaired then
17 |  Update the item status as available.
18 |  Notify the Stock-In-Charge, Custodian with a successful repair message.
19 else
20 |  if the item is unrepairable then
21 |  |  Update the item status as unserviceable.
22 |  |  Notify the Stock-In-Charge, Custodian with the unrepairable message.
23 |  end
24 end
```

3.2.3.6 State Chart diagram and algorithm of Stock Handover

The State chart diagram of complaining and feedback system is given in figure 3.21

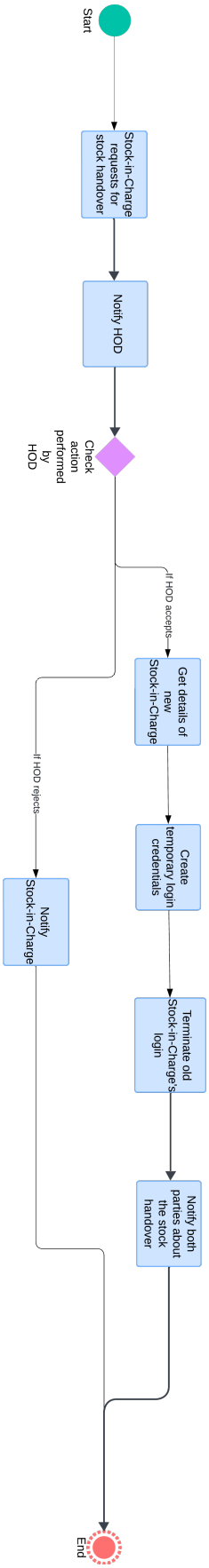


Figure 3.21: State-Chart Diagram for Stock Handover

The algorithm for Stock Handover is given In Algorithm 6.

Algorithm 6: Algorithm for Stock Handover

Input: Handover request details (from current stock-in-charge), new stock-in-charge information (appointed by HOD).
Result: Stock responsibility transferred, notifications sent, and login credentials updated.

```
1  Initial Screen: /* Stock Handover page                              */
2  if the Stock-In-Charge requests stock handover then
3  |  Notify the HOD about request.
4  end
5  if the HOD accepts the handover request then
6  |  Input details of new stock in charge.
7  |  Creates temporary login credentials.
8  |  Terminates the login credential of former stock in charge.
9  |  Notify both parties (current and new stock-in-charge) about the stock handover.
10 else
11 |  if the HOD rejects the handover request then
12 |  |  Notify the requesting stock in charge that the request has been rejected.
13 |  end
14 end
```

3.2.3.7 State Chart diagram and algorithm of assign faculty for verification

The State chart diagram of assign faculty for verification is given in figure 3.22

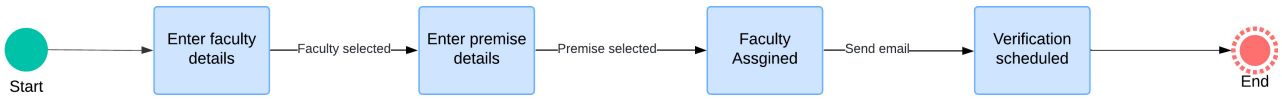


Figure 3.22: State-Chart Diagram for assign faculty for verification

The algorithm for assign faculty for verification is given In Algorithm 7.

Algorithm 7: Algorithm for Assign Faculty for Verification

Input: Name and email-ID of the faculty , department and premise to be verified.
Result: Notification send to HOD of the specified department.

```
1 Initial Screen: /* Assign Faculty page                               */
2 Enter the name of the faculty
3 Enter email id of the faculty
4 Select the department where verification is to be performed
5 Select the premise within the department
6 Enter the deadline date
7 Assign faculty
```

3.2.3.8 State Chart diagram and algorithm of Create temporary login for verifying faculty

The State chart diagram of create temporary login for verifying faculty is given in figure 3.23

Figure 3.23: State-Chart Diagram for Create temporary login for verifying faculty

The algorithm for Create temporary login for verifying faculty is given In Algorithm 8.

Algorithm 8: Algorithm for Create temporary login for verifying faculty

Input: Notification from principle regarding assigned faculty.
Result: Temporary login created for faculty and credentials is sent via mail.

1 **Initial Screen:** `/* Register page */`
2 Input email ID of the faculty
3 Enter a password
4 Give access to specified inventory
5 Share login credentials via email

3.2.3.9 State Chart diagram and algorithm of Stock Verification

The State chart diagram of Stock Verification is given in figure 3.24

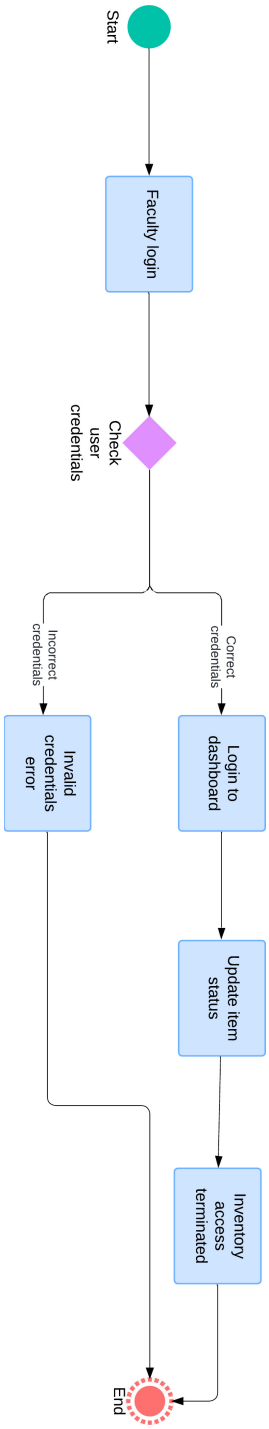


Figure 3.24: State-Chart Diagram for Stock Verification

The algorithm for Stock Verification is given In Algorithm 9.

Algorithm 9: Algorithm for Stock Verification

Input: Login credentials received in mail
Result: Verification report sent to principal
Initial Screen: `/* Verification page */`
Enter email ID and password
Fetch email ID and password from database
if *email ID and password are correct* **then**
    for *each item in inventory* **do**
        Mark item if it is present
        Mark status of item as working or not working
        Enter remarks if any
    end
    Submit verification
    Send report to principal
    Send notification to principal
else
    Display error message: "Invalid Credentials"
end

3.2.3.10 State Chart diagram and algorithm of Approval of stock verification report

The State chart diagram of Approve stock verification report is given in figure 3.25

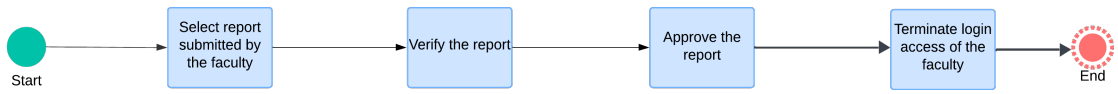


Figure 3.25: State-Chart Diagram for Approve stock verification report

The algorithm for Approve stock verification report is given In Algorithm 10.

Algorithm 10: Algorithm for Approve stock verification report

Input: Verification report received by principal
Result: Approved report
Initial Screen: `/* Reports page */`
Select the report submitted by the faculty
Verify the report
Approve report
Terminate access to inventory for the faculty

3.2.3.11 State Chart diagram and algorithm of Add a new stock system

The State chart diagram of Add a new stock system is given in figure 3.26

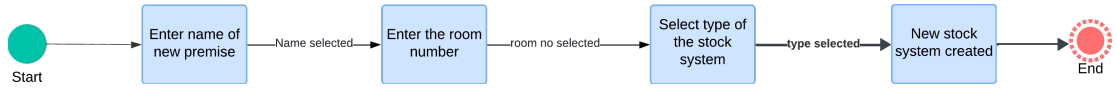


Figure 3.26: State-Chart Diagram for Add a new stock system

The algorithm for Add a new stock system is given In Algorithm 11.

Algorithm 11: Algorithm for Add a new stock system

Input: Name,room no and type of new stock system
Result: A new database is created
1 **Initial Screen:** `/* Add stock system page */`
2 Enter name of the premise
3 Enter the room no
4 Enter the type of the stock system
5 Create the stock system

3.2.3.12 State Chart diagram and algorithm of Remove a faculty

The State chart diagram of Remove a faculty is given in figure 3.27

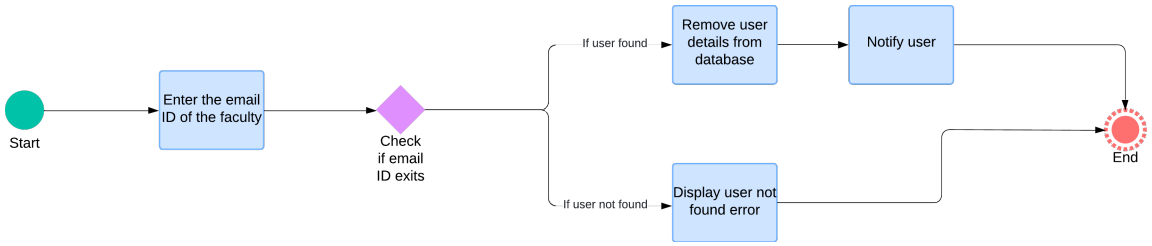


Figure 3.27: State-Chart Diagram for Remove a faculty

The algorithm for Remove a faculty is given In Algorithm 12.

Algorithm 12: Algorithm for Remove a faculty

Input: email-ID of the faculty to be removed
Result: The details of the faculty is removed from the database
1 **Initial Screen:** `/* Remove Account Page */`
2 Enter the email-ID of the faculty
3 **if** *User with email-ID is found* **then**
4 | Remove the details from the database
5 **else**
6 | Display error message: "User not found !"
7 **end**

3.2.3.13 State Chart diagram and algorithm of Add an item to the inventory

The State chart diagram of Add an item to the inventory is given in figure 3.28

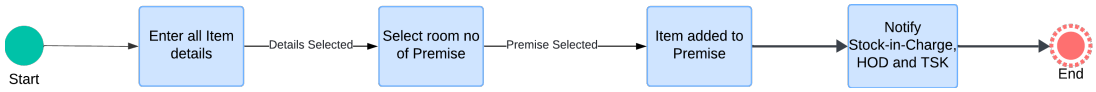


Figure 3.28: State-Chart Diagram for Add an item to the inventory

The algorithm for Add an item to the inventory is given In Algorithm 13.

Algorithm 13: Algorithm for Add an item to the inventory

Input: Name,ID,date of purchase,date of intend,description,warranty period and room no of the item
Result: The item gets allocated to the premise and notifications are send to stock-in-charge,HOD and TSK

1 **Initial Screen:** `/* Add Stock page */`
2 Enter the item name
3 Enter the item ID
4 Enter the date of purchase
5 Enter the date of intend
6 Enter the item description
7 Enter the warranty period
8 Enter the room no of the premise
9 Add the item to the premise
10 Send the notifications to stock-in-charge,HOD and TSK

3.2.3.14 State Chart diagram and algorithm of Assign faculty in-charge for an inventory

The State chart diagram of Assign faculty in-charge for an inventory is given in figure 3.29

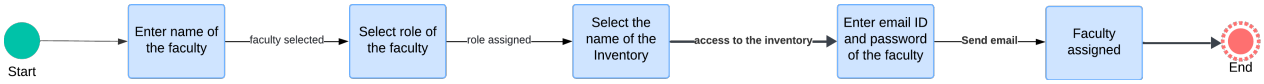


Figure 3.29: State-Chart Diagram for Assign faculty in-charge for an inventory

The algorithm for Assign faculty in-charge for an inventory is given In Algorithm 14.

Algorithm 14: Algorithm for Assign faculty in-charge for an inventory

Input: Name,role,email-ID and password of the faculty and name of the inventory
Result: Faculty in-charge is assigned for the inventory

1 **Initial Screen:** `/* Register page */`
2 Enter the name of the faculty
3 Enter the role of the faculty
4 Enter the email-ID
5 Enter the password
6 Enter the name of the inventory
7 Assign faculty in-charge for the inventory

3.2.3.15 State Chart diagram and algorithm of Filter And Report Generation

The State chart diagram of Filter And Report Generation for an inventory is given in figure 3.30

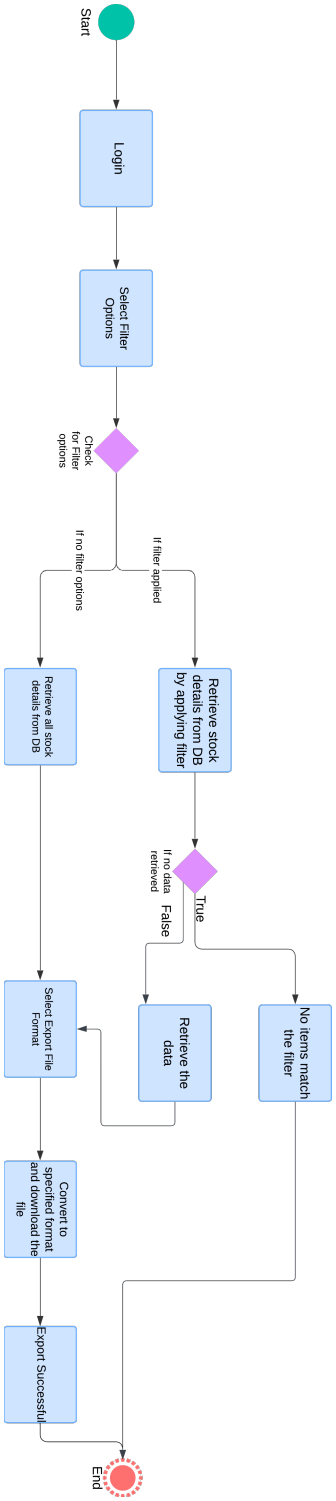


Figure 3.30: State-Chart Diagram for Filter And Report Generation

The algorithm for Filter And Report Generation is given In Algorithm 15.

Algorithm 15: Algorithm for Filter And Report Generation for an inventory

Input: Filter options: Item Type,condition,other specifications
Export options: PDF,Excel
Result: Filtered Inventory Report or Full Stock Report in the chosen format (PDF or Excel).

```
1  Initial Screen: /* Stock Details page                                    */
2  Input filter options (Item Type, Condition,etc.) and export format (PDF or Excel).
     check filters.
3  if no Filters are applied then
4  |  Query the database for all stock details
5  else
6  |  Query the database for records matching the selected filter options.
7  end
8  Check the Query result if no records are found then
9  |  Display "No items match the selected filters." Exit
10 else
11 |  Retrieve the matching records or full stock details.
12 end
13 Choose the Export format by user
14 if Export format chosen by user = PDF then
15 |  Format the data (filtered or full stock) into a PDF report using a PDF
   |    generation library.
16 |  Save the PDF file.
17 |  Display "PDF Report Generated Successfully."
18 else
19 |  if Export format chosen by user=Excel then
20 |  |  Format the data (filtered or full stock) into a Excel report using a Excel
   |  |    generation library.
21 |  |  Save the Excel file.
22 |  |  Display "Excel Report Generated Successfully."
23 |  end
24 end
25 if User clicks download option then
26 |  Download the generated report.
27 end
```

3.3 Summary

The proposed system for Department Stock Management System consists of different modules for the UI, database, and connectivity, making sure they work together smoothly. In the detailed design phase, the organization of each functionality was described using state chart diagrams and algorithms.The next chapter, *Implementation*, focuses on developing and integrating these components.

Chapter 4

Implementation

The development or implementation stage [4, 5] is the stage where the development of the code and modules are performed. The development and integration of the units and creation of the project build are done according to the design documents' specifications. The goal of this phase is to translate the design of the proposed system into a working model. Since this phase affects the testing phase, the coding for the project is performed in such a way as to maintain simplicity and clarity.

4.1 Tools Used

This project is associated with web application development. The following are the tools used to develop Department Stock Management System web application.

1. **Figma** [6, 7]

   Figma is a collaborative web application tailored for interface design. The feature set of Figma focuses on user interface and user experience design, with an emphasis on real-time collaboration, utilizing a variety of vector graphics editors and prototyping tools.

2. **Lucidchart** [8]

   Lucidchart is a web-based diagramming tool used for creating flowcharts, UML diagrams, and system models. It supports real-time collaboration, making it useful for visualizing software architectures and workflows.

3. **MongoDB** [9]

   MongoDB is a source-available, cross-platform, document-oriented database program. Classified as a NoSQL database product, MongoDB utilizes JSON-like documents with optional schemas.

4. **Node.js** [10, 11]

   Node.js is an open-source and cross-platform JavaScript runtime environment. Node.js app runs in a single process, without creating a new thread for every request. Node.js provides a set of asynchronous I/O primitives in its standard library that prevent JavaScript code from blocking and generally, libraries in Node.js are written using non-blocking paradigms, making blocking behavior the exception rather than the norm

5. **Express.js** [11]

   Express.js is a back-end web application framework for building RESTful APIs with Node.js. It is a free and open-source software. It is designed for building web applications and APIs. It is a widely used server framework for Node.js.

   Express is the MERN stack's back-end component, with the MongoDB database software and a JavaScript front-end framework or library.

6. **React.js** [12]

   React.js is an open-source JavaScript library for building user interfaces, primarily for web applications. It allows developers to create reusable UI components and efficiently update and render user interfaces based on state changes. React.js uses a virtual DOM to optimize rendering performance, resulting in a fast and responsive user experience. It enables component-based development, making applications easier to maintain and scale.

7. **Vite** [13]

   Vite is a modern frontend build tool and development server optimized for fast performance. It provides instant server startup, lightning-fast hot module replacement (HMR), and optimized production builds. Vite enhances the development experience for React.js applications by ensuring rapid updates and efficient code bundling.

8. **Render** [14]

   Render is a cloud-based hosting platform used for deploying backend applications. It supports Node.js, automatic scaling, and database integration, making it ideal for hosting the backend of the Department Stock Management System. It provides a secure and efficient environment for running server-side operations.

9. **Vercel** [15]

   Vercel is a frontend deployment platform optimized for modern web applications. It offers fast content delivery, automatic builds, and seamless integration with Git repositories. Vercel ensures that the frontend of the Department Stock Management System is deployed efficiently, providing a smooth user experience.

10. **Visual Studio Code** [16]

    Visual Studio Code offers a wide range of features to enhance the coding experience, such as syntax highlighting, intelligent code completion, and debugging capabilities. The code editing capabilities, extensions, and integration with Git contribute to increased productivity and collaboration throughout the development.

11. **GitHub** [17]

    GitHub is a web-based platform for version control using Git. It provides hosting for software development projects, enabling developers to collaborate on code, track changes, and manage project workflows. With GitHub, developers can contribute to projects by making pull requests, reviewing code changes, managing issues, and collaborating with others

4.2 Packages Used

This section details the external libraries and dependencies utilized in developing the Department Stock Management System. These packages provide essential functionalities for frontend development, backend operations, authentication, data processing, and UI enhancements.

4.2.1 Frontend Packages

4.2.1.1 React.js

React.js is a JavaScript library for building dynamic user interfaces. It enables component-based development, improving modularity and maintainability. The following core React functions are used:

- `React`: The core library for creating UI components.
- `useState`: A hook for managing state in functional components.
- `useEffect`: A hook for handling side effects in components.

4.2.1.2 Vite

Vite is a frontend build tool designed for fast development and optimized production builds, offering quick server startup and hot module replacement (HMR).

4.2.1.3 Material-UI (MUI)

Material-UI is a component library that provides pre-built UI elements for creating a visually consistent and responsive interface.

4.2.1.4 React Router

React Router enables navigation between different pages in the application without requiring full page reloads.

- `BrowserRouter`: Manages client-side routing.
- `Routes`: Defines paths for different views.
- `Link`: Enables seamless navigation between components.

4.2.1.5 JWT-Decode

JWT-Decode is used to parse JSON Web Tokens (JWTs) and extract user information on the frontend.

4.2.1.6 jsPDF and jsPDF-AutoTable

jsPDF allows generating PDF reports from web applications, while the `jspdf-autotable` plugin enables structured table exports for inventory records.

4.2.1.7 XLSX

XLSX is a JavaScript library for reading and writing Excel files, supporting data export for inventory records.

4.2.1.8 Axios

Axios is a promise-based HTTP client used to handle API requests between the frontend and backend.

4.2.2 Backend Packages

4.2.2.1 Node.js

Node.js provides a runtime environment for executing JavaScript on the server side.

4.2.2.2 Express.js

Express.js is a lightweight web framework that simplifies routing, middleware management, and API creation.

4.2.2.3 Mongoose

Mongoose is an Object Data Modeling (ODM) library for MongoDB, offering schema-based validation and structured database interactions.

4.2.2.4 jsonwebtoken

jsonwebtoken is a library for generating and verifying JSON Web Tokens (JWTs), enabling secure authentication.

4.2.2.5 bcryptjs

bcryptjs is a package used for hashing passwords before storing them in the database, enhancing security.

4.2.2.6 Nodemailer

Nodemailer facilitates email communication by sending notifications for stock verification and approvals.

4.2.2.7 Validator

Validator provides built-in functions to validate user input, ensuring correct data formats during registration and stock transactions.

4.2.2.8 Dotenv

Dotenv is used for managing environment variables, keeping sensitive data like API keys and database credentials secure.

4.2.2.9 Cors

Cors is a middleware that enables secure communication between the frontend and backend by handling Cross-Origin Resource Sharing (CORS) policies.

4.2.2.10 Morgan

Morgan is a logging middleware for monitoring HTTP requests in Express.js applications, useful for debugging API calls.

4.3 Module Implementation

In this section, we describe how each module is implemented in detail.

4.3.1 Front-end Module

We developed our frontend module using React.js and JavaScript. More specifically, we used Vite as our build tool to streamline the development process, ensuring fast server startup, hot module replacement (HMR), and optimized production builds.

We utilized multiple packages such as react, react-router-dom, @mui/material, react-icons, jwt-decode, and axios to enhance functionality and improve the user experience by providing efficient routing, UI components, authentication handling, and API communication.

4.3.2 Database Module

Our web application's MongoDB database is divided into the following collections. The content of each collection is described below:

1. *User Collection*: Stores user details, authentication credentials, and role-based access information. Each user is identified by a unique `email_id` and has associated fields such as `name`, `password`, and `designation` (e.g., HOD, Faculty, Stock-In-Charge). The collection also includes `otp` and `otpExpires` fields for handling authentication via one-time passwords when required. This collection plays a crucial role in managing user authentication, authorization, and system access.

2. *Room Collection*: Stores information about rooms where inventory items are located. Each room is identified by its `room_no` and includes details such as `name` (e.g., DBMS Lab), `custodian` (responsible for room maintenance), and `in_charge` (faculty overseeing stock management). The `type` field specifies the room category (e.g., Lab, Office, Storage). This collection helps in organizing and managing stock distribution across different departmental locations.

3. *Stock Collection*: Stores details of inventory items within the department. Each item is identified by its `indent_no` and includes fields such as `name` (e.g., LAPTOP), `sl_no` (serial number), and `qty` (quantity available). Additional details include `date_of_purchase`, `specification` (e.g., HP I3 12th Gen), `warranty_period`, and `price`. This collection helps in tracking inventory, managing stock movement, and maintaining purchase records.

4. *Item Collection*: Stores information about individual inventory items within the department. Each item is identified by its `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 15) and includes details such as `status` (e.g., Not Working) and `type` (e.g., Electronics). This collection helps in tracking the condition and classification of stock items.

5. *Includes Collection*: Maintains the relationship between stock items and their corresponding records. Each entry is identified by its `_id` and includes fields such as `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 10), `indent_no` (purchase reference), and `sl_no` (serial number). This collection ensures accurate mapping of inventory items to their procurement details.

6. *BelongsTo Collection*: Maintains the mapping between inventory items and their assigned locations. Each entry includes `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 17) and `room_no` (e.g., 101), ensuring that every stock item is properly associated with its designated room. This collection helps in tracking the physical placement of inventory within the department.

7. *Access Collection*: Manages user permissions for accessing specific rooms and inventory. Each entry includes `email_id` and `room_no` (e.g., 101), determining which users have access to manage stock in a given room. This collection ensures controlled and authorized inventory handling.

8. *AssignedFacultyList Collection*: Stores records of faculty members assigned to oversee stock management within specific premises. Each entry includes `facultyName` (e.g., Arjun), `facultyemail`, `department` (e.g., CSE), and `premise` (e.g., OS LAB). It also tracks assignment details such as `assigned_date`, `last_date`, `verified_date`, and `status` (e.g., Completed). This collection ensures accountability and proper faculty supervision of inventory verification.

9. *Clearance Collection*: Stores records of inventory items marked for clearance. Each entry includes `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 21), `room_no` (e.g., 101), `type` (e.g., Electronics), and `status` (e.g., Cleared). Additional details such as `remarks` (e.g., need to dispose) and `clearance_date` ensure proper tracking of disposed or obsolete stock items. This collection helps maintain an updated inventory by removing unusable items.

10. *CSEMain Collection*: Stores inventory records specific to the Computer Science department. Each entry includes `sl_no` (serial number), `indent_no` (purchase reference), `date_of_purchase`, `price` (e.g., 25000), `quantity` (total purchased stock), and `remaining` (available stock). This collection helps in tracking stock usage, procurement details, and ensuring proper inventory management within the department.

11. *Main Collection*: Stores general inventory records across all departments. Each entry includes `sl_no` (serial number), `indent_no` (purchase reference), `date_of_purchase`, `price` (e.g., 1500), `quantity` (total purchased stock), and `department` (e.g., CSE). This collection helps in managing stock procurement and tracking inventory distribution across multiple departments.

12. *Maintenance Collection*: Stores records of maintenance requests for inventory items. Each entry includes `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 15), `complaint_date` (date the issue was reported), `service_provider` (responsible for maintenance), and `status` (e.g., Pending). This collection ensures proper tracking of maintenance activities and helps manage equipment repairs efficiently.

13. *MaintenanceHistory Collection*: Stores records of completed maintenance activities for inventory items. Each entry includes `item_no` (e.g., RIT/CSE/DBMS LAB/DESK 6), `status` (e.g., Completed), `completed_date` (date of maintenance completion), `remarks` (e.g., ok), `item_status` (e.g., Working), and `amount` (maintenance cost, e.g., 1500). This collection helps in tracking maintenance history and ensuring proper record-keeping of repaired items.

14. *Notification Collection*: Stores system-generated notifications related to stock transactions and updates. Each entry includes `type` (e.g., tskstockforward), `sender` (e.g., tsk@rit.ac.in), `receiver`, `indent_no` (e.g., IC), `sl_no` (serial number), `status` (e.g., read), and `date` (timestamp of the notification). This collection ensures that users receive timely updates regarding stock movements and approvals.

15. *VerificationList Collection*: Stores records of stock verification conducted by authorized personnel. Each entry includes `verifier_name` (e.g., Adwaith), `verifier_email`, `item_no` (e.g., RIT/CSE/DBMS LAB/LAPTOP 1), `date_of_verify` (timestamp of verification), `status` (e.g., Working), and `remarks` (e.g., ok). This collection ensures accountability and proper tracking of inventory verification processes.

16. *Counters Collection*: Stores aggregated inventory data for each lab or department. Each entry includes `labname` (e.g., DBMS LAB), `itemname` (e.g., DESK), and `value` (total count of the item, e.g., 25). This collection helps in maintaining quick access to stock summaries and tracking inventory levels efficiently.

4.3.3 Connectivity Module

The Connectivity module is a critical component of any web application, designed to facilitate seamless communication and data exchange between the Front-end module and Database module of the system. This module ensures that users can interact with the website's features efficiently. It leverages various technologies and frameworks to handle HTTP requests, manage user sessions, and maintain data consistency across the application.

4.3.3.1 Implementation of features in detail

A detailed description of the implementation of each feature in Department Stock Management System is included in this section.

1. **User Authentication**

   The **User Authentication** feature ensures secure access to the system based on role-based authentication. It includes user login, password reset via OTP, and session management. The authentication process is implemented using React for the frontend and Node.js with Express.js for backend API communication.

Login Process:
The login page is built using React with state management via *useState* hooks. The form consists of input fields for email and password, managed within a functional component. The *handleSubmit* function is triggered upon clicking the login button. It validates inputs, then makes an HTTP POST request to the backend authentication endpoint using *axios.post*. The backend verifies credentials and, upon successful authentication, returns a JWT token. The frontend stores this token in *sessionStorage.setItem* and navigates users to their respective dashboards using *navigate*.

Forgot Password and OTP Verification:
Users can reset their passwords if forgotten. They enter their email, and an OTP is sent via an API request using *axios.post*. Upon receiving the OTP, the user enters it into the verification form, which is then validated via another API request. If the OTP is correct, the user can reset their password. The new password is updated in the database via an API call using *axios.post*.

User Registration:
New users are registered through a form that collects name, email, role, password, and assigned inventory. Input validation is performed before making a request to the backend using *axios.post*. The backend hashes the password before storing it in the database and assigns a role-based dashboard upon successful registration. If the registration is successful, a confirmation message is displayed, and the user is redirected to the login page using *navigate*.

Key Methods Used:

- *axios.post* – To send authentication requests to the backend.
- *sessionStorage.setItem* – To store the authentication token securely.
- *navigate* – To redirect users based on their role.
- *useState* – For managing user input within React components.

2. **Stock Allocation**

The **Stock Allocation** functionality ensures the allocation of inventory items (e.g., computers, furniture) to specific classrooms and labs within the department based on requirements.

Stock Forwarding from TSK to HOD:
The stock forwarding process involves filling out a form containing details such as Serial Number, Indent Number, Date of Purchase, Price, and Quantity.

The form is built using React, with input fields managed using *useState*. The form submission is handled via an API request using *axios.post*. The request is authenticated using a JWT token retrieved from *sessionStorage.getItem*.

The system validates the input fields to ensure Serial Number, Price, and Quantity are valid numbers. If the validation fails, an error message is displayed.

Upon successful submission, the stock details are sent to the backend for processing. If the request is successful, an alert message is shown, and the user is redirected to the *Tskdash* dashboard using *navigate*.

Stock Forwarding from HOD to SIC:
Once the stock is forwarded to the Head of Department (HOD), it is further allocated to specific premises such as classrooms or labs.

When the form loads, it fetches available stock data using *axios.get*. The stock details, including Serial Number, Indent Number, Item Name, Quantity, Price, and Date of Purchase, are retrieved and prefilled in the form.

Additionally, the system fetches room names from an inventory database to allow the HOD to assign the stock to a specific premise.

Upon submission, the selected stock item is forwarded to the specified premise using *axios.post*. The request includes sender information, ensuring traceability. If the request is successful, the user is redirected to the *Hoddash* dashboard.

Key Methods Used:

- *useState* – To manage input field state in React.
- *axios.post* – To send stock forwarding requests to the backend.
- *axios.get* – To fetch available stock and premises from the database.
- *sessionStorage.getItem* – To retrieve authentication tokens.
- *navigate* – To redirect users to their respective dashboards after successful stock forwarding.

3. **Stock Verification and Remarks**
This feature allows the annual stock verification process and submission of the annual report in online mode. It consists of assigning faculty for verification by the principal, allowing faculty to view and update the status of inventory items, and enabling the principal to review and approve the verification reports. Temporary login credentials are automatically deleted upon completion and approval.

Assigning Faculty for Verification (Principal):

- The Principal uses the `AssignFaculty` component to assign faculty members to verify specific premises.
- The form uses `useState` to manage input fields for faculty name, email, department, premise, and last date.
- The form fetches the list of available premises from the backend using `axios.get`.
- On form submission, an HTTP POST request is sent to the backend API using `axios.post` with the form data and a JWT token for authentication.
- The backend stores the assignment details and sends a notification to the assigned faculty.
- Upon successful assignment, a success message is displayed, and the form is reset.

Stock Verification by Faculty:

- The Faculty uses the `Stockverifications` component to view and update the status of inventory items.
- The component fetches the list of stock details from the backend using `fetch` with an authenticated header.

- The faculty can update the status of each item (Working, Not Working, Not Repairable) using a dropdown menu.
- Remarks can be added for each item using an input field.
- On form submission, the updated status and remarks are sent to the backend using `fetch` with HTTP PUT and POST requests.
- The backend updates the stock status and creates a verification record.
- A notification is sent to the principal after the verification is submitted.
- After successful submission, the temporary user credentials are deleted, and the user is redirected.

Verification Report Review and Approval (Principal):

- The Principal uses the `Reportdetails` component to view the verification reports submitted by faculty.
- The component fetches the report details from the backend using `axios.post` with the notification ID and a JWT token for authentication.
- The report details, including verifier email, verification date, and item details, are displayed in a table.
- The principal can search for items using an input field and filter the report.
- The report can be exported as a PDF or Excel file using `jsPDF` and `xlsx` libraries.
- Upon approval, the principal can log remarks for future reference.
- The system automatically deletes the temporary user credentials on completion of the stock verification and the approval of the principal.

Key Methods Used:

- `useState` - For managing component state, including form data and stock details.
- `useEffect` - For fetching data from the backend and decoding JWT tokens.
- `axios.get` and `axios.post` - For making HTTP requests to the backend API.
- `fetch` - For making HTTP requests to the backend API.
- `jwtDecode` - For decoding JWT tokens to get user information.
- `jsPDF` - For exporting reports as PDF files.
- `xlsx` - For exporting reports as Excel files.
- `useSearchParams` and `useParams` - For handling URL parameters.

4. **User-Based Notification**
   This feature enables real-time notifications to users based on their roles and actions within the system. It handles various notification types, including stock forwarding, verification reports, and messages between users.

   Notification Retrieval:

   - The frontend retrieves notifications by making API requests to the backend.
   - Two main API endpoints are used:

  - `/api/notifications`: This endpoint fetches general, unread notifications. It requires the `receiver` email as a query parameter. The backend filters notifications in the `GeneralMiniNotification` model where the `receiver` matches and the `status` is "unread".
  - `/api/fetch-notifications`: This endpoint fetches detailed, unread notifications related to specific actions (e.g., stock forwarding). It also requires the `receiver` email as a query parameter. This endpoint queries various notification models (e.g., `TskNotification`, `HodAcceptNotification`) and processes the results.
- The backend processes the retrieved notifications to format the messages and include relevant data (e.g., indent number, quantity).

Notification Types and Processing:
The backend handles different notification types, each with its own processing logic:

- **General Notifications:**
  - A `switch` statement is used to generate user-friendly messages based on the `type` of notification (e.g., "tskstockforward" becomes "New message from TSK").
  - Notifications are fetched from the `GeneralMiniNotification` model.
- **Action-Specific Notifications:**
  - Notifications are fetched from various models, including `TskNotification`, `HodAcceptNotification`, `HODForwardNotification`, `SicStockAccept`, `SicRejectNotification`, `AssignfacultyNotification`, `VerifyNotification`, `HandoverStockNotification`, and `StockTransferNotification`.
  - `Promise.all` is used to process notifications that require fetching related data (e.g., stock details from the `MainStock` model).
  - The notifications are mapped to a standardized format, including fields like `_id`, `type`, `message`, `status`, and `createdAt`.

Key Methods Used:

- `express` - For creating the backend API routes.
- `mongoose` - For interacting with the MongoDB database.
- `router.get` - For defining GET API endpoints.
- `req.query` - For accessing query parameters in the request.
- `res.json` - For sending JSON responses to the client.
- `res.status` - For sending HTTP status codes.
- `Promise.all` - For handling asynchronous operations concurrently.
- `switch` statement - For conditional message formatting.

5. **Report Generation**

   This feature enables users to generate filtered and comprehensive reports about inventory items based on specific criteria. It supports on-demand report generation and export in PDF and Excel formats.

   Report Generation and Filtering:

- The `Reportdetails` component allows users to view and generate reports.
- The component fetches report details from the backend using `axios.post` with a notification ID and a JWT token for authentication.
- The fetched data includes verifier email, verification date, and an array of report items.
- Users can filter the report items using a search term entered in an input field.
- The report items are displayed in a table with columns for Item No, Status, Remarks, and Date of Verify.
- The table data is filtered based on the search term using the `filter` method and `includes` string method.

Report Export:

- The component provides options to export the generated report in PDF and Excel formats.
- **Export to PDF:**
  - The `exportToPDF` function uses the `jsPDF` library to generate a PDF document.
  - The function creates a table with columns for Item No, Status, Remarks, and Date of Verify.
  - The table data is populated using the report items.
  - The PDF document is saved with the name "Verification_Report.pdf".
- **Export to Excel:**
  - The `exportToExcel` function uses the `xlsx` library to generate an Excel file.
  - The function creates a worksheet with columns for Item No, Status, Remarks, and Date of Verify.
  - The worksheet data is populated using the report items.
  - The Excel file is saved with the name "Verification_Report.xlsx".

Key Methods Used:

- `React` - For building the user interface.
- `axios` - For making HTTP requests to the backend API.
- `jsPDF` - For generating PDF documents.
- `xlsx` - For generating Excel files.
- `useState` and `useEffect` - For managing component state and fetching data.
- `filter` and `includes` - For filtering report items based on search criteria.
- `map` - For transforming report items into table rows and worksheet data.
- `useSearchParams` - For handling URL parameters.

6. **Stock Transfer**

   This feature enables the stock-in-charge to transfer items from one premise to another. It includes selecting destination premises, handling transfer requests, and updating stock records.

   Initiating Stock Transfer:

- The `Stocktransfer` component allows the stock-in-charge to initiate stock transfers.
- The component fetches stock details using `fetch` and room inventory using `axios.get`.
- The stock details are displayed in a table, with each row allowing the user to select a destination room.
- The `handleRoomChange` function updates the `room_no` and `room_name` in the stock state when a new room is selected.
- The `handleTransfer` function sends a batch transfer request to the backend API using `fetch`.
- The backend creates a notification for the destination room's stock-in-charge.

Backend Processing (Transfer Creation):

- The `/api/ststock/transfer` endpoint handles the transfer request.
- It validates the request body, ensuring `item_no` and `room_no` are provided.
- It finds the destination room using the `Room` model.
- It finds the stock-in-charge user for the destination room using the `Access` and `User` models.
- It creates a new notification using the `TransferStockNotification` model for each item being transferred.
- It returns a success message with the processed items.

Handling Transfer Requests (Accept/Reject):

- The `/api/accept-stock-transfer` endpoint handles accepting transfer requests.
- It verifies the user's authorization using a JWT token.
- It updates the `room_no` in the `BelongsTo` model for the transferred item.
- It updates the notification status to "accepted" in the `TransferStockNotification` model.
- The `/api/reject-stock-transfer` endpoint handles rejecting transfer requests.
- It verifies the user's authorization using a JWT token.
- It updates the notification status to "rejected" in the `TransferStockNotification` model.

Key Methods Used:

- `React` - For building the user interface.
- `axios` and `fetch` - For making HTTP requests to the backend API.
- `jsonwebtoken` - For verifying JWT tokens.
- `mongoose` - For interacting with the MongoDB database.
- `useState` and `useEffect` - For managing component state and fetching data.
- `map` and `filter` - For transforming and filtering stock data.

- `express` - For creating the backend API routes.

7. **Stock Clearance**

   This feature allows custodians to remove damaged or unwanted items from their premises. It includes selecting items for clearance, updating stock records, and sending notifications.

   Selecting Items for Clearance:

   - The `Stockclears` component allows custodians to select items for clearance.
   - The component fetches stock clearance details from the backend using `fetch`.
   - The fetched data includes item IDs, remarks, status, and clearance dates.
   - Users can select items for clearance using checkboxes.
   - The `handleCheckboxChange` function updates the `checkedStocks` state when a checkbox is changed.

   Clearing Selected Items:

   - The `handleClearStocks` function sends a clearance request to the backend API using `fetch`.
   - The request includes an array of selected item IDs.
   - The backend updates the stock records and sends notifications to relevant parties.
   - Upon successful clearance, the component updates the UI to reflect the cleared items.
   - The `setcStocks` function updates the component's state to reflect the status changes.
   - The `setCheckedStocks` function resets the checked items.

   Backend Processing (Stock Clearance):

   - The `/api/stockclearance` endpoint fetches stock clearance details.
   - It verifies the user's authorization using a JWT token.
   - It fetches clearance records based on the user's room access and item type.
   - It constructs stock information, including item IDs, remarks, status, and clearance dates.
   - The `/api/clear-stock` endpoint handles clearing selected items.
   - It updates the `Clearance` model to mark the selected items as cleared.
   - It removes the cleared items from the `BelongsTo`, `Includes`, and `Item` models.

   Key Methods Used:

   - `React` - For building the user interface.
   - `fetch` - For making HTTP requests to the backend API.
   - `jsonwebtoken` - For verifying JWT tokens.
   - `mongoose` - For interacting with the MongoDB database.

- `useState` and `useEffect` - For managing component state and fetching data.
- `map` and `filter` - For transforming and filtering stock data.
- `express` - For creating the backend API routes.
- `updateMany` and `deleteMany` - For bulk updates and deletions in the database.
- `Promise.all` - For parallel execution of database operations.

8. **Stock Maintenance**

   This feature allows users to record and manage maintenance details for inventory items. It includes logging maintenance requests, updating maintenance status, and tracking maintenance history.

   Logging Maintenance Requests:

   - The `MaintenanceList` component displays a list of maintenance requests.
   - The component fetches maintenance data from the backend using `fetch`.
   - The fetched data includes item IDs, repair dates, service providers, amounts, remarks, item statuses, and maintenance statuses.
   - Users can update the amount and remarks using `TextField` components.
   - The `handleFieldUpdate` function sends an update request to the backend API using `fetch`.
   - Users can update the item status and maintenance status using dropdown menus.
   - The `handleStatusUpdate` function sends a status update request to the backend API using `fetch`.

   Backend Processing (Maintenance Requests):

   - The `/api/maintenance/list` endpoint fetches maintenance data.
   - It verifies the user's authorization using a JWT token.
   - It fetches maintenance records based on the user's room access and item type.
   - It constructs a transformed list of maintenance items with relevant details.
   - The `/api/maintenance/update` endpoint handles updating maintenance details.
   - It validates the update field and updates the `Maintenance` model.
   - The `/api/maintenance/complete` endpoint handles updating maintenance status.
   - It updates the `Maintenance` and `Item` models based on the status update.
   - It creates a new record in the `MaintenanceHistory` model when maintenance is completed.

   Tracking Maintenance History:

   - The `Maintenancehistorydetails` component displays the maintenance history.
   - The component fetches maintenance history data from the backend using `fetch`.
   - The fetched data includes item IDs, maintenance statuses, completion dates, remarks, amounts, and item statuses.

- The maintenance history is displayed in a table.

Backend Processing (Maintenance History):

- The `/api/maintenance/maintenancehistory` endpoint fetches maintenance history data.
- It verifies the user's authorization using a JWT token.
- It fetches maintenance history records based on the user's room access and item type.
- It constructs a list of maintenance history items with relevant details.

Key Methods and Used:

- `React` - For building the user interface.
- `fetch` - For making HTTP requests to the backend API.
- `jsonwebtoken` - For verifying JWT tokens.
- `mongoose` - For interacting with the MongoDB database.
- `useState` and `useEffect` - For managing component state and fetching data.
- `map` and `filter` - For transforming and filtering maintenance data.
- `express` - For creating the backend API routes.
- `findByIdAndUpdate`, `findByIdAndDelete`, and `findOne` - For database operations.
- `TextField` and `select` - For user input.

9. **Stock Handover**

This feature enables the transfer of inventory responsibilities from one stock-in-charge to another. It includes sending handover requests to the HOD, appointing a new faculty member, and managing user credentials.

Initiating Handover:

- The `HandoverPopup` component allows the current stock-in-charge to initiate a handover request.
- The `handleHandover` function is triggered when the user confirms the handover.
- The function retrieves the user's email from the session token using `jwtDecode`.
- A POST request is sent to the `/api/handover` endpoint with the sender's email.
- The backend creates a handover notification and sends it to the HOD.

Backend Processing (Handover Request):

- The `/api/handover` endpoint handles the handover request.
- It retrieves the sender's email from the request body.
- It fetches the `room_no` associated with the sender's email from the `Access` model.
- It fetches the `room_name` from the `Room` model using the retrieved `room_no`.

- It creates a new handover notification using the `HandoverStockNotification` model.
- The notification includes the sender's email, a fixed receiver email (HOD), `room_no`, and `room_name`.
- The notification status is set to "unread".
- The notification is saved to the database.

Key Methods Used:

- `React` - For building the user interface.
- `fetch` - For making HTTP requests to the backend API.
- `jwtDecode` - For decoding JWT tokens to get user information.
- `express` - For creating the backend API routes.
- `mongoose` - For interacting with the MongoDB database.
- `useState` - For managing component state.
- `findOne` and `save` - For database operations.

10. **Add a Stock System**

This feature allows the Head of Department (HOD) to add a new premise, such as a classroom, lab, or workspace, to the department's inventory system. It involves creating a new data storage and assigning roles.

Adding a New Stock System:

- The `NewStockSystem` component provides a form for the HOD to enter details about the new premise.
- The form includes input fields for `room_no`, `name`, and `type`.
- The `useState` hook is used to manage the form data.
- The `handleChange` function updates the form data when input fields are changed.
- The `handleSubmit` function sends a POST request to the backend API to create the new premise.
- The backend creates a new data storage for the premise's inventory.

Backend Processing (Creating a New Premise):

- The backend API endpoint `/api/rooms` handles the creation of new premises.
- It receives the premise details from the request body.
- It creates a new entry in the database to store the premise's information.
- It creates a new, related data storage to record the inventory of the new premise.

Assigning Roles:

- The HOD can assign a stock-in-charge and a custodian for the new stock system.
- This assignment is handled through a separate interface or within the same form.

- The assignment process involves selecting users from a list and associating them with the new premise.
- The backend updates the user roles and access permissions accordingly.

Key Methods and Used:

- `React` - For building the user interface.
- `useState` - For managing component state.
- `fetch` - For making HTTP requests to the backend API.
- `Link` - For navigation within the application.
- `express` - For creating the backend API routes.
- `mongoose` - For interacting with the MongoDB database.

4.4 Summary

This chapter details the *Implementation* phase, covering User Interface, Database, and Connectivity Modules. It highlights tools like Express.js for API handling, Mongoose for MongoDB integration, and JWT for secure authentication. Express.js simplifies backend development, while Mongoose ensures efficient data management. bcryptjs strengthens security by hashing passwords. The next chapter, *Testing* , focuses on evaluating system functionality and performance.

Chapter 5

Testing

Testing [18] is a dynamic method for verification and validation, where the software to be tested is executed with carefully designed test cases and the behavior of the software system is observed. Testing intends to increase confidence in the correctness of the software. For this, the set of test cases used for testing should be such that for any defect in the system, there is likely to be a test case that will reveal it. To ensure this, the test cases must be carefully designed with the intent of revealing defects.

5.1 Testing Strategies Used

In this section, the strategies that were employed to test our project will be discussed. The following list outlines the testing strategies that were utilized in our project to ensure its quality and effectiveness.

5.1.1 Unit Testing

Once a programmer has written the code for a module, it has to be verified before it is used by others. Testing remains the most common method of this verification. At the programmer level testing done for checking the code the programmer has developed(as compared to checking the entire software system) is called unit testing [18].

Unit testing is like regular testing where programs are executed with some test cases except that the focus is on testing smaller programs or modules which are typically assigned to one programmer(or a pair) for coding. A unit may be a function or a small collection of functions for procedural languages, or a class or a small collection of classes for object-oriented languages [18]. The following unit testing strategies were used to test the developed system.

5.1.1.1 Black Box Testing

Black-box testing [18] is not concerned with the structure of the program. Test cases are decided solely based on the requirements or specifications of the program or module, and the internals of the module or the program are not considered for the selection of the test cases

In black-box testing, the tester only knows the input that can be given to the system and what output the system should give. In other words, the basis for deciding test cases is the requirements or specifications of the system or module. This form of testing is also called functional or behavioral testing. Some techniques for black-box testing include Equivalence class partitioning, Boundary value analysis, etc.

5.1.1.2 White Box Testing

White-box testing [18] is concerned with testing the implementation of the program. The intent of this testing is not to exercise all the different input or output conditions (although that may be a by-product) but to exercise the different programming structures and data structures used in the program. White-box testing is also called structural testing.

To test the structure of a program, structural testing aims to achieve test cases that will force the desired coverage of different structures. Since the intent of white-box testing is to test the structure of the code, various structures in the code like nodes, branches, and paths should at least be executed once during testing

5.1.2 Integration Testing

Integration testing [18] is the process of detecting any inconsistencies between the software units that are integrated.

The goal of integration testing is to see if the modules can be integrated properly. In this, many unit-tested modules are combined into subsystems, which are then tested. Hence, the emphasis is on testing interfaces between modules. This testing activity can be considered testing the design.

5.2 Testing Results

The above-mentioned testing strategies were implemented in our project. The following are the results obtained.

5.2.1 Results of Black Box Testing

Black box testing was conducted to evaluate the system's functionality from an end-user perspective. Testers performed operations such as login, user management, stock addition, stock verification, stock transfer, stock clearance, and generating reports. Errors identified during testing and their resolutions are as follows:

1. *Stock Addition*: Some invalid stock entries were accepted due to missing validation checks. The issue was resolved by enforcing field constraints for mandatory fields.
2. *Stock Verification*: Initially, verified stocks were not updating correctly in the database. This was due to an incorrect API response handling, which was corrected to ensure proper data updates.
3. *Stock Transfer*: The stock transfer notifications were not reaching the assigned faculty. This issue was fixed by adjusting the backend notification service to properly map recipients.

After resolving these issues, the system performed as expected across various modules.

5.2.2 Results of White Box Testing

White box testing focused on reviewing the source code, logic flow, and database interactions. A trial database was created with test data to simulate real-world scenarios. One significant issue encountered was:
API Connectivity Issue: Axios requests were failing due to incorrect base URLs. Initially, localhost URLs were used, which prevented access across networks. The issue was resolved by configuring API endpoints with dynamic environment-based URLs.

Post-testing, database consistency checks confirmed that stock transactions were accurately recorded and retrieved.

5.2.3 Results of Integration Testing

The integration of different system modules was tested to ensure seamless interactions. Some issues encountered included:

1. *Mismatch in Data Models*: Certain fields were missing when retrieving stock details due to inconsistencies in schema definitions. This was fixed by aligning frontend API calls with the backend database schema.

2. *Notification Delivery Failures*: Some notifications were not being sent due to missing recipient details. The notification system was updated to ensure accurate message delivery.

Once these issues were resolved, the integrated system was re-tested and verified to function correctly.

5.3 Summary

This chapter describes the testing strategies used to ensure the quality of the project. Unit testing verified individual code modules. Black-box testing focused on functionalities from the user's perspective, while white-box testing examined the code's internal structure. Finally, integration testing ensured all modules worked together seamlessly, meeting the desired nonfunctional requirements.The next chapter, *Results and Discussion*, presents the system's performance evaluation and key findings.

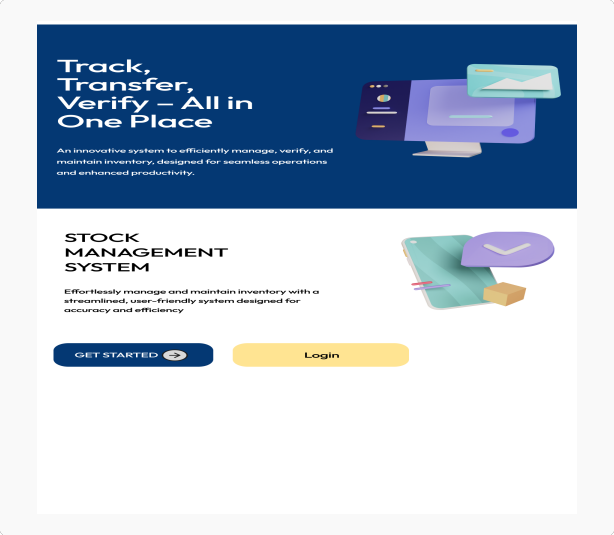
Chapter 6

Results and Discussion

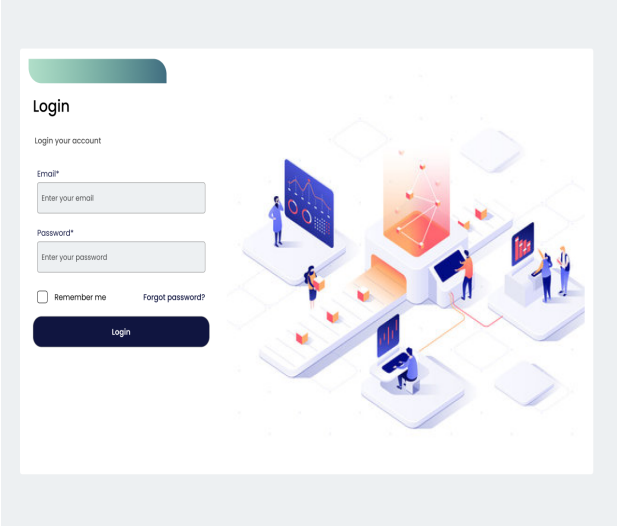
The DEPARTMENT STOCK MANAGEMENT SYSTEM was successfully implemented and the following results were obtained

6.1 Output Screenshots

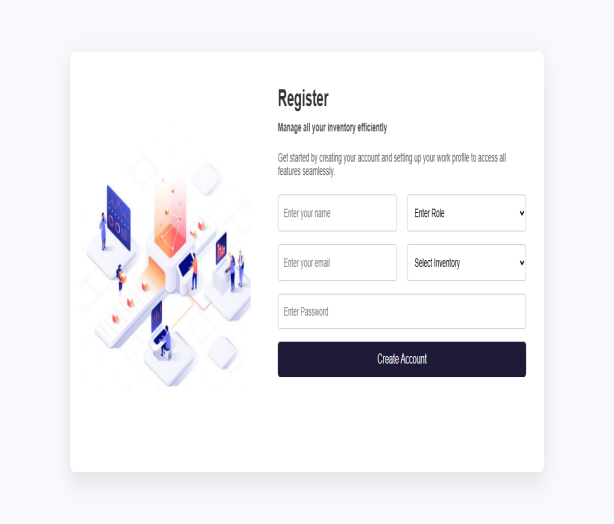
Given below are the screenshots of the results obtained after the implementation and testing of DEPARTMENT STOCK MANAGEMENT SYSTEM .



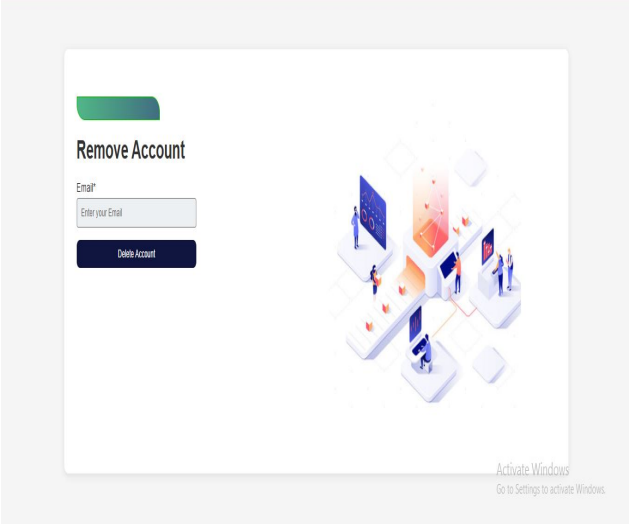
(a) Home Page



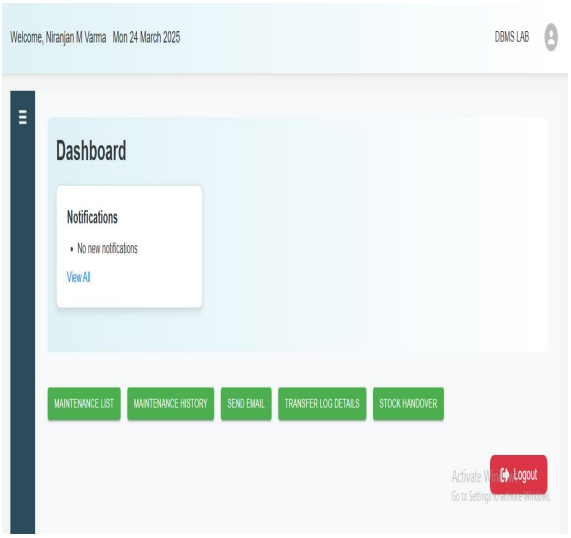
(b) User login



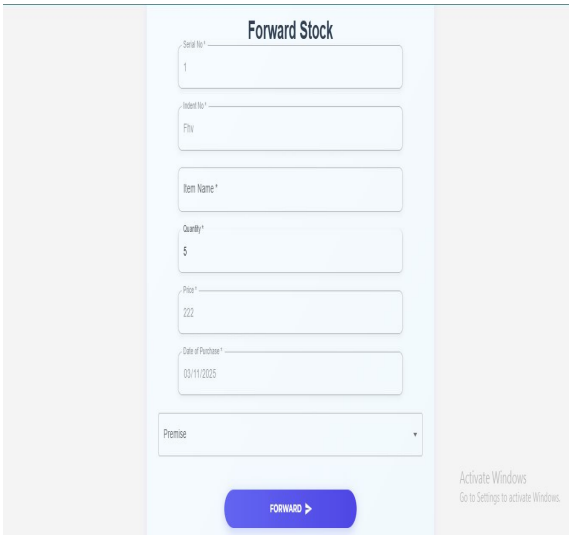
(c) Create account



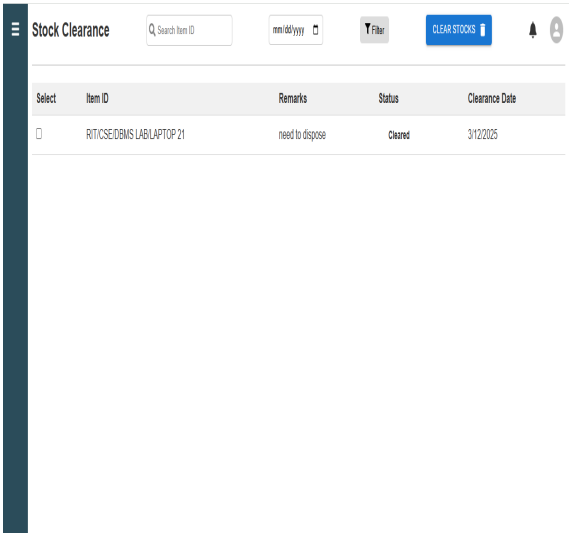
(d) Remove account



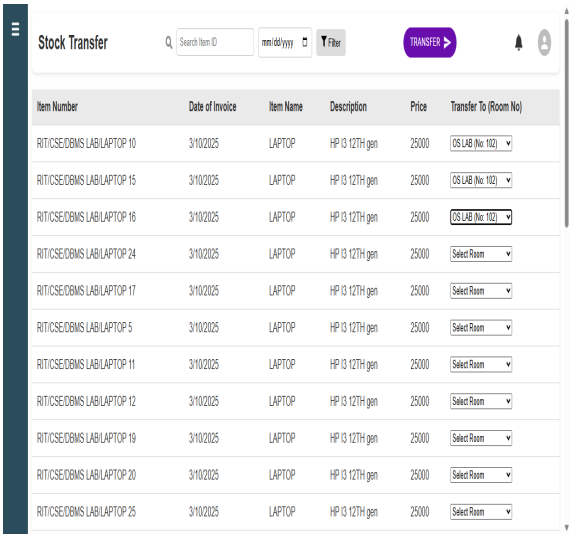
(a) Dashboard



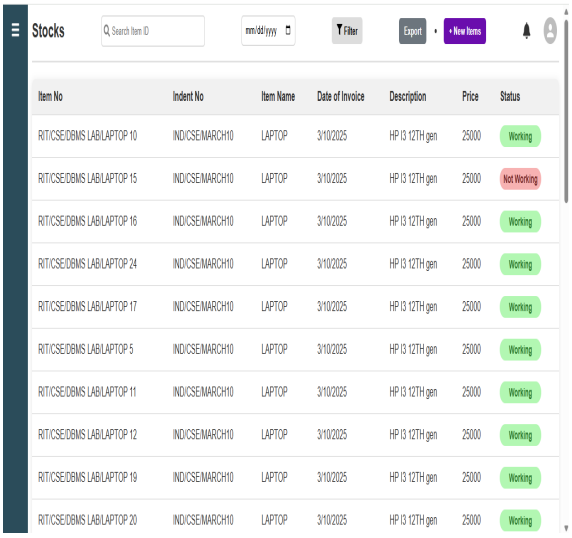
(b) Forward stock



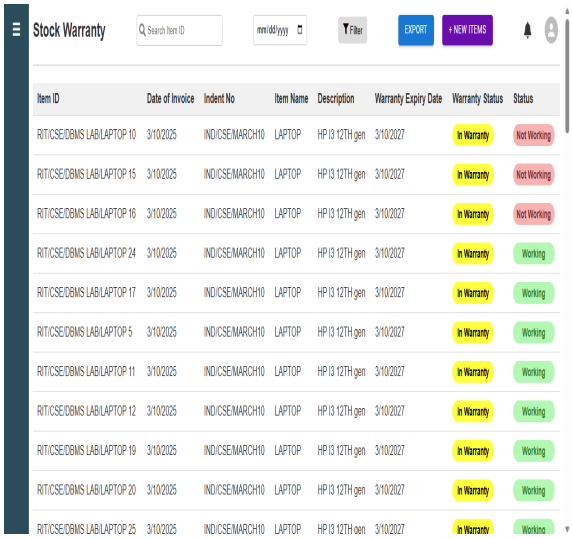
(c) Stock Clearance



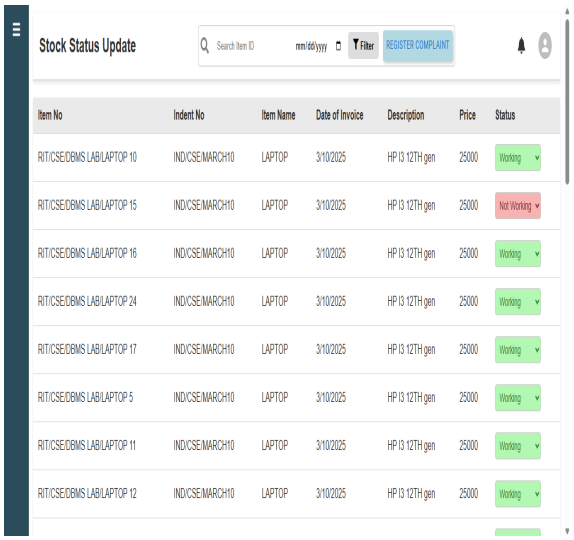
(d) Stock transfer



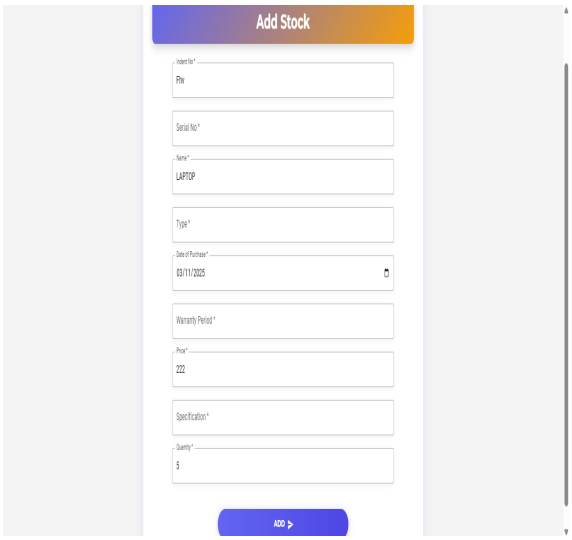
(e) Stock details



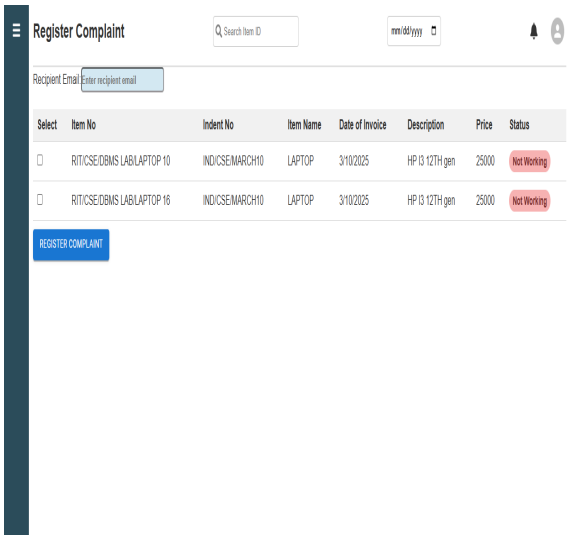
(f) Stock Warranty



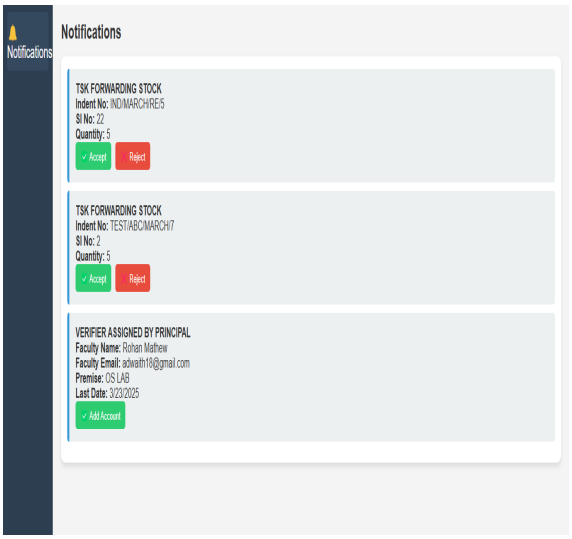
(a) Stock status



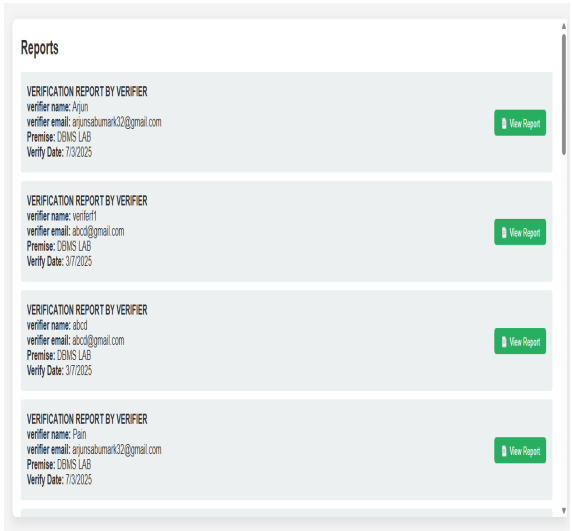
(b) Add stock



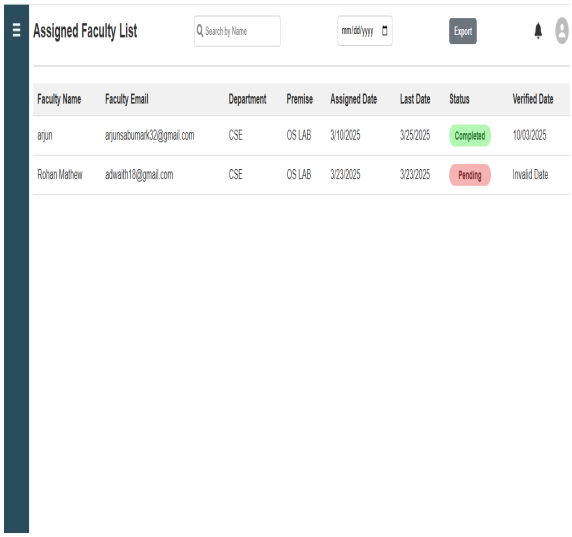
(c) Register complaint



(d) Notification



(e) Stock report



(f) Faculty list

6.2 Result Analysis

Result Analysis aims to analyze the data collected, draw meaningful conclusions, and evaluate the extent to which the project objectives have been achieved. Through a systematic and comprehensive analysis of the results, the following inferences were made.

1. The system meets all of its functional requirements.
2. The system meets all of its non-functional requirements as observed from the following.
    - Performance: The system ensures efficient execution of tasks, offering low response times and high reliability. The backend architecture is optimized for handling multiple concurrent requests without performance degradation.
    - Usability: The user interface (UI) is designed to be intuitive and user-friendly, ensuring a smooth experience for different user roles. The UI follows industry best practices for accessibility and ease of navigation.
    - Reliability: The system is designed to handle failures gracefully, ensuring uninterrupted service with minimal downtime. Automatic data backups and fail-safe mechanisms are incorporated to maintain data integrity.
    - Security: The system enforces stringent security measures, including role-based access control (RBAC), secure authentication, encrypted data transmission, and protection against unauthorized access or modifications.
    - Scalability: The architecture supports horizontal and vertical scaling, allowing the system to accommodate a growing number of users and inventory records without significant reconfiguration.
    - Maintainability: The system follows a modular design, allowing easy debugging, updates, and feature enhancements. Code maintainability is ensured through structured documentation and adherence to coding standards.
    - Portability: The system is cross-platform compatible, ensuring seamless operation on different devices and operating systems, including Windows, macOS, and Linux.

6.3 Summary

In this chapter, the sample output screens of different interfaces implemented are displayed along with what each screens are intended for. The analysis section gives a detailed analysis of the implemented features and their responsiveness.The next chapter, *Conclusion and Future Scope*, summarizes the project outcomes and discusses potential enhancements for further improving the system.

Chapter 7

Conclusion and Future Scope

In the preceding chapters, all the phases involved in the development of the Department Stock Management System have been discussed in detail. This project was initiated to address the challenges in managing departmental stock efficiently, ensuring proper tracking of assets and reducing discrepancies in inventory management.

Before the development process began, a thorough requirement analysis was conducted. This involved studying existing stock management systems and identifying gaps that needed to be addressed. The system requirements were finalized through detailed discussions with stakeholders, and a Software Requirements Specification (SRS) document was prepared to outline the functional and non-functional aspects of the system. These details were covered in Chapter 2.

Following this, the design phase was carried out, detailing the System Design and Detailed Design. The architecture was carefully planned using a modular approach, which included key modules such as Stock Management, User Authentication, Reports, and Notifications. The design followed best practices for maintainability and scalability, ensuring a structured and efficient workflow. These aspects were covered in Chapter 3.

Once the design phase was completed, the implementation phase began using modern web technologies such as React.js, MongoDB, Node.js, and Express.js. The development followed an iterative approach, ensuring incremental improvements and continuous integration of features. Upon completion of implementation, rigorous testing was performed, including unit testing, integration testing, and system testing. Issues found during testing were resolved to enhance system stability and performance. These details were discussed in Chapter 4.

The fully developed system was then deployed and evaluated based on functional and non-functional requirements. The results demonstrated that the system successfully met the objectives outlined in the requirement phase, providing an efficient and reliable stock management solution. The findings and analysis of the developed system were presented in Chapter 6.

Through this structured approach, the Department Stock Management System was successfully designed and implemented to streamline stock tracking, enhance security, and improve accessibility. The project serves as a foundation for further enhancements, ensuring scalability and adaptability to future departmental needs.

7.1 Future Scope

Although the system effectively manages departmental inventory, several enhancements can be implemented in the future to extend its functionality and usability across the entire institution:

1. *Institution-Wide Implementation*: Expanding the system to manage stock across all departments, integrating a centralized inventory for the entire college.

2. *Enhanced Maintenance Management*: Introducing a dedicated maintenance module with a streamlined interface for tracking repair requests, maintenance history, and automated service reminders

Appendices

Appendix A

Software Requirement Specification

A.1 Introduction

A.1.1 Purpose

The purpose of this document is to provide a detailed description about the requirements of the project Department Stock Management System. This document outlines the functional and non-functional requirements of the project, the constraints in which it operates, the deliverables, and the assumptions based on which it is built. It provides a clear understanding of the system's capabilities and constraints to its readers.

A.1.2 Document Conventions

AWS: Amazon Web Service
TSK : Technical Stock Keeper

A.1.3 Intended Audience and Reading Suggestions

1. Team members
2. Project Guide
3. Project coordinator
4. Anyone who is interested in developing similar stock management system for an organization.

A.1.4 Project Scope

The scope of the project includes:

1. *Need:-*
    - Provide a secure, role-based system for faculty and principals to streamline inspection workflows and maintain stock records.
    - Enable centralized access to inventory data for better decision-making and transparency.
    - Automate inventory management to reduce manual effort and errors.
2. *Deliverables:-*
    - User registration and login capabilities.
    - Role-based dashboards for principals and faculty.

- Automated workflows for assigning, inspecting, and reviewing inventory tasks.
- Secure, cloud-hosted storage for inventory and user data.

3. *Exclusions:-*
   - The system does not cover inventory management for departments outside of Computer Science.
   - Mobile application development is not included.
   - The system does not automate the manual stock verification.
4. *Assumptions:-*
   - The department will provide accurate initial data to populate the inventory database.
   - User roles and permissions are predefined and will not require frequent changes.

A.1.5 References

IEEE Standards Association. (1998). IEEE 830-1998 - IEEE Recommended Practice for Software Requirements Specifications.

A.2 Overall Description

A.2.1 Product Perspective

The Department Stock Management System is a specialized web application designed to streamline and automate the inventory management processes within the Computer Science Engineering Department of RIT Kottayam. This system integrates role-based workflows Principal, HOD, Stock In Charge and Custodian, ensuring secure and efficient management of departmental assets like computers, furniture, and other lab equipment.

A.2.2 Product Features

1. User Authentication
2. Stock Allocation
3. Stock Verification
4. Stock Transfer
5. Stock Maintenance
6. Stock Clearance
7. Stock Handover
8. User Based Notification
9. Report Generation
10. Add a Stock System

A.2.3 User Classes and Characteristics

1. Principal
2. HOD
3. Faculty In charge
4. Custodian
5. Technical Stock Keeper

A.2.4 Operating Environment

The Hardware and Software required for this project are mentioned in section 4.2 and section 4.3. The client-side operates on desktops, laptops, or tablets using widely supported web browsers such as Google Chrome, Mozilla Firefox, and Microsoft Edge. On the server side, the system utilizes Django as the backend framework and ReactJS for the frontend. Data storage is managed using MongoDB, ensuring secure and efficient database operations. The system supports deployment on cloud platform AWS for scalability or can be hosted on on-premises servers.

A.2.5 Design and Implementation Constraints

The implementation is limited to a small development team to maintain simplicity and manage resource constraints effectively. Collaborative development is supported using tools like Git for version control, enabling seamless integration of modular components. The project adheres to well-known design conventions and standardized programming practices to enhance code readability, scalability, and ease of maintenance.The use of Django and ReactJS ensures the system can be easily modified and expanded in the future.

A.2.6 Assumptions and Dependencies

The key assumption of the project include

1. The user is an employee of the institution.
2. The user has an active role in stock keeping.

A.3 System Features

The system has following features.

1. User Authentication
2. Stock Allocation
3. Stock Verification
4. Stock Transfer

5. Stock Maintenance
6. Stock Clearance
7. Stock Handover
8. User Based Notification
9. Report Generation
10. Add a stock System

A.3.1 User Authentication

A.3.1.1 Description and Priority

This feature ensures secure access to the system based on user roles, Each valid user is suppose to be authenticated by the system so as to have access to the system. This feature enables to provide different interfaces and functionalities for various users, depending on their roles. This in turn ensures accountability and smooth workflow management.
Priority: High

A.3.1.2 Stimulus/Response Sequences

- Stimulus: User credentials.
- Response: Successful login if the user is an authenticated user. An Error message if the user is not an authenticated user.

A.3.1.3 Functional Requirements

1. The system must provide a secure login mechanism that validates users based on their credentials.
2. Failed login attempts should trigger alerts after a certain threshold.

A.3.2 Stock Allocation

A.3.2.1 Description and Priority

This functionality ensures allocation of inventory items (e.g., computers, furniture) to specific classrooms, labs within department based on requirements.
Priority: High

A.3.2.2 Stimulus/Response Sequences

- Stimulus: Notification send by TSK to the HOD for allocating some inventory items into one of the department premises.
- Response: The items get allocated to the respective premises and notifications are send to stock-in-charge, HOD and TSK .

A.3.2.3 Functional Requirements

1. The HOD must be able to forward the notification send by the TSK to stock-in-charge of respective premises.
2. The stock-in-charge must be able to forward the notification send by the HOD to the respective custodian.
3. The custodian accepts the notification from the stock in charge and inserts the items into the respective stock inventory.
4. The stock notifications must be send to stock in charge, HOD and TSK to notify that the inventory items are successfully allocated to the respected premises.

A.3.3 Stock Verification and Remarks

A.3.3.1 Description and Priority

This feature allows the annual stock verification process and the submission of the annual report in online mode.
Priority: High

A.3.3.2 Stimulus/Response Sequences

- Stimulus: Assigned Faculty performs a stock verification for their assigned inventory.
- Response: A Report is forwarded to the principal and after verification the system updates the condition of items and logs remarks for future reference.

A.3.3.3 Functional Requirements

1. Principal must be able to assign a faculty for verifying the stock and send a notification to the respective persons.
2. The HOD must be able to create a temporary login credentials for each faculty who were assigned stock verification duty by the principal and to send these details to the respective faculty.
3. Each Faculty who was assigned the stock verification duty must be able to view the items of the respective premises and must be able to mark the present status of the items (working, damaged, or under maintenance). Further, on completion of the status entries, the faculty must be able to submit the inferences to the principal.
4. The Principal must be able to access the inferences sent by the faculty who were assigned stock verification duty as separate reports and approve them.
5. The HOD must be able to delete the temporary user credentials on completion of the respective stock verification an the approval of the principal.

A.3.4 User Based Notification

A.3.4.1 Description and Priority

This feature ensures timely updates for all users about assigned tasks, approvals, and changes in inventory status, enhancing communication and transparency.

Priority: High

A.3.4.2 Stimulus/Response Sequences

- Stimulus: Successful Login of users.
- Response: System sends notifications to corresponding users.

A.3.4.3 Functional Requirements

1. Notifications should be available on the user dashboard.
2. Alerts must include task details, user assignment updates, and approval updates.

A.3.5 Report Generation

A.3.5.1 Description and Priority

This feature enables filtered and comprehensive reports about inventory items based on specific criteria. It supports on-demand report generation for better decision-making and resource planning. **Priority: High**

A.3.5.2 Stimulus/Response Sequences

- Stimulus: Specific filter options chosen by the user
- Response: Reports based on specified filter options.

A.3.5.3 Functional Requirements

1. The Users must be able to select various filter options including the item type, condition (working, damaged, under maintenance), location and other specifications.
2. Reports should support export in formats like PDF and Excel for offline use.

A.3.6 Stock Transfer

A.3.6.1 Description and Priority

This feature enables the stock-in-charge to transfer items from one premise to another premise.

Priority: High

A.3.6.2 Stimulus/Response Sequences

- Stimulus: The stock-in-charge initiates the transfer of items from one inventory to another.
- Response: The item is removed from the source inventory and added to the specified destination inventory, with records updated accordingly.

A.3.6.3 Functional Requirements

1. The stock-in-charge must be able to select the destination premise and specify the item to be transferred.
2. The stock-in-charge of the destination premise must receive the notification to accept or reject the item.
3. A provision to update the stock of both the premises if the stock-in-charge of the respective premise accepts the notification.
4. A provision to send notifications regarding the transfer to the custodians of both the premises.

A.3.7 Stock Clearance

A.3.7.1 Description and Priority

This feature allows the custodian to remove damaged or unwanted items from the premise.
Priority: High

A.3.7.2 Stimulus/Response Sequences

- Stimulus: The item(s) to be removed
- Response: The removal of the selected items from the respective premise.

A.3.7.3 Functional Requirements

1. The stock-in-charge of the respective premise must be able to identify and select items that are eligible for clearance.
2. Once cleared, the system should automatically update the respective stock records to reflect the removed items.
3. A provision to send notifications to relevant parties.
4. A data storage to keep up the records of all the cleared items.

A.3.8 Stock Maintenance

A.3.8.1 Description and Priority

This feature ensures that the maintenance details of all inventory items is recorded. It allows users to identify and log items requiring repair or maintenance.

Priority: High

A.3.8.2 Stimulus/Response Sequences

- Stimulus: A notification sent by stock-in-charge of the item requiring maintenance to the party entrusted to do the maintenance.
- Response: The details of the maintenance performed on the item.

A.3.8.3 Functional Requirements

1. There must be a provision to send an email to the service provider.
2. Completed task should update the item status and maintenance details in inventory database.

A.3.9 Stock Handover

A.3.9.1 Description and Priority

This feature enables the transfer of inventory responsibilities from one stock-in-charge to another stock-in-charge.

Priority: low

A.3.9.2 Stimulus/Response Sequences

- Stimulus: The faculty sends a handover request to HOD.
- Response: HOD accepts the request and appoints a new faculty.

A.3.9.3 Functional Requirements

1. The party who wants to transfer the responsibility must be able to send a notification to HOD.
2. The HOD must be able to appoint a new faculty as stock-in-charge.
3. The HOD must be able to create a temporary login credentials for the new faculty.
4. The HOD must be able to remove the login credentials of the previous stock-in-charge.
5. Notifications regarding the stock handover must be sent to both the parties.

A.3.10 Add a Stock System

A.3.10.1 Description and Priority

This feature allows the addition of a new premise such as classroom, lab or any other workspace within the department.

Priority: High

A.3.10.2 Stimulus/Response Sequences

- Stimulus: The HOD add a new stock system.
- Response: A new inventory database is created.

A.3.10.3 Functional Requirements

1. The system must allow the creation of a new data storage for recording the details of the new stock system.
2. The HOD must be able to assign stock-in-charge and the custodian for the new stock system.

A.4 External Interface Requirements

A.4.1 User Interfaces

- Login page:- Allows various users to log in to the system based on their roles.
- Principal Dashboard:- Provides an overview of the entire department's inventory, including stock levels, recent activity, requests for approval and assign inspection duty
- HOD Dashboard:- Displays detailed information about stock levels, pending verification tasks, and transfer requests within their department.
- Faculty In-charge Dashboard:- Allows the Lab In-charge to enter new stock items, verify existing stock, and manage transfers of inventory items between labs.
- Custodian Dashboard:- Enables the custodian to view stock levels, assist with stock verification, and facilitate the transfer of items between different rooms or labs.
- TSK Dashboard:-

A.4.2 Hardware Interfaces

- Intel Core i3 processor or above
- Minimum 8 GB RAM

A.4.3 Software Interfaces

- Windows 10 or above/Mac/Linux
- MongoDB DBMS
- Any web browser

A.5 Other Nonfunctional Requirements

A.5.1 Performance Requirements

(a) Reliability:- Reliability is the ability to consistently perform its intended functions over time and under different conditions without unexpected failures or errors. The product is expected to have high reliability

(b) Response time:- Response time refers to the amount of time it takes for the web server to respond to a request made by a user.The product is expected to have a low response time.

A.5.2 Security Requirements

(a) Access Control:- The system should ensure that only authorized users can access or modify stock records.

(b) Role-Based Access Control:- Ensure users have access only to the resources and operations for which they are authorized.

A.5.3 Software Quality Attributes

(a) Usability:- The Usability refers to the ease with which the system can be used by the users of the system. It is expected to have high usability for the project under consideration.

(b) Testability and Maintainability:- The testability refers to how effectively the system can be evaluated and validated to meet its performance and functionality requirements. The Maintainability refers to how easily the system can be repaired and improved. It is expected to have high testability and maintainability for the project under consideration.

A.6 Other Requirements

There are no other requirements in this project

References

[1] Hemant Kumar. Utilization of inventory management system in educational organization. *The Academic*, 2(10):38–45, 2024.

[2] ASAP Systems. Sparrow academy school inventory & asset tracking case study, 2024.

[3] E. S. Soegoto and A. F. Palalungan. Web-based online inventory information system. *IOP Conference Series: Materials Science and Engineering*, 879:1–6, 2020.

[4] Ian Sommerville. *Software Engineering (tenth edn. global edition)*. Boston, MA: Pearson Education, Inc, 2016.

[5] Roger S Pressman. *Software engineering: a practitioner's approach*. Palgrave macmillan, 2005.

[6] figma documentation. `https://help.figma.com/hc/en-us`, 2024.

[7] figma wikipedia. `https://en.wikipedia.org/wiki/Figma`, 2024.

[8] Lucidchart. `https://www.lucidchart.com`, 2024.

[9] mongodb docs. `https://www.mongodb.com/docs/`, 2024.

[10] Node.js v22.1.0 documentation. `https://nodejs.org/docs/latest/api/`, 2024.

[11] Express-node mdn documentation. `https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs`, 2024.

[12] Introducing react.dev. `https://react.dev/blog/2023/03/16/introducing-react-dev`, 2024.

[13] Vite documentation. `https://vitejs.dev`, 2024.

[14] Render documentation. `https://render.com/docs`, 2024.

[15] Vercel documentation. `https://vercel.com/docs`, 2024.

[16] Documentation for visual studio code. `https://code.visualstudio.com/docs`, 2024.

[17] Github docs. `https://docs.github.com/en`, 2024.

[18] Pankaj Jalote. *An integrated approach to software engineering*. Springer Science & Business Media, 2012.

Index