HOSTEL MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

BHARANIDHARAN B
(Reg. No: 24MCR010)

INIYAN B
(Reg. No: 24MCR035)

JAYASURYA S
(Reg. No: 24MCR041)

in partial fulfillment of the
requirements for the award of the
degree of

MASTER OF COMPUTER APPLICATIONS

DEPARTMENT OF COMPUTER APPLICATIONS

KONGU ENGINEERING COLLEGE
(Autonomous)

PERUNDURAI, ERODE – 638 060

DECEMBER 2024

DEPARTMENT OF COMPUTER APPLICATIONS

KONGU ENGINEERING COLLEGE

(Autonomous)

PERUNDURAI, ERODE – 638 060

DECEMBER 2024

BONAFIDE CERTIFICATE

This is to certify that the project report entitled **"HOSTEL MANAGEMENT SYSTEM"** is the bonafide record of project work done by **BHARANIDHARAN B (24MCR010), INIYAN B (24MCR035) and JAYASURYA S (24MCR041)** in partial fulfilment of the requirements for the award of the Degree of Master of Computer Applications of Anna University, Chennai during the year 2024-2025.

SUPERVISOR **HEAD OF THE DEPARTMENT**

(Signature with seal)

Date:

Submitted for the end semester viva voce examination held on _________________

INTERNAL EXAMINER **EXTERNAL EXAMINE**

DECLARATION

We affirm that the project report entitled **"HOSTEL MANGEMENT SYSTEM"** being submitted in partial fulfilment of the requirements for the award of Master of Computer Applications is the original work carried out by us. It has not formed the part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

BHARANIDHARAN B
(Reg. No: 24MCR010)

INIYAN B
(Reg. No: 24MCR035)

JAYASURYA S
(Reg. No: 24MCR041)

Date:

I certify that the declaration made by the above candidates is true to the best of my knowledge.

Date:

Name and Signature of the supervisor

(Dr. L RAHUNATHAN)

ABSTRACT

The Hostel Management System is designed to make managing groceries and stock easier and more efficient. It has two main parts: the Grocery Approval System and the Stock Management Module. The Grocery Approval System allows caretakers to request groceries they need for the hostel. These requests are then reviewed and approved by the wardens or administrators. This process ensures that there is clear communication and that the grocery needs are properly managed and controlled. The Stock Management Module helps keep track of the groceries available in the hostel. Caretakers can update the stock levels, check how much inventory is left, and make sure that there is neither too much nor too little stock.

This helps avoid problems like overstocking which can lead to waste or running out of important supplies. Together, these two modules help manage resources efficiently and keep the hostel running smoothly. The Hostel Management System is built using up-to-date web technologies. For the frontend is the part users interact with, it uses React, which helps create a fast and responsive interface. The backend is the part that handles the logic and data is powered by Django, a powerful framework for web development. The system stores data in MongoDB, a type of database that helps manage and organize information efficiently. The user interface is designed to be simple and easy to use, making sure it works well on all types of devices, whether it's a computer, tablet, or smartphone.

Although the system currently focuses on managing groceries and stock, it is designed in a way that makes it easy to add other features for hostel management in the future.

ACKNOWLEDGEMENT

We respect and thank our correspondent **Thiru. A.K.ILANGO BCom., MBA., LLB.,** and our Principal **Dr.V.BALUSAMY BE(Hons)., MTech., PhD** Kongu Engineering College, Perundurai for providing us with the facilities offered.

We convey our gratitude and heartfelt thanks to our Head of the Department Professor **Dr.R.THAMILARASI ME., PhD** Department of Computer Applications, Kongu Engineering College for his perfect guidance and support that made this work to be completed successfully.

We also like to express our gratitude and sincere thanks to our project coordinators **Dr.T.KAVITHA MCA., MPhil., PhD.,** Assistant Professors (Sr.G), Department of Computer Applications, Kongu Engineering college who have motivated us in all aspects for completing the project in scheduled time.

We would like to express our gratitude and sincere thanks to our project guide **Dr.L.RAHUNATHAN MSc., MPhil., PhD., MTech.,** Professor, Department of Computer Applications, Kongu Engineering College for giving her valuable guidance and suggestions which helped us in the successful completion of the project.

We owe a great deal of gratitude to our parents for helping us to overwhelm in all proceedings. We bow our heart and head with heartfelt thanks to all those who thought us their warm services to succeed and achieve our work.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

ABBREVIATION	EXPLANATON
JS	JavaScript
UI	User Interface
API	Application Programming Interface
DB	Database
SRS	Software Requirement Specification

CHAPTER 1

INTRODUCTION

1.1 ABOUT THE PROJECT

Efficient management of resources is very important for running a hostel, especially when it comes to handling groceries and stock. The Grocery Approval System helps create a clear and organized process where caretakers can request the grocery items they need. These requests are then sent to wardens and administrators for approval, ensuring that every step is tracked and that there's accountability for the decisions made.

The Stock Management Module allows caretakers to keep an eye on the inventory and update it whenever groceries are used or restocked. This system replaces manual processes, which are often slow and prone to mistakes, with a faster and more accurate digital solution. By combining these two modules, the system enhances the accuracy, transparency, and speed of managing resources in the hostel.

The software is designed to be simple to use and provides real-time data, making it easier to make quick decisions. The use of modern technologies ensures that the system is secure and can grow as needed, meeting future demands.

1.2 EXISTING SYSTEM

The current system for hostel management relies heavily on manual processes or outdated digital methods, which hinder efficiency and reliability. Records, room assignments, and grocery stock levels are typically maintained in physical registers or basic spreadsheets. This approach not only makes data retrieval slow and cumbersome but also increases the likelihood of errors and data loss. The lack of a centralized system for managing user roles (Admin, Warden, Caretaker) further complicates task delegation and accountability.

Grocery stock management is another critical area where the existing system falls short. Inventory levels are updated manually, often leading to discrepancies and mismanagement of resources. Grocery requests are submitted through informal communication channels, such as handwritten notes or verbal messages, making it difficult to track approvals and maintain transparency. There is no efficient way to monitor the status of requests, resulting in delays and inefficiencies in the workflow.

Furthermore, the existing system lacks real-time updates and consolidated reporting. Administrators, wardens, and caretakers must rely on periodic, manual reports to understand stock levels and request statuses, which is both time-consuming and error-prone. This fragmented approach also makes auditing and accountability challenging. Overall, the existing system fails to meet the demands of a modern hostel environment, necessitating a more robust, automated, and centralized solution.

1.2.1 DRAWBACKS OF EXISTING SYSTEM

The current system for managing hostels has many problems that make it slow and inefficient. Most tasks, managing grocery stock, and handling requests, are done manually. This often leads to mistakes and takes a lot of time. Information is kept in separate files or spreadsheets, which makes it hard to share or update quickly. There is no single system where all the data can be stored and accessed easily, causing confusion and delays for everyone involved, such as Admins, Wardens, and Caretakers. Another issue is that the system does not update stock levels or request statuses in real time, making it hard to make quick decisions.

The process for grocery requests is especially difficult. Requests are made informally, either by writing them down or speaking directly, which means there is no clear way to track or check them. This lack of a proper system causes delays and makes it harder to ensure the process is fair and efficient. Managing all the data manually also increases the chance of losing important records or making errors, which makes the system unreliable. These issues show that a better and more organized system is needed to fix these problems and make hostel management easier and more effective.

1.3 PROPOSED SYSTEM

The proposed system is designed to address the problems of the current manual system and make hostel management more efficient and reliable. This system will be completely automated and centralized, allowing all data to be stored and accessed from a single platform. Tasks like user registration, grocery stock management, and request approvals will be handled digitally, reducing errors and saving time. Admins, Wardens, and Caretakers will have specific roles, with clear permissions and responsibilities, ensuring a smooth workflow.

The system will enable caretakers to submit grocery requests through a structured form, which can then be approved or rejected by wardens and overseen by admins. All request statuses will be updated in real time, making the process faster and more transparent. Grocery stock levels will also be tracked digitally, ensuring that inventory is always accurate and up to date. Reports on stock levels, user details, and request histories can be generated quickly, helping in better decision-making.

By using modern tools like React for the frontend and Node.js with MongoDB for the backend, the system will be fast, scalable, and secure. This new system will not only save time but also reduce the chances of data loss or misuse, providing a reliable solution for hostel management.

1.4 ADVANTAGES OF THE PROPOSED SYSTEM

The proposed system brings many benefits that make hostel management faster, easier, and more reliable. One of the biggest advantages is that all data is centralized, meaning everything from user information to stock levels can be stored in one place and accessed by authorized people anytime. This reduces the need for manual record-keeping and minimizes errors. Real-time updates ensure that grocery stock levels and request statuses are always accurate and up to date, which helps in quick decision-making and avoids delays.

The system improves transparency by providing a clear workflow for grocery requests. Caretakers can submit requests online, and wardens and admins can approve or reject them with just a few clicks. Everyone involved can track the status of requests, making the process more efficient and fair. Additionally, generating reports on stock levels or request histories becomes much faster and easier, saving time and effort for administrators.

Another advantage is that the system is scalable and secure. Built with modern tools like React, Node.js, and MongoDB, it can handle large amounts of data and users without slowing down. Role-based access control ensures that only authorized users can perform specific tasks, reducing the risk of data misuse. Overall, the proposed system saves time, reduces errors, and provides a reliable solution for managing hostel operations.

1.5 SUMMARY:

This chapter explained the existing system and its drawbacks, as well as the proposed solution and its advantages. The current system is inefficient because it relies on manual processes, scattered records, and lacks real-time updates. Tasks like grocery requests and stock management are prone to delays and errors, which make the workflow slow and unreliable. The proposed system solves these issues by introducing an automated and centralized platform where all operations are streamlined. It allows real-time updates, role-based access, and faster decision-making, ensuring that hostel management is more efficient and transparent.

CHAPTER 2

SYSTEM ANALYSIS

2.1 IDENTIFICATION OF NEED

Managing a hostel involves handling many responsibilities, managing grocery stock, and processing requests efficiently. The existing manual system struggles to keep up with these demands due to errors, delays, and lack of transparency. As hostels grow larger and handle more students and staff, the challenges increase, making it harder to manage operations smoothly. Tasks like tracking stock levels, processing grocery requests, and ensuring accountability require a system that is fast, accurate, and reliable.

The need for an automated and centralized system arises from these challenges. By replacing manual processes with a digital platform, hostel administrators can save time, reduce errors, and improve overall efficiency. Real-time updates and role-based access will allow caretakers, wardens, and admins to collaborate better and stay informed about important activities. A system that provides clear workflows and easy reporting will not only improve daily operations but also ensure transparency and accountability. The proposed system fulfills this need, making hostel management easier, faster, and more reliable for all stakeholders.

2.2 FEASABILITY STUDY

A f Before implementing the proposed system, it is important to check if the solution is practical and achievable. A feasibility study evaluates whether the system can be developed effectively based on

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility

These aspects ensure that the project is realistic and provides value to the users.

2.2.1 ECONOMIC FEASIBILITY

The system is cost-effective to develop and maintain because it uses open-source tools and technologies. MongoDB, Node.js, Express, and React are free to use, which eliminates licensing costs. Hosting the system on a local server or cloud platform requires minimal investment, and the benefits of increased efficiency, reduced errors, and time savings outweigh these costs. By replacing manual processes, the system reduces administrative overhead and ensures a faster return on investment. Overall, the project is economically viable.

2.2.2 TECHNICAL FEASIBILITY

The proposed system is technically feasible because it uses modern and widely accepted technologies like MongoDB, Express, React, and Node.js (MERN stack). These technologies are well-supported, scalable, and suitable for developing a robust web-based system. With MongoDB's document-oriented database, the system can handle large amounts of data without performance issues. Additionally, the use of React ensures a user-friendly interface, while Node.js and Express provide a fast and secure backend. Since these tools are already familiar to many developers and have extensive community support, implementing the system is technically achievable.

2.2.3 OPERATIONAL FEASIBILITY:

From an operational perspective, the system is easy to integrate into the daily processes of hostel management. Caretakers, wardens, and admins can quickly adapt to the system because of its intuitive user interface and streamlined workflows. The system reduces the workload by automating tasks like grocery requests, stock management, and user registrations. Training the staff to use the system will be simple due to its logical design and ease of use. Additionally, the system ensures better coordination among roles, making operations smoother and more transparent.

2.3 SOFTWARE REQUIREMENT SPECIFICATION

The system specifications include the hardware and software requirements needed to develop and run the proposed hostel management system. These specifications ensure that the system operates efficiently and meets user expectations.

2.3.1 Hardware Requirements

The system requires basic hardware components to run both the backend server and the frontend application. The following are the hardware specifications:

- **Processor**: Intel Core i3 or higher
- **RAM**: 4GB or more (8GB recommended for better performance)
- **Storage**: 50GB or more of free space
- **Display**: Monitor with a resolution of 1366x768 or higher
- **Network**: Stable internet connection for accessing APIs and the database

These specifications are sufficient for hosting the application locally or on a server.

2.3.2 Software Requirements

The software requirements outline the tools and technologies used for development and deployment:

- **Operating System**: Windows, macOS, or Linux
- **Frontend**: React.js (with Node.js for development)
- **Backend**: Node.js with Express.js
- **Database**: MongoDB (NoSQL database)
- **Code Editor**: Visual Studio Code or any preferred editor
- **Tools**:
    - Postman for API testing
    - Git for version control
    - npm or yarn for package management

These software tools and technologies ensure smooth development and deployment of the project.

2.4 FRONTEND AND BACKEND

2.4.1 REACT

React is a popular JavaScript library developed by Facebook that is widely used for building dynamic and interactive user interfaces. It is especially useful for creating single-page applications where the content changes frequently. React uses a component-based structure, allowing developers to break the user interface into small, reusable pieces of code. This approach not only makes development faster but also makes the code more organized and easier to maintain. One of the key features of React is the Virtual DOM, a lightweight representation of the actual DOM. When changes occur in the application, the Virtual DOM determines the minimal set of updates needed to apply those changes to the real DOM, resulting in faster performance and smoother user experiences.

Another strength of React is its declarative programming style, where developers describe the desired outcome, and React handles the rendering process. This makes code more predictable and easier to debug. React also supports unidirectional data flow, which means that data moves in a single direction through the application, making it easier to track and manage changes. Furthermore, React has a large ecosystem and community, offering a wealth of tools, libraries, and resources to extend its functionality. Developers can integrate React with backend technologies, use it for mobile app development with React Native, or enhance their projects with state management libraries like Redux.

React is highly flexible and scalable, making it suitable for projects of any size, from simple websites to complex web applications. Its popularity has grown significantly due to its performance, developer-friendly nature, and the extensive support available in its ecosystem. Whether you're building a small personal project or a large enterprise-level application, React provides the tools and features needed to create responsive, modern, and efficient web applications.

2.4.2 FEATURES OF REACT

1. **Component-Based Architecture**
   - Allows developers to build reusable and independent pieces of UI.
   - Promotes modularity and code reusability.

2. **Virtual DOM**
    - Provides a lightweight representation of the real DOM.
    - Updates only the necessary parts of the actual DOM, ensuring high performance.
3. **JSX (JavaScript XML)**
    - Combines HTML and JavaScript syntax for defining component structure.
    - Improves readability and simplifies code writing.
4. **Cross-Platform Development**
    - Use React Native to build mobile applications for iOS and Android.
    - Share logic and components across platforms.
5. **High Performance**
    - Efficient updates through the Virtual DOM.
    - Optimized rendering process.
6. **Flexibility and Scalability**
    - Suitable for projects of all sizes, from simple static pages to large-scale applications.
    - Integrates well with other technologies and frameworks.

2.4.3 NODE

Node.js is a powerful JavaScript runtime that allows developers to run JavaScript code on the server-side. Unlike traditional JavaScript, which is typically run in the browser, Node.js enables the execution of JavaScript outside of the browser, making it ideal for backend development. Built on the V8 JavaScript engine (the same engine used by Google Chrome), Node.js is known for its high performance and scalability. It is particularly well-suited for building applications that need to handle many concurrent connections, such as real-time applications, APIs, and streaming services.

One of the core features of Node.js is its non-blocking, event-driven architecture, which allows it to handle multiple requests simultaneously without waiting for each task to complete. This makes Node.js a great choice for applications that require high I/O operations, like chat applications, data streaming, or APIs that need to respond quickly. Its asynchronous nature also helps in improving performance and minimizing downtime, as it doesn't need to wait for processes like reading from a database or making HTTP requests.

Node.js is commonly used to build web servers, REST APIs, and microservices. It's also ideal for building real-time applications such as messaging platforms, online gaming, and collaborative tools. The vast npm (Node Package Manager) ecosystem provides thousands of pre-built libraries and modules, making it easy to add functionality to a project without having to build everything from scratch. Popular frameworks like Express.js further simplify backend development by providing powerful tools for routing, middleware handling, and managing server requests.

Because Node.js uses JavaScript, it allows for full-stack development with a single language, meaning both the front-end and back-end code can be written in JavaScript. This streamlines development, reduces context switching, and allows developers to work more efficiently. Node.js is used by many large-scale companies, including Netflix, LinkedIn, and Uber, due to its speed, scalability, and the ability to handle high-volume, real-time operations.

2.4.4 FEATURES OF NODE

1. **Asynchronous and Event-Driven Architecture**
   - Handles multiple requests simultaneously without blocking the thread.
   - Ideal for high I/O operations and real-time applications.
2. **High Performance with V8 Engine**
   - Built on Google Chrome's V8 engine for fast JavaScript execution.
   - Optimized for scalability and speed.
3. **Cross-Platform Compatibility**
   - Runs on various operating systems, including Windows, macOS, and Linux.
   - Ideal for building scalable, platform-independent applications.
4. **Extensive npm Ecosystem**
   - Access to thousands of pre-built modules and libraries via Node Package Manager (npm).

5. **Single-Language Full-Stack Development**
    - Enables developers to write both front-end and back-end code in JavaScript.
    - Streamlines development and reduces context switching.

Node.js's combination of performance, scalability, and extensive tooling makes it a top choice for building modern, efficient, and real-time web applications.

2.4.5 MONGODB

MongoDB is a popular NoSQL database designed to handle large volumes of data with flexibility and scalability. MongoDB stores data in a document-based structure using JSON-like objects called BSON (Binary JSON). This format makes MongoDB highly adaptable, as it allows for dynamic schemas where the structure of data can change without needing to redefine the database schema. This is especially beneficial for applications that deal with unstructured or semi-structured data, such as user profiles, logs, or product catalogs. One of the key advantages of MongoDB is its ability to scale horizontally using sharding, where data is distributed across multiple servers. This makes MongoDB ideal for handling big data applications and high-traffic websites. Its flexible data model supports embedded documents and arrays, allowing developers to store related data in a single document, which reduces the need for complex joins and improves query performance.

MongoDB is commonly used in web and mobile applications where agility and scalability are critical. It is a great choice for applications like e-commerce platforms, real-time analytics, IoT devices, content management systems, and social networks. With its built-in support for replication and failover using replica sets, MongoDB ensures high availability and reliability. This means that even if one server goes down, data remains accessible from replicas. MongoDB integrates well with modern development stacks, such as Node.js, and provides a range of tools like MongoDB Atlas for cloud-based database management and MongoDB Compass for graphical data visualization. It also supports rich querying, indexing, and aggregation, making it powerful for data manipulation and retrieval.

2.4.6 Features of MONGODB:

1. **Document-Based Data Model**
    - Stores data in flexible, JSON-like BSON format.

   - Supports dynamic schemas, allowing for changes in data structure without redefining the schema.
2. **Scalability and Sharding**
   - Horizontal scaling through sharding, distributing data across multiple servers.
   - Ideal for handling large-scale applications and high-traffic workloads
3. **High Availability with Replication**
   - Built-in replication via replica sets ensures data reliability.
   - Provides failover support, keeping data accessible during server outages.
4. **Powerful Querying and Indexing**
   - Supports rich queries, filtering, and aggregation for complex data retrieval.
   - Offers indexing to improve search performance.
5. **Integration and Tools**
   - Seamlessly integrates with modern development stacks like Node.js.
   - Provides tools like MongoDB Atlas for cloud management and MongoDB Compass for data visualization.

MongoDB's flexibility, scalability, and robust ecosystem make it an excellent choice for handling diverse, evolving datasets in modern applications.

2.5 SUMMARY

The **frontend** refers to the part of a website or application that users interact with directly. It includes everything visible on the screen, such as layouts, buttons, text, and images. Frontend development focuses on creating a smooth and responsive user experience using technologies like **HTML**, **CSS**, and **JavaScript**, along with frameworks like React. The **backend**, on the other hand, is the server-side of an application where data processing, storage, and logic occur. It handles tasks like managing databases, authenticating users, and communicating with the frontend. Backend development relies on languages and frameworks like**Node.js**.

CHAPTER 3

SYSTEM DESIGN

3.1 MODULE DESCRIPTION

The hostel management system is divided into several key modules, each designed to handle specific functionalities efficiently. These modules work together to streamline daily operations and ensure smooth management of tasks.

The project contains the following module:

1. User Management Module

2. Stock/Grocery Management Module

3. Grocery Request Workflow Module

4. Dashboard Module

1. User Management Module

This module handles the registration, login, and role-based access control for all users, including Admins, Wardens, Caretakers. Admins can add or edit users, while other roles can access the system based on their permissions. Secure authentication ensures that only authorized users can log in and perform tasks.

2. Stock/Grocery Management Module

This module allows caretakers to manage grocery stock and submit requests for new items. It tracks stock levels in real-time and ensures accuracy in inventory updates. Wardens can review and approve or reject grocery requests, while Admins oversee the entire process to ensure accountability.

3. Grocery Request Workflow Module

This module facilitates the submission, approval, and tracking of grocery requests. Caretakers submit requests through an online form, which are then processed by Wardens. The status of each request (e.g., Pending, Approved, or Rejected) is updated in real-time, allowing for better communication and faster decision-making.

4. Dashboard Module

Each role has access to a customized dashboard that displays relevant information. For example:

- Admins can view user details, stock levels, and grocery requests across the system.
- Wardens can monitor pending requests and stock reports.
- Caretakers can track stock usage and request statuses.

These modules are designed to integrate seamlessly, ensuring a streamlined and efficient hostel management system.

3.2 DATAFLOW DIAGRAM

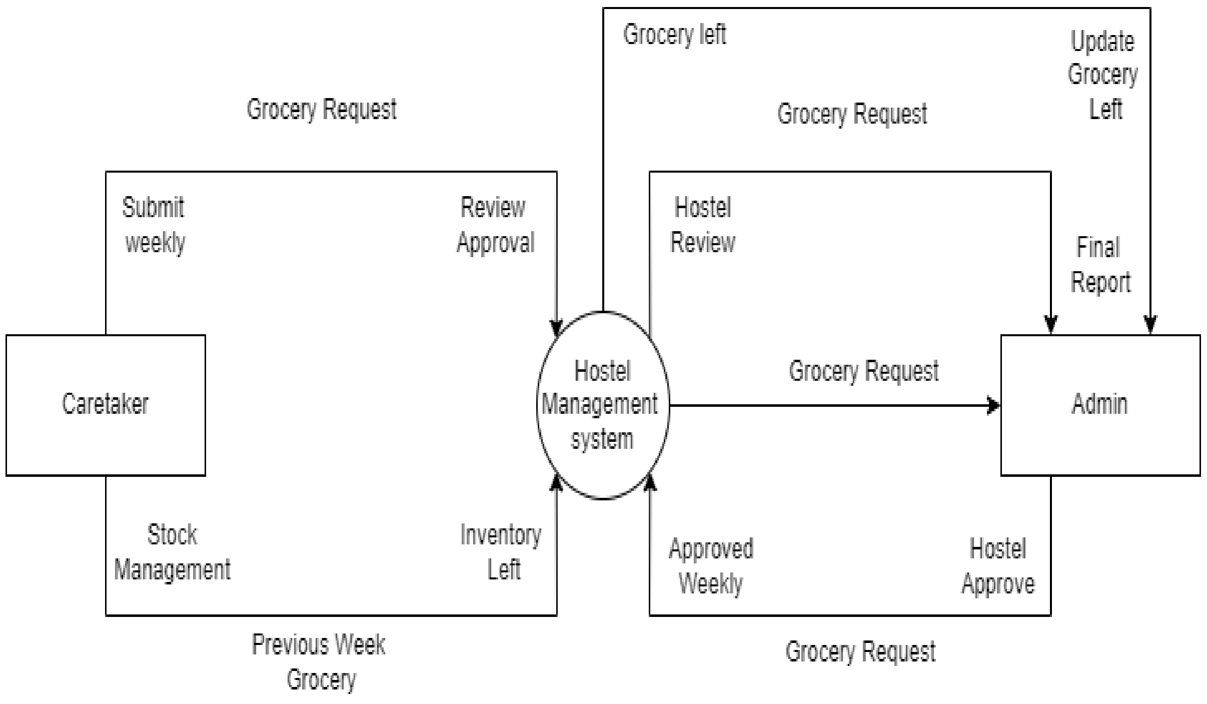


Figure 3.2 DATA FLOW DIAGRAM (LEVEL 0)

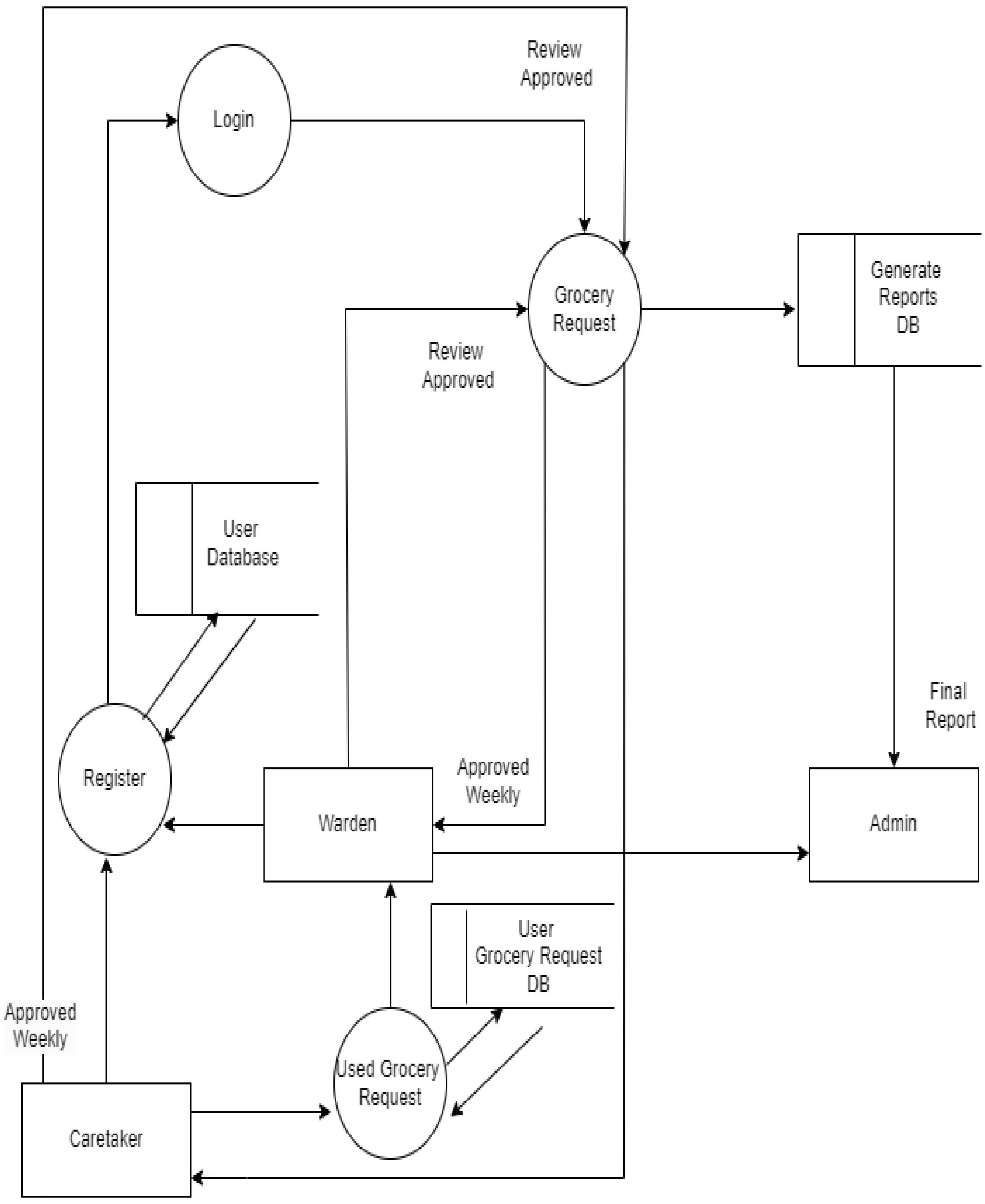


Figure 3.2 DATA FLOW DIAGRAM (LEVEL 1)

3.3 DATABASE DESIGN

The database design for the hostel management system is built using MongoDB, a NoSQL, document-oriented database. MongoDB is well-suited for this project because it handles large amounts of unstructured or semi-structured data efficiently and provides flexibility for adding new fields or features.

The system uses collections to represent key entities, such as users, grocery requests, and stock. Each collection contains documents with fields corresponding to the relevant data. Relationships between collections are established using references (ObjectIDs) or embedded documents, depending on the use case.

Key Collections

1. **Users Collection**
    - Fields: _id, name, email, password, role (e.g., Admin, Warden, Caretaker), createdAt, updatedAt.
    - Purpose: Stores user details and manages role-based access control.
2. **Grocery Requests Collection**
    - Fields: _id, requester_id (reference to Users collection), items (array of item details), status (Pending, Approved, Rejected), createdAt.
    - Purpose: Tracks grocery requests submitted by caretakers.
3. **Stock Collection**
    - Fields: _id, itemName, quantity, unit, updatedAt.
    - Purpose: Maintains inventory details, including available stock levels.

Relationships

- **Users and Grocery Requests**
    - A user (caretaker) can submit multiple grocery requests, creating a one-to-many relationship.
- **Grocery Requests and Stock**
    - Approved requests update the stock levels in the Stock collection.

The database design ensures that data is stored efficiently, is easy to retrieve, and supports the operations of the system seamlessly.

3.4 INPUT DESIGN:

The input design defines how users interact with the system to provide the required information. The goal is to create an interface that is simple, user-friendly, and minimizes errors. In the hostel management system, inputs are collected through structured forms and interactive fields, ensuring accuracy and efficiency.

1. **Login Form**
   - Fields: Email, Password.
   - Purpose: Allows users to securely access the system based on their roles (Admin, Warden, or Caretaker).
   - Validation: Ensures correct email format and minimum password length.
2. **User Registration Form (Admin)**
   - Fields: Name, Email, Role, Password.
   - Purpose: Enables the admin to register new users (Caretakers or Wardens).
   - Validation: Prevents duplicate emails and ensures proper role assignment.
3. **Grocery Request Form (Caretaker)**
   - Fields: Item Name, Quantity, Unit.
   - Purpose: Allows caretakers to submit detailed grocery requests for approval.
   - Validation: Ensures the quantity is a positive number and the unit is valid (e.g., kg, liters).
4. **Stock Management Form (Caretaker)**
   - Fields: Item Name, Quantity, Unit.
   - Purpose: Enables caretakers to update the current stock levels in the system.
   - Validation: Prevents negative quantities and ensures that all fields are filled.

Error Prevention and Validation

- All forms include input validation to avoid errors. For example:
  - Empty fields trigger a warning message.
  - Invalid inputs (e.g., negative numbers or invalid email formats) are rejected.
- Dropdown menus or predefined options are used for fields like roles and units to limit user mistakes.

3.5 OUTPUT DESIGN

The output design focuses on how the system presents processed data and results to the users. A well-designed output ensures that information is clear, accessible, and actionable for all roles, including Admins, Wardens, and Caretakers. The hostel management system uses dashboards, reports, and dynamic updates to display outputs effectively.

1. **User Dashboards**
   - Each user role (Admin, Warden, Caretaker) has a customized dashboard.
   - **Admins** see system-wide data, such as the total number of users, pending grocery requests, and stock levels.
   - **Wardens** view grocery requests awaiting approval and stock reports.
   - **Caretakers** track stock levels and the statuses of their submitted requests.
   - Dashboards update in real-time, ensuring users always see the latest data.
2. **Grocery Request Status**
   - Caretakers receive updates on their submitted requests (e.g., Pending, Approved, or Rejected).
   - Statuses are displayed clearly in a table format, along with request details like item names, quantities, and submission dates.
3. **Stock Reports**
   - Displays current stock levels, including item names, available quantities, and units.
   - Helps caretakers plan grocery requests and allows wardens to monitor inventory effectively.
4. **Error Messages and Alerts**
   - Users are notified about errors, such as invalid inputs or failed submissions.
   - Alerts are also displayed for low stock levels or pending approvals, helping users take timely action.
5. **Exportable Reports**
   - Admins can generate and download detailed reports on user data, stock levels, and grocery request histories.
   - These reports are useful for audits, decision-making, and record-keeping.

Design Approach:

The output is presented using clear tables, charts, and notifications to ensure ease of understanding. By focusing on simplicity and relevance, the output design enables users to make informed decisions quickly and efficiently.

3.6 SUMMARY

The hostel management system is designed with a MongoDB database for flexibility and efficiency, using collections such as Users (storing user details and roles), Grocery Requests (tracking caretaker submissions), and Stock (managing inventory levels). Relationships are established using references and embedded documents, ensuring data integrity and easy retrieval. Input design focuses on user-friendly forms for logging in, registering users, submitting grocery requests, and updating stock. Validation mechanisms prevent errors like empty fields, invalid formats, or negative values, while dropdown menus and predefined options enhance accuracy. Output design prioritizes clear and actionable information through customized dashboards for Admins, Wardens, and Caretakers, providing real-time updates on grocery request statuses, stock levels, and other key data. Exportable reports, error messages, and alerts ensure timely actions and informed decision-making. This design ensures seamless, efficient, and user-friendly operations for all roles involved.

CHAPTER 4

IMPLEMENTATION

4.1 CODE DESCRIPTION

Code description can be used to summarize code or to explain the programmer’s intent. Good comments don’t repeat the code or explain it. They clarify its intent. Comments are sometimes processed in various ways to generate documentation external to the source code itself by document generator or used for integration with systems and other kinds of external programming tools. . I have chosen Node as front end, back end and mongoDB for database connectivity.

4.2 STANDARDIZATION OF THE CODING

Coding standards define a programming style. A coding standard does not usually concern itself with wrong or right in a more abstract sense. It is simply a set of rules and guidelines for the formatting of source code. The other common type of coding standard is the one used in or between development teams. Professional code performs a job in such a way that is easy to maintain and debug. All the coding standards are followed while creating this project. Coding standards become easier, the earlier you start. It is better to do a neat job than cleaning up after all is done. Every coder will have a unique pattern than head here to. Such a style might include the conventions he uses to name variables and functions and how he comments his work. When the said pattern and style is standardized, it pays off the effort well in the long.

4.3 ERROR HANDLING

Exception handling is a process or method used for handling the abnormal statements in the code and executing them. It also enables to handle the flow control of code/program. For handling the code, various handlers are used that process the exception and execute the code. Mainly if-else block isused to handle errors using condition checking, If else errors as a conditional statement.

In many cases there are many corner cases which must be checking during an execution but “if-else” can only handle the defined conditions. In if-else, conditions are manually generated based on the task. An error is a serious problem than an application doesn’t usually get pass without incident. Errors cause an application to crash, and ideally send an error message offering some suggestion to resolve the problem and return to a normal operating state, There’s no way to deal with errors “live” or in production the only solution is to detect them via error monitoring and bug tracking and dispatch a developer or two to sort out the code.

Exceptions, on the other hand are exceptional conditions an application should reasonably be accepted to handle. Programming language allow developers to include try…catch statements to handle exceptions and apply a sequence of logic to deal with the situation instead of crashing. And when an application encounters an exception that there’s no workaround for, that’s called an unhandled exception, which is the same thing as an error.

4.4 SUMMARY

In Chapter 4, the implementation of the project is discussed, emphasizing code description and the use of React for front-end and back-end with MongoDB for database connectivity. The importance of coding standardization is highlighted, emphasizing the creation of professional, maintainable, and easily debug able code. The chapter also covers error handling through exception handling, distinguishing between errors that cause crashes and exceptions that can be handled gracefully using try...catch statements. Overall, the chapter emphasizes clarity of code intent, adherence to coding standards, and effective error management.

CHAPTER 5

TESTING AND RESULTS

5.1 TESTING

Software testing serves as the final assessment of specifications, designs, and coding and is a crucial component of software quality assurance. The system is tested throughout the testing phase utilizing diverse test data. The preparation of test data is essential to the system testing process.Thesystemunderstudyistestedafterthetestdatapreparation.Oncethesource code is complete, relevant data structures should be documented. The finished project must go through testing and validation, when errors are explicitly targeted and attempted to be found.

The project developer is always in charge of testing each of the program's separate units, or modules. Developers frequently also perform integration testing, which is the testing phase that results in the creation of the complete program structure.

This project has under gone the following testing procedures to ensure its correctness

- Unit testing
- Integration Testing
- Validation Testing

5.1.1 Unit Testing

Unit testing is performed to test individual modules of the system in isolation. Each feature, such as login, stock updates, and grocery requests, is tested separately to ensure it functions correctly. This helps identify issues in the code logic and individual components at an early stage.

Example Test Cases

- **Test Case 1**
  - **Module**: User Login
  - **Input**: Valid username and password.
  - **Output**: User is successfully logged in.
  - **Event**: Submit login form.
  - **Result**: The system authenticates the user and redirects them to their

- **Test Case 2**
    - **Module**: User Login
    - **Input**: Invalid password for a valid username.
    - **Output**: Login failed message is displayed.
    - **Event**: Submit login form.
    - **Result**: The system rejects the incorrect credentials and does not allow access.

5.1.2 Integration Testing

Integration testing ensures that the individual modules of the system work together as a single, cohesive unit. It verifies the interactions between the frontend and backend, as well as between different components, such as user management and stock management.

Example Test Cases

- **Test Case 1**
    - **Module**: Grocery Request Submission
    - **Input**: Item name, quantity, and unit.
    - **Output**: Grocery request successfully submitted and added to the database.
    - **Event**: Submit request form.
    - **Result**: The system processes the input and displays a confirmation message.
- **Test Case 2**
    - **Module**: Stock Updates
    - **Input**: Valid stock details (e.g., item name and quantity).
    - **Output**: Stock levels are updated in the database.
    - **Event**: Submit stock update form.
    - **Result:** The system updates the stock and displays the updated stock list.31

5.1.3 Validation Testing

Validation testing ensures that the system meets the user's requirements and performs as intended. It checks the system's ability to handle valid and invalid inputs gracefully, ensuring user-friendly feedback in case of errors.

Example Test Cases

- **Test Case 1**:
    - **Module**: User Registration
    - **Input**: Valid username, email, role, and password.
    - **Output**: User registered successfully.
    - **Event**: Submit registration form.
    - **Result**: The system creates a new user and redirects the admin to the user list.
- **Test Case 2**
    - **Module**: Stock Update
    - **Input**: Negative quantity value.
    - **Output**: Error message is displayed, and the update is rejected.
    - **Event**: Submit stock update form.
    - **Result**: The system prevents invalid inputs and ensures data integrity.

5.2 SUMMARY

Chapter 5 The testing phase confirmed that the system functions as intended. Issues identified during unit and integration testing were resolved, ensuring that the modules interact seamlessly. Validation testing ensured that the system handles errors gracefully and provides clear feedback to users. The final tests verified that the system meets the requirements and is ready for deployment.

CHAPTER 6

6.1 CONCLUSION

In conclusion, effective management of hostel operations is essential for ensuring smooth and transparent administration for wardens, and caretakers. A well-functioning hostel management system plays a significant role in streamlining operations such as user management, stock tracking, and grocery request approvals. To achieve this, implementing an automated and centralized solution is crucial for reducing errors, improving efficiency, and maintaining accountability across all roles.

By incorporating modern technologies such as the MERN stack, this system enhances daily hostel operations through real-time updates, structured workflows, and secure role-based access. It minimizes manual effort, eliminates redundant tasks, and ensures that critical information is readily accessible. The system not only simplifies routine processes but also enables faster decision-making, promoting better coordination among stakeholders.

Overall, a robust hostel management system improves the quality of services provided, reduces administrative overhead, and fosters a more efficient and transparent environment. A commitment to technological integration and continuous improvement will ensure that the system remains relevant and effective in addressing the evolving needs of hostel management.

6.2 FUTURE ENHANCEMENT

In envisioning future enhancements for the hostel management system, a comprehensive approach that leverages modern technology and innovative solutions is essential. Firstly, integrating mobile applications can provide users with seamless access to their dashboards and allow them to perform tasks such as submitting grocery requests and tracking updates from anywhere. The introduction of push notifications can further enhance user experience by delivering real-time alerts about important actions, such as grocery request approvals or low stock warnings.

Additionally, implementing advanced reporting and data analytics can help administrators make informed decisions by identifying patterns in stock usage and predicting future needs. The use of IoT-enabled devices, such as smart sensors, can enable real-time tracking of stock levels and provide automated updates, reducing the need for manual intervention. Furthermore, incorporating QR code or barcode scanning can streamline inventory management, making the system more efficient and user-friendly.

The system can also be extended to include features such as a meal management module, enabling wardens and caretakers to plan and track daily meal schedules effectively. Feedback mechanisms can be introduced to allow students to share their opinions on hostel facilities, fostering a culture of continuous improvement. Migrating the system to a cloud-based infrastructure would enhance scalability and accessibility, ensuring smooth operation even as the number of users increases.

These future enhancements aim to transform the hostel management system into a more robust, intelligent, and adaptable solution that meets the evolving needs of all stakeholders while improving overall efficiency and transparency.

6.3 SUMMARY

In conclusion, the hostel management system requires a holistic and flexible approach to ensure efficient and transparent operations. By adopting modern technologies like real-time updates, role-based access, and a centralized database, the system effectively addresses the challenges of manual processes and data inconsistencies. To further enhance the system, future improvements should focus on introducing mobile accessibility, advanced reporting, IoT integration, and feedback mechanisms.

These enhancements will not only improve efficiency but also foster better coordination among stakeholders and create a more user-friendly environment. By continuously innovating and adapting to new requirements, the hostel management system can evolve into a scalable, reliable, and comprehensive solution that meets the dynamic needs of students, caretakers, and administrators.

APPENDICES

A. SAMPLE CODING

APP.JS

```
import React, { useState, useEffect } from 'react';
import { BrowserRouter as Router, Route, Routes } from 'react-router-dom';
import Login from './components/Login';
import Register from './components/Register';
import GroceryRequestList from './components/GroceryRequestList';
import SubmitGroceryRequest from './components/SubmitGroceryRequest';
import AddStock from './components/AddStock';
import UpdateStock from './components/UpdateStock';
import StockList from './components/StockList';
import AdminHeader from './components/AdminHeader';
import CaretakerHeader from './components/CaretakerHeader';
import WardenHeader from './components/WardenHeader';
import AdminFooter from './components/AdminFooter';
import CaretakerFooter from './components/CaretakerFooter';
import WardenFooter from './components/WardenFooter';
import UpdatePassword from './components/UpdatePassword';
import ResetPassword from './components/ResetPassword';
import HomePage from './components/HomePage';
import AdminDashboard from './components/AdminDashboard';
import CaretakerDashboard from './components/CaretakerDashboard';
import WardenDashboard from './components/WardenDashboard';
import NotFound from './components/404';
import UsedApproval from './components/UsedApproval';
import UsedGroceryRequestFormfrom './components/UsedGroceryRequestForm';
const App = () => {
  const [token, setToken] = useState(localStorage.getItem('token'));
  const [userRole, setUserRole] = useState(null);

useEffect(() => {
    if (token) {
      try {
        const decodedToken = JSON.parse(atob(token.split('.')[1]));
setUserRole(decodedToken.role);
      } catch (error) {
console.error('Error decoding token:', error);
localStorage.removeItem('token');
setUserRole(null);
      }
    }
  }, [token]);

  const handleLogout = () => {
localStorage.removeItem('token');
setToken(null);
setUserRole(null);
  };

  const renderHeaderFooter = () => {
    if (!token) return { header: null };
    switch (userRole) {
      case 'Admin':
        return { header: <AdminHeaderhandleLogout={handleLogout} />, footer:
<AdminFooter /> };
      case 'Caretaker':
        return { header: <CaretakerHeaderhandleLogout={handleLogout} />, footer:
<CaretakerFooter /> };
      case 'Warden':
        return { header: <WardenHeaderhandleLogout={handleLogout} />, footer:
<WardenFooter /> };
      default:
        return { header: null, footer: null };
    }
  };

  const { header } = renderHeaderFooter();

  return (
<Router>
     {header}
<Routes>
       {/* Public Routes */}
<Route path="/" element={<HomePage />} />
<Route path="/login" element={<Login setToken={setToken} />} />
<Route path="/reset-password" element={<ResetPassword />} />
<Route path="/update-password" element={<UpdatePassword />} />

       {token && (
<>
           {/* Admin Routes */}
           {userRole === 'Admin' && (
<>
<Route path="/admin-dashboard" element={<AdminDashboard />} />
<Route path="/register" element={<Register isAdmin={true} />} />
<Route path="/grocery-request-list" element={<GroceryRequestList />} />
<Route path="/stock-list" element={<StockList />} />
<Route path="/used-approval" element={<UsedApproval role="Admin" />}
<Route path="/feedback" element={<ViewFeedbackPage />} />
</>
           )}

           {/* Caretaker Routes */}
           {userRole === 'Caretaker' && (
<>
<Route path="/caretaker-dashboard" element={<CaretakerDashboard />} />
<Route path="/grocery-request-list" element={<GroceryRequestList />} />
<Route path="/submit-grocery-request" element={<SubmitGroceryRequest />} />
<Route path="/add-stock" element={<AddStock />} />
<Route path="/update-stock" element={<UpdateStock />} />
<Route path="/stock-list" element={<StockList />} />
<Route path="/used-approval" element={<UsedApproval role="Caretaker" />} />
<Route path="/used-grocery-request-form" element={<UsedGroceryRequestForm />} />
<Route path="/feedback" element={<ViewFeedbackPage />} />
</>
            )}

            {/* Warden Routes */}
            {userRole === 'Warden' && (
<>
<Route path="/warden-dashboard" element={<WardenDashboard />} />
<Route path="/update-meals" element={<UpdateMealsPage />} />
<Route path="/stock-list" element={<StockList />} />
<Route path="/used-approval" element={<UsedApproval role="Warden" />} />
<Route path="/grocery-request-list" element={<GroceryRequestList />} />
<Route path="/feedback" element={<ViewFeedbackPage />} />
</>
            )}
</>
        )}

<Route path="*" element={<NotFound />} />
</Routes>
</Router>
  );
};

export default App;
```

REGISTER.JS

```
import React, { useState } from 'react';

import axios from 'axios';

import './Register.css';

import { useNavigate } from 'react-router-dom';


const Register = ({ isAdmin }) => {

  const [form, setForm] = useState({

    username: '',

    email: '',

    role: isAdmin ? '' : 'Student',

  });

  const [message, setMessage] = useState('');

  const [error, setError] = useState('');

  const [loading, setLoading] = useState(false);

  const navigate = useNavigate();


  // Handle form input changes

  consthandleInputChange = (e) => {

    const{ name, value } = e.target;

    setForm({ ...form, [name]: value });

  };


  // Form submission

  consthandleRegister = async (e) => {

    e.preventDefault();
```

```
setMessage('');
setError('');


// Basic validation
if (!form.username.trim()) {
  setError('Username is required.');
  return;
}
if (!form.email.trim() || !/^[\w-.]+@[\w-]+\.+[\w-]{2,4}$/.test(form.email)) {
  setError('Valid email is required.');
  return;
}
if (isAdmin&& !form.role) {
  setError('Please select a role.');
  return;
}


try {
  setLoading(true);
  const token = localStorage.getItem('token');
  const response = await axios.post(
    'http://localhost:3001/api/users/register',
    form,
    {
      headers: {
        Authorization: `Bearer ${token}`,
      },
      }
    );

    setMessage('User registered successfully!');
    setForm({ username: '', email: '', role: isAdmin ? '' : 'Student' }); // Reset form

    // Redirect if needed (optional)
    if (isAdmin) {
      setTimeout(() => navigate('/admin/dashboard'), 2000);
    }
  } catch (error) {
    console.error('Registration Error:', error);
    setError(error.response?.data?.message || 'Error registering user.');
  } finally {
    setLoading(false);
  }
};

return (
  <div className="register-container">
    <h2 className="register-title">Create an Account</h2>
    <form className="register-form" onSubmit={handleRegister}>
      <div className="input-group">
        <input
          type="text"
          name="username"
          placeholder="Username"
    value={form.username}
    onChange={handleInputChange}
    className="input-field"
    required
  />
</div>
<div className="input-group">
  <input
    type="email"
    name="email"
    placeholder="Email"
    value={form.email}
    onChange={handleInputChange}
    className="input-field"
    required
  />
</div>
{isAdmin&& (
  <div className="input-group">
    <select
      name="role"
      value={form.role}
      onChange={handleInputChange}
      className="input-field"
      required
    >
      <option value="">Select Role</option>
```

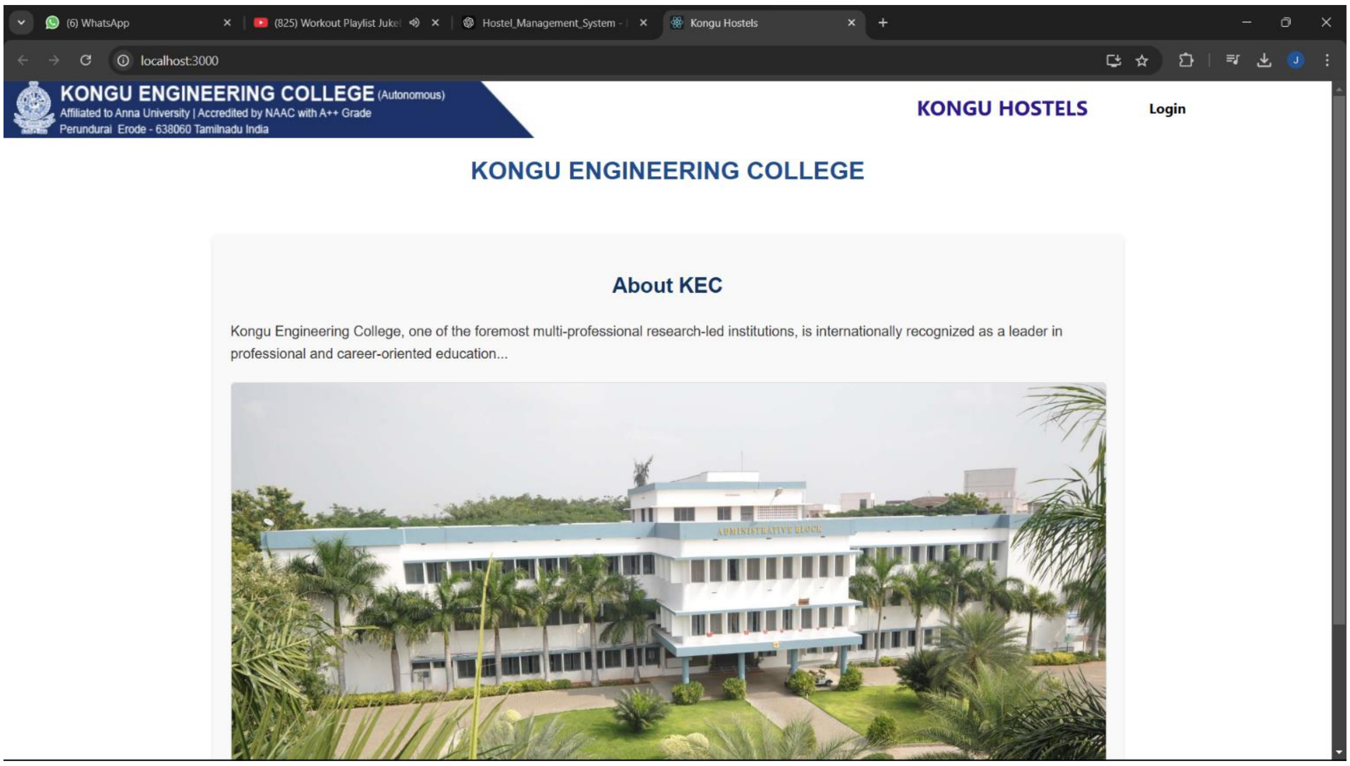
```
            <option value="Caretaker">Caretaker</option>

            <option value="Warden">Warden</option>

            <option value="Admin">Admin</option>

           </select>

          </div>

        )}

        <button type="submit" className="submit-btn" disabled={loading}>

          {loading ? 'Registering...' : 'Register'}

        </button>

      </form>

      {message &&<p className="register-message success-message">{message}</p>}

      {error &&<p className="register-message error-message">{error}</p>}

    </div>

  );

};


export default Register;
```

LOGIN.JS

```
import React, { useState } from 'react';

import axios from 'axios';

import './Login.css';

import { Link, useNavigate } from 'react-router-dom';


const Login = ({ setToken }) => {

  const [email, setEmail] = useState('');

  const [password, setPassword] = useState('');

  const [error, setError] = useState('');
const navigate = useNavigate();


consthandleLogin = async (e) => {
  e.preventDefault();
  setError('');


  try {
    const response = await axios.post('http://localhost:3001/api/users/login', { email, password });
    const token = response.data.token;


    localStorage.setItem('token', token);
    setToken(token);


    constdecodedToken = JSON.parse(atob(token.split('.')[1]));
    constuserRole = decodedToken.role;


    alert('Login successful!');




    if (userRole === 'Admin') navigate('/admin-dashboard');
    else if (userRole === 'Caretaker') navigate('/caretaker-dashboard');
    else if (userRole === 'Warden') navigate('/warden-dashboard');
    else navigate('/404');
  } catch (err) {
    console.error(err);
    setError(err.response?.data?.message || 'Invalid credentials.');
  }
```

```
};

return (

  <div className="login-container">

    <form className="login-form" onSubmit={handleLogin}>

      <h2>Login</h2>

      {error &&<p className="error-message">{error}</p>}

      <input

        type="email"

        placeholder="Email"

        value={email}

        onChange={(e) =>setEmail(e.target.value)}

        required

      />

      <input

        type="password"

        placeholder="Password"

        value={password}

        onChange={(e) =>setPassword(e.target.value)}

        required

      />

      <button type="submit">Login</button>

      <div>

        <br></br>

        Forgot Password? <Link to="/reset-password">Click Here</Link>

      </div>

    </form>

  </div>
  );
};
export default Login;

Server.js:const express = require('express');
const mongoose = require('mongoose');
constbodyParser = require('body-parser');
constcors = require('cors');
const path = require('path');
constdotenv = require('dotenv');
constusedGroceryRequests = require('./routes/usedGroceryRequests');

// Import route files
constuserRoutes = require('./routes/users');
constgroceryRequestsRouter = require('./routes/groceryRequests');
conststockRoutes = require('./routes/stocks');
constmealsRoutes = require('./routes/meals');

// Load environment variables
dotenv.config({ path: path.join(__dirname, 'config/config.env') });

const app = express();

// Middleware
app.use(bodyParser.json());
app.use(cors());
app.use(express.json());
app.use(express.static(path.join(__dirname, 'frontend/build')));

// Connect to MongoDB
mongoose
  .connect(process.env.DB_LOCAL_URI)
  .then(() => console.log('MongoDB connected successfully'))
  .catch((err) =>console.error('Error connecting to MongoDB:', err));

// API Routes
app.use('/api/users', userRoutes); // User routes
app.use('/api/grocery-requests', groceryRequestsRouter); // Grocery requests routes
app.use('/api/stocks', stockRoutes); // Stock routes
app.use('/api/meals', mealsRoutes); // Meals routes
app.use('/api/used-grocery-requests', usedGroceryRequests);

// Catch-all for serving frontend (React)
app.get('*', (req, res) => {
  res.sendFile(path.resolve(__dirname, 'frontend', 'build', 'index.html'));
});

// Start the server
const PORT =  3001;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

B. SCREENSHOTS


(6) WhatsApp
(825) Workout Playlist Juke
Hostel_Management_System
Kongu Hostels
localhost:3000
KONGU ENGINEERING COLLEGE (Autonomous)
Affiliated to Anna University | Accredited by NAAC with A++ Grade
Perundurai Erode - 638060 Tamilnadu India
KONGU HOSTELS
Login

KONGU ENGINEERING COLLEGE

About KEC

Kongu Engineering College, one of the foremost multi-professional research-led institutions, is internationally recognized as a leader in professional and career-oriented education...

Fig. 3.4 - Home Page

Homepage (Landing Page)

The homepage serves as the initial landing page for all users. It provides a simple and welcoming interface, offering navigation options to the login page. This page introduces the system and guides users to log in and access their respective dashboards.

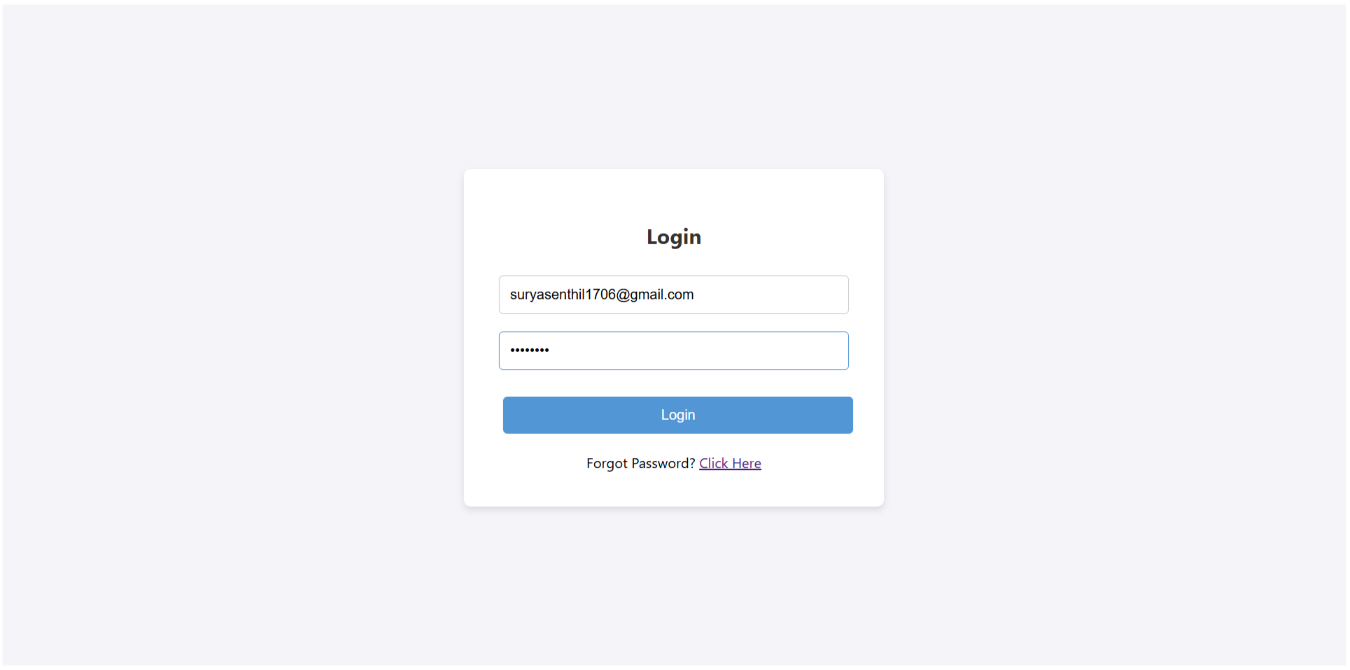
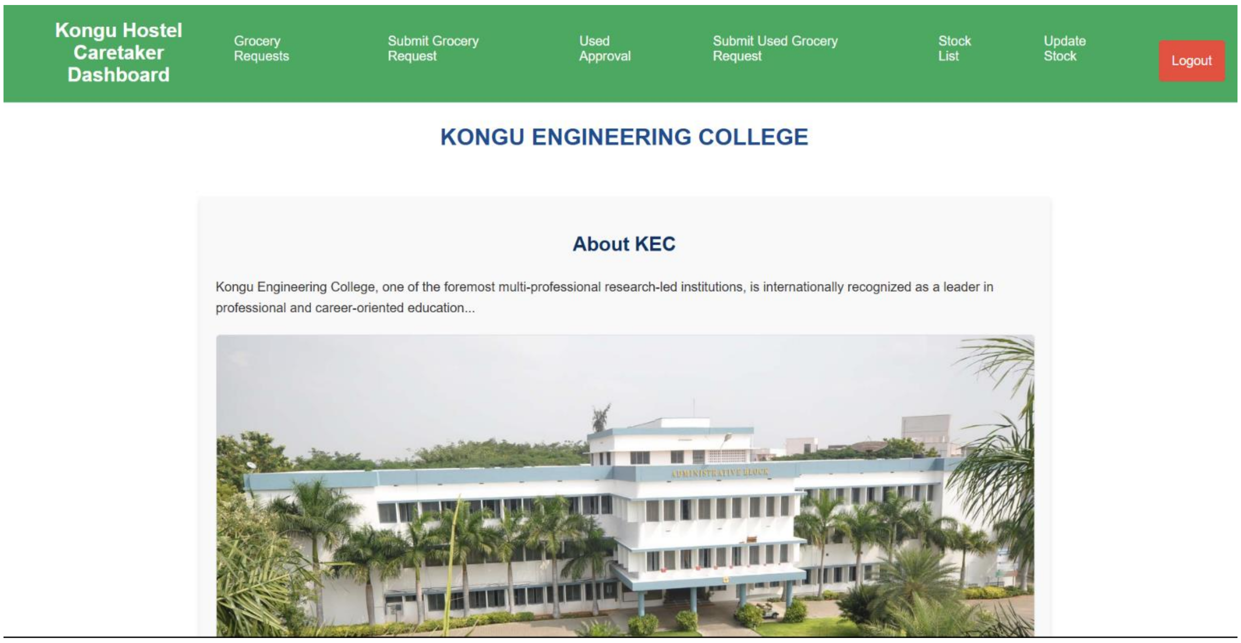


Fig. 3.5 - Login Page

Login Page

The login page is the entry point for all users, where they can securely log in with their credentials. It redirects users to their role-specific dashboards (Admin, Warden, or Caretaker) after successful authentication, ensuring secure and personalized access.



KONGU ENGINEERING COLLEGE

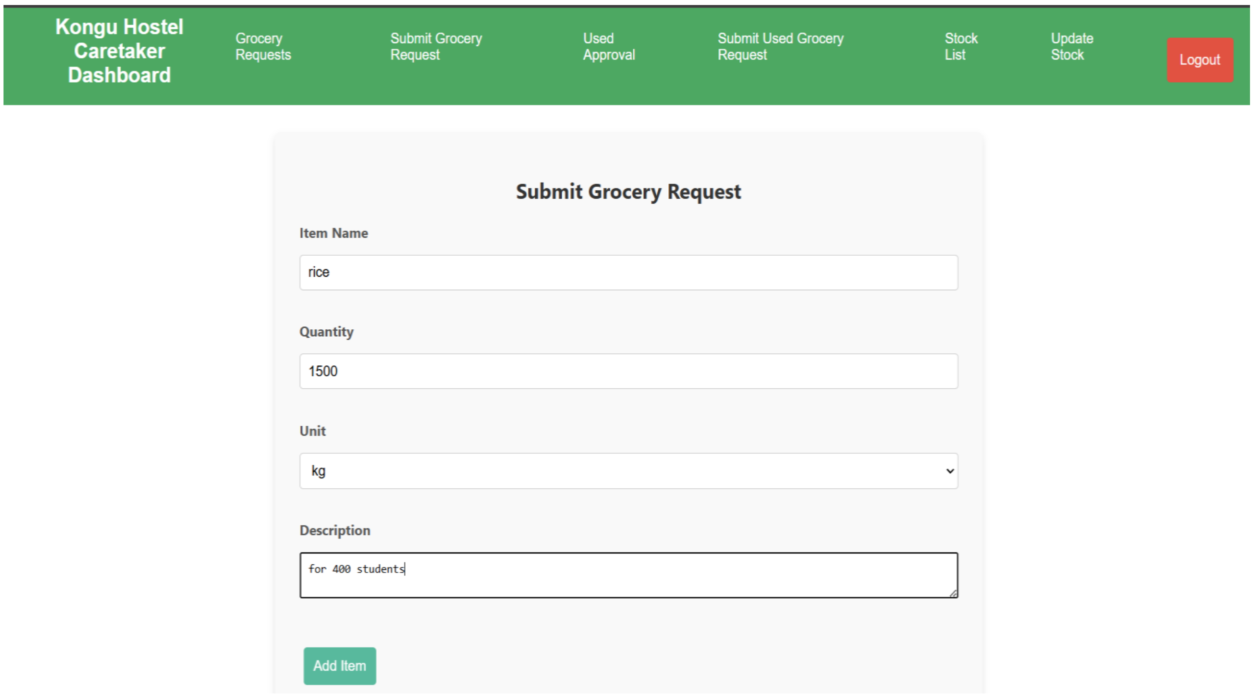
About KEC

Kongu Engineering College, one of the foremost multi-professional research-led institutions, is internationally recognized as a leader in professional and career-oriented education...

Fig. 3.6 - Home Page

Caretaker Dashboard

This dashboard is tailored for caretakers, allowing them to manage stock and grocery requests effectively. It provides a clear overview of tasks, such as submitting new requests, tracking request statuses, and updating inventory.



Submit Grocery Request

Item Name	rice
Quantity	1500
Unit	kg
Description	for 400 students

Caretakers can specify the items required, along with their quantities and measurement units (e.g., kg, liters). The form ensures that all necessary details are provided to avoid miscommunication or errors.

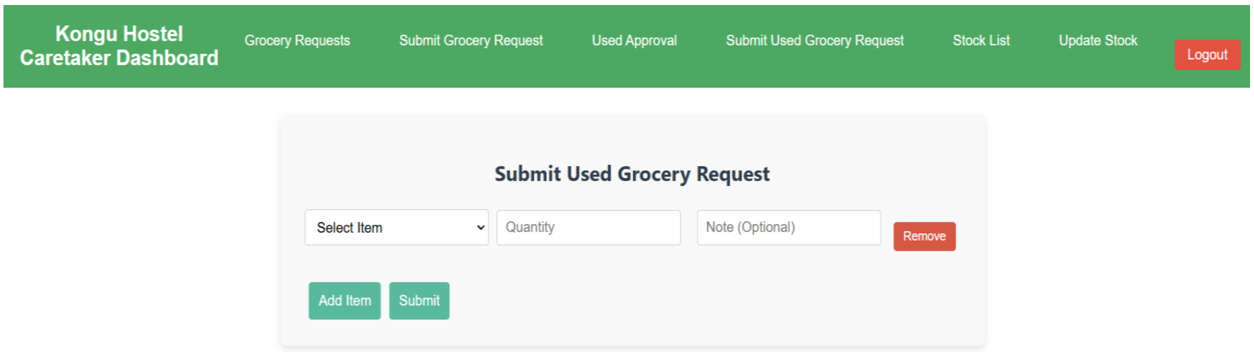
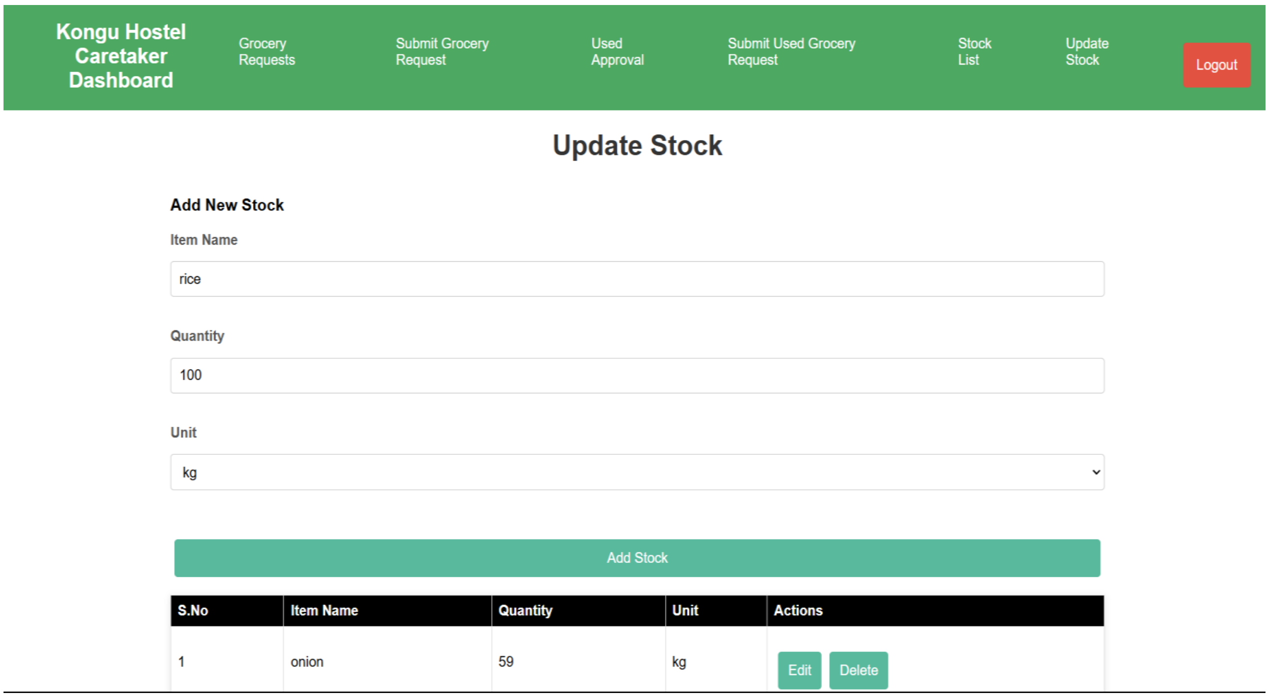


Fig - 3.7 Submit Used Grocery Request Page

Used Grocery Request Page

This page allows caretakers to view and manage submitted grocery requests. Caretakers can see details of previously submitted requests, including their statuses (pending, approved, or rejected), ensuring transparency and efficient tracking.



Update Stock

Add New Stock

Item Name: rice

Quantity: 100

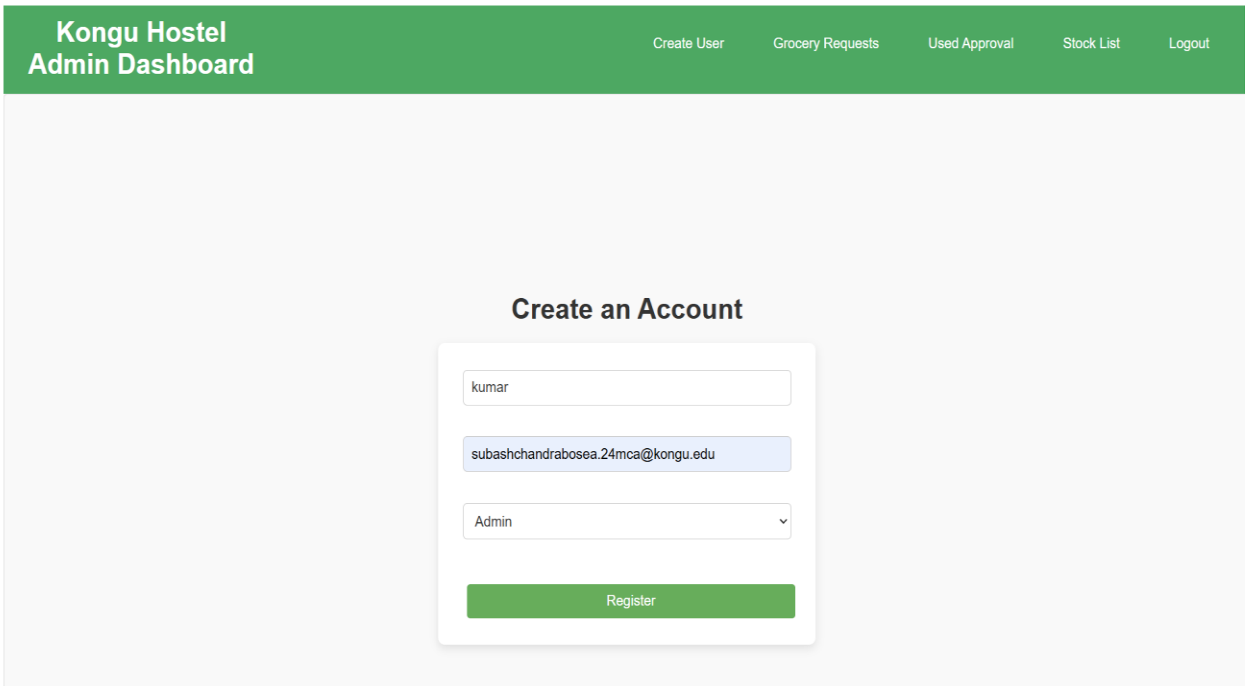
Unit: kg

S.No	Item Name	Quantity	Unit	Actions
1	onion	59	kg	Edit Delete

Fig. 3.8 – Stock Update Page

Stock Update Page

The stock update page is where caretakers can modify inventory levels. They can update quantities, add new items, or remove outdated ones, ensuring that the stock records remain accurate and up-to-date.



kumar

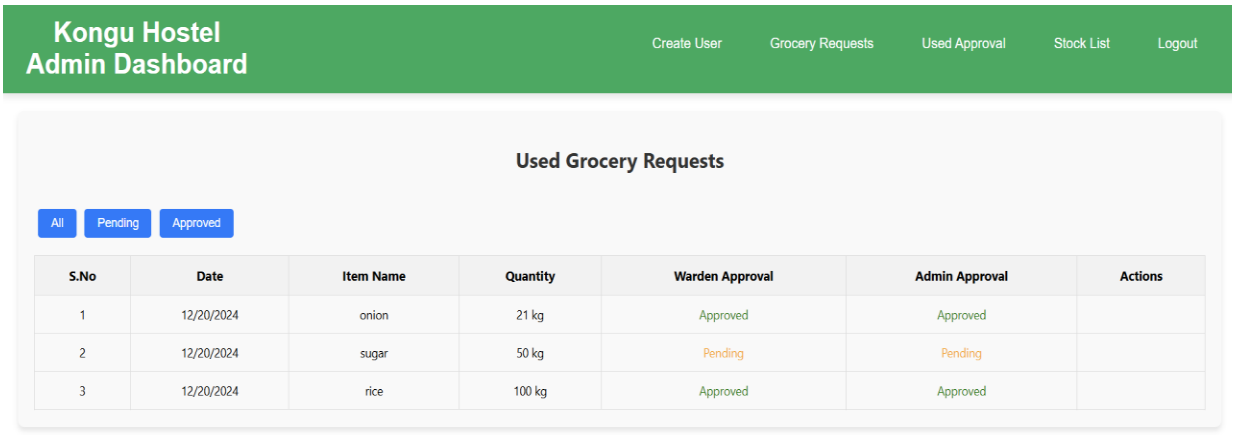
subashchandrabosea.24mca@kongu.edu

Admin

Fig. 3.9 – User Register Page

User Register Page

The admin dashboard is the control center for administrators. It includes features for user management, such as registering or updating user details, and approving grocery requests. It also provides insights into the overall system status and operations.



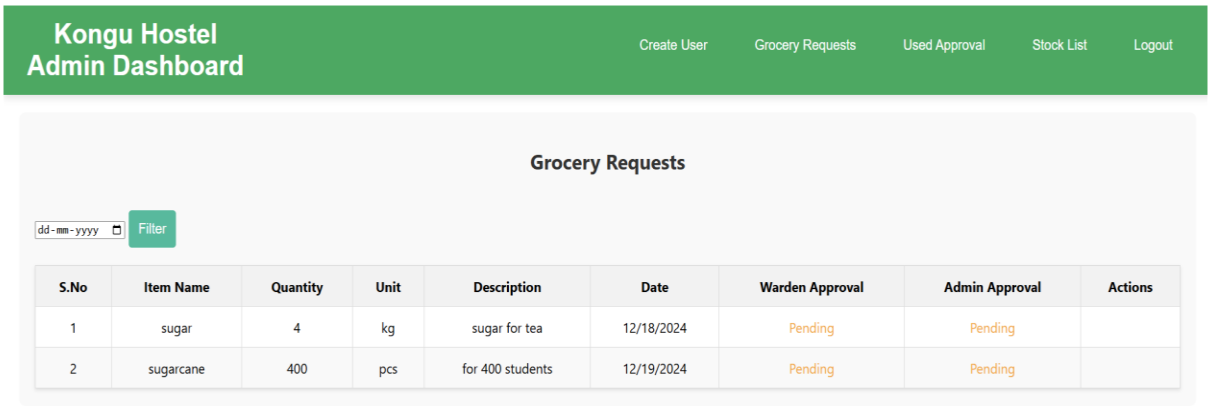
Used Grocery Requests

S.No	Date	Item Name	Quantity	Warden Approval	Admin Approval	Actions
1	12/20/2024	onion	21 kg	Approved	Approved	
2	12/20/2024	sugar	50 kg	Pending	Pending	
3	12/20/2024	rice	100 kg	Approved	Approved	

Fig. 3.10 – Used Grocery Request Page

Used Grocery Request Page

This page allows caretakers to view and manage submitted grocery requests. Caretakers can see details of previously submitted requests, including their statuses (pending, approved, or rejected), ensuring transparency and efficient tracking.



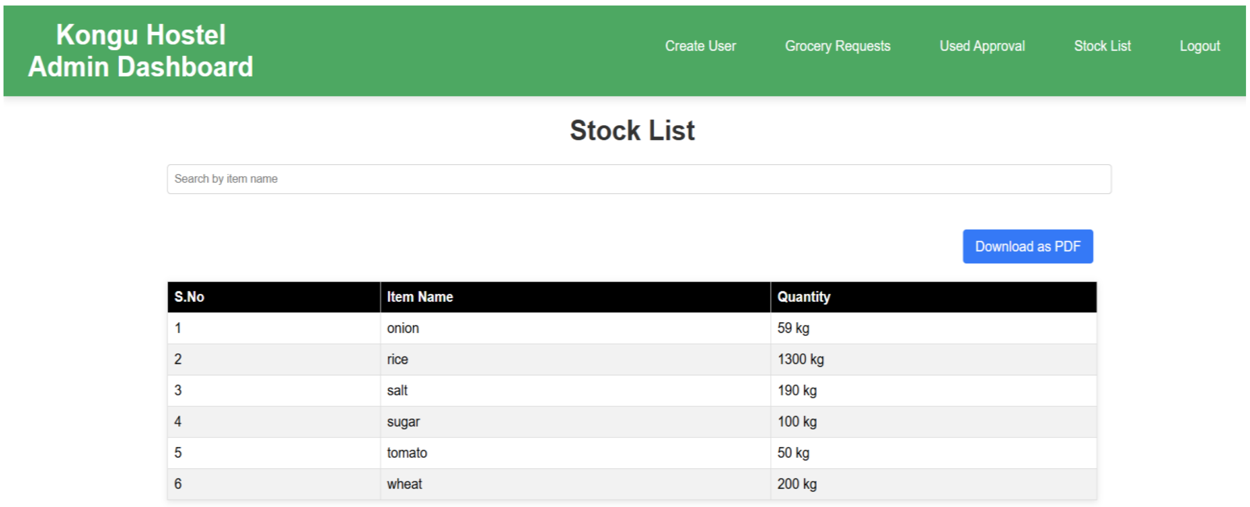
Grocery Requests

S.No	Item Name	Quantity	Unit	Description	Date	Warden Approval	Admin Approval	Actions
1	sugar	4	kg	sugar for tea	12/18/2024	Pending	Pending	
2	sugarcane	400	pcs	for 400 students	12/19/2024	Pending	Pending	

Fig. 3.11 - Future Grocery Approval Page

Future Grocery Approval Page

This page allows wardens and admins to view a list of all pending requests, along with detailed information such as item names, quantities, and units. Wardens can approve or reject requests based on the hostel's requirements, after which the requests are forwarded to the admin for final approval.



Stock List

S.No	Item Name	Quantity
1	onion	59 kg
2	rice	1300 kg
3	salt	190 kg
4	sugar	100 kg
5	tomato	50 kg
6	wheat	200 kg

Fig. 3.12 – Stock List Page

Stock List Page

The stock list page provides an overview of all available grocery stock. Users, particularly caretakers, can view item names, quantities, and units in a well-organized format, ensuring accurate inventory tracking and management.

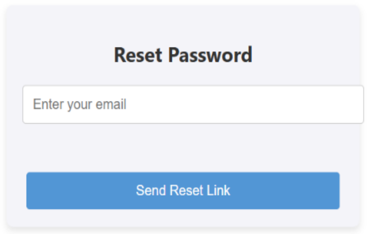


Fig. 3.13 – Password Reset Page

Password Reset Page

The password reset page is designed for users who have forgotten their passwords. It facilitates secure password recovery by sending a reset link to the user’s registered email, ensuring they can regain access to their account.

REFERENCES

1. React.js - A JavaScript library for building user interfaces:
   https://reactjs.org/
2. Node.js - JavaScript runtime built on Chrome's V8 engine:
   https://nodejs.org/
3. MongoDB - NoSQL database for modern applications:
   https://www.mongodb.com/
4. Express.js - Fast, unopinionated, minimalist web framework for Node.js:
   https://expressjs.com/