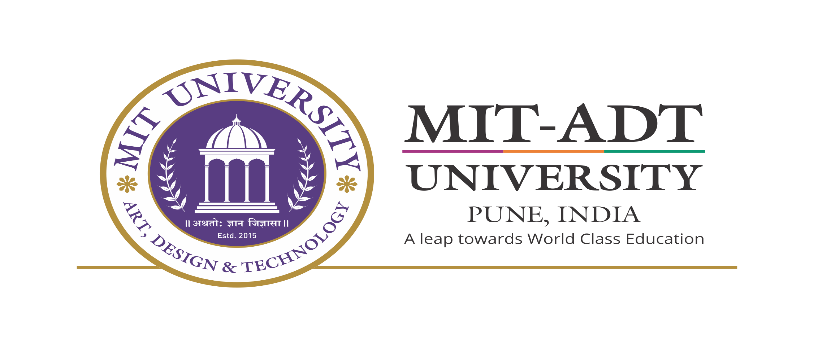
**A**

**DISSERTATION PROJECT**

**ON**

**“ROOMIEHUB ”**

****

**IN THE REQUIREMENT OF AWARD DEGREE OF**

**MASTER OF COMPUTER APPLICATION (MCA)**

**UNDER THE GUIDANCE OF**

**Prof. Harshit Kumar**

**SUBMITTED BY**

**LIKITA KALLURI ADT23MGTM0824**

**RASHI KUDALE ADT23MGTM0835**

**BATCH- 2023-25**

**SUBMITTED TO**

****

**MIT COLLEGE OF MANAGEMENT, PUNE**

**2024-2025**

****

**MIT ADT University, Pune**

**MIT College of Management, Pune**

**Program : Master of Computer Application**

**CERTIFICATE**

This is to certify that, Miss LIKITA KALLURI , RASHI KUDALE has submitted a Project Report on ROOMIEHUB to MIT – ADT University, Pune for the award degree of Master in Computer Application (Cloud Computing) submitted during the academic year 2024-25

We further certify that to the best of our knowledge and belief, the matter presented in this project has not been submitted to any Degree or Diploma course.

***PRN No: ADT23MGTM0824***

***PRN No: ADT23MGTM0835***

Dr. Alkawati Magadum Dr. Sangita Phunde Dr. Vijay Gondane Prof. Dr. Sunita Karad

**HOD, MCA** **Principal**  **PG HEAD Executive Director**

**Internal Examiner** **Sign of Examiners:**

**1.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**External Examiner** **Sign of Examiners:**

**2.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_***

****

**MIT ADT University, Pune**

**MIT College of Management, Pune**

**Program : Master of Computer Application**

**DECLARATION (Group Project)**

We hereby declare that the project work entitled “ROOMIEHUB” submitted to the MIT–ADT University, Pune, is a record of an original work done by us and this project work is submitted in the requirements for the award degree of Master of Computer Application. The project work in this report has not been submitted to any other University or Institute for the award of any degree or diploma.

As authors of this Group Project report, entitled RoomieHub our signatures on the document signify our joint responsibility in this project.

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Seat No./PRN NO** | **Name of Student** | **Functionality/ Module** |
| 1 | ***ADT23MGTM0824*** | LIKITA KALLURI | BACKEND , CLOUD |
| 2 | ***ADT23MGTM0835*** | RASHI KUDALE | FRONTEND, API |

LIKITA KALLURI RASHI KUDALE

Sign.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Sign.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This Group Project Report is authored by MIT College of Management Pune students and has been reviewed and approved by:

Prof . Harshit Kumar Prof . Harshit Kumar

Sign\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Sign \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

****

**MIT ADT University, Pune**

**MIT College of Management, Pune**

**Program : Master of Computer Application**

**CERTIFICATE OF THE GUIDE**

This is to certify that, Miss. LIKITA KALLURI, RASHI KUDALE of MCA Course (Cloud Computing) have successfully completed their Project Work Titled “ROOMIEHUB”, under my guidance during the Academic Year 2024-2025.

Date:

**Prof. Harshit Kumar**

Project Guide Signature:

****

**MIT ADT University, Pune**

**MIT College of Management, Pune**

**Program : Master of Computer Application**

**ACKNOWLEDGEMENT**

I would like to convey my sincere gratitude to all those who have been instrumental in the development of the project.  I am greatly thankful to Honorable **Dr.** **Prof. Sunita Karad**, Executive Director of MITCOM for all her timely support.

 I express my gratitude to the PG Head **Dr. Vijaya Gondane** & Principal **Dr. Sangita Phunde** and Head of MCA Department **Dr. Alkawati Magadum** who helped me in my extreme solutions.

 I am also thankful to Prof. ***Harshit Kumar,*** my internal project guide for his/her invaluable guidance, help and great support during the project work.

 I am greatly thankful to the staff of MITCOM, Pune for helping me through the entire course.

**Student Name & Signature: Student Name & Signature:**

**Date:**

**Place: MITCOM, MIT-ADT University, Pune**

**INDEX**

|  |  |  |
| --- | --- | --- |
| **Sr. No.** | **Title** | **Page No** |
| 1. | **Abstract** | **1** |
| 2. | **Introduction**   * problem statement * objective and goals * literature survey * project scope | **2** |
| 3. | **System analysis**   * Existing systems * scope and limitations of existing and proposed systems * project features * project Feasibility * Requirement analysis – Functional requirements, performance requirements, security requirements etc. | **7** |
| 4. | **System Design**   * Design constraints (Table) * System Model: UML Diagrams and ER Diagram * User interfaces design (Input/Outputs screens) | **11** |
| 5. | **Reports Testing**  Test Plan, Black Box Testing or Data Validation Test Cases, White Box Testing or Functional Validation Test cases and results. | **25** |
| 6. | **Cloud Architecture**  • Virtual Cloud Network (VCN)  • Storage • Security • Compute | **26** |
| 7. | **Cloud Services**  • Detailed Description of Required Services for Deploying Application on Cloud | **28** |
| 8. | **Deploying API for the Application** | **31** |
| 9. | **Drawbacks and Limitations** • Proposed Enhancements | **34** |
| 10. | **Conclusion And Recommendation** | **36** |
| 11. | **Future Scope** | **37** |
| 12. | **Bibliography and References** | **39** |

**Abstract**

The Hostel Management System is a web application built with the MERN stack (MongoDB, Express.js, React.js, Node.js) that is meant to automate and streamline hostel-related administrative and student tasks. The system allows for effective management of hostel day-to-day operations using a responsive and intuitive interface.

Major functionalities are admin features like student registration, attendance, complaint and suggestion management, mess management, and invoice generation. Students, on the other hand, can log in to view their attendance, apply for mess leave, view invoices, and file complaints or suggestions from their dashboard.

The software is deployed safely on the cloud for high availability, scalability, and real-time accessibility from anywhere. Cloud deployment boosts system performance, facilitates remote usage, and offers a professional touch appropriate for institutional adoption. Utilizing the latest technologies and cloud infrastructure, the system offers a safe, efficient, and paperless experience of hostel management.

**Introduction:**

In the modern technologically advanced world, effective management of resources is no longer a choice but a need. Schools have huge rolls of students, and manually managing their accommodation needs results in administrative inefficiencies, redundancy of data, human inaccuracies, and discontent among stakeholders. Paper-based and physically dependent hostel management systems are archaic in a world where students prefer digital self-service solutions.

RoomieHub is a cloud-hosted Hostel Management System designed to automate everything that matters in a hostel: students' registrations, room allocations, fee payments, complain handling, notice broadcasting, and reporting of data. It was developed using the MERN stack (MongoDB, Express.js, React.js, Node.js) and hosted on Amazon Web Services (AWS) for scalability, security, and high availability.

RoomieHub is envisioned to benefit not only universities, but also private hostel providers and institutions wishing to automate their administrative processes. Through digitalization of processes, it increases transparency, minimizes administrative expenses, enhances operational effectiveness, and grants real-time access to stakeholders at any time and place.

The system is architected with the key considerations of contemporary software systems: availability, scalability, security, performance, and user experience. It uses cloud-native technologies, state-of-the-art web design principles, and well-built backend systems to provide an optimal experience to both hostellers and hostel administrators.

**Problem Statement and Objectives:**

**Problem Statement:**

Conventional hostel management processes are plagued by a number of recurring problems:

Manual Errors: Manual entry of data tends to lead to room allocation errors, missing payment records, or lost complaint registers.

• Limited Access: Students and parents have to physically go to hostel offices to carry out activities like requesting a room or paying a fee.

• Delayed Communication: Notices and complaints are processed through physical means, resulting in delays and miscommunication.

• Ineffective Resource Allocation: Administrators are unable to allocate rooms or monitor occupancy rates in real-time effectively.

• Security Issues: Paper-based systems are susceptible to tampering, loss, and unauthorized access.

• Risk of Data Loss: Physical records are susceptible to destruction by catastrophes such as fire, floods, or negligence.

There is, therefore, an urgent need for a secure, scalable, and digitally accessible Hostel Management System.

**Objectives:**

RoomieHub seeks to achieve the following:

• Automate room allocation, complaint management, registration, and fee collection processes.

• Offer Anytime Anywhere Access via a secure web portal.

• Facilitate Real-time Monitoring of hostel occupancy, complaints, and fee payment status.

• Secure Data through cloud hosting, secure APIs, and role-based access control (RBAC).

• Reduce Operational Costs by minimizing paperwork, physical visits, and administrative overheads.

**•** Provide Analytical Insights through dashboards and reporting tools to aid decision-making.Through these goals, RoomieHub aims to revolutionize hostel management with cloud-driven, user-centric solutions

**Literature Survey:**

* The development of hostel management systems can be categorized into four distinct phases:

**1. Manual Management Systems**

In the initial stages, hostel management was purely manual:

* Room allocations, fee payments, and complaints were registered in physical books.
* Notices were posted physically on notice boards.
* Fee collections required dealing with cash, causing high risk of theft or mistakes.
* Retrieval of past data was slow and inefficient.

**Disadvantages:**

* + Slow
  + Risk of data loss
  + Lack of transparency
  + Bad scalability

**2. Standalone Computer Applications**

* + During the 2000s, institutions moved to offline desktop applications:
  + Local databases (Microsoft Access, MySQL on local servers).
  + Offered simple CRUD operations (Create, Read, Update, Delete).

**Disadvantages:**

* + No remote access
  + Single-user or highly restricted concurrency
  + Manual backups necessary

**3. Web-based Self-hosted Systems**

* + With increased internet penetration:
  + Institutions self-hosted hostel management systems on local servers.Students could access portals from within campus networks.
  + Systems could manage multiple hostels within a single institution.
* **Drawbacks:**
  + High cost of operations (server hardware, IT maintenance staffs).
  + No flexibility: hard to cope with spontaneous user traffic surge.
  + Prone to server crash, local catastrophe.

**4. Cloud-Based Management Systems (Latest Trend)**

The latest solutions exploit cloud computing:

* + AWS, Azure, and GCP provide elastic, secure infrastructure.
  + Institutions pay per real usage of resources ("Pay as you go").
  + High availability (HA) configurations minimize downtime to a great extent.

**Research Findings:**

* + Cloud systems cut downtime in operations by 45%.
  + Institutions saw 30-40% augmented administrative efficiency.
  + Students like mobile-accessible, online self-service portals.
  + Therefore, the development of RoomieHub is totally in line with today's industry best practices, with emphasis on cloud-native architecture and user experience.

**Project Scope:**

The scope of this project includes the design, development, and deployment of a Hostel Management System with the MERN stack (MongoDB, Express.js, React.js, and Node.js), to digitize and automate hostel operations like room allotment, student registrations, complaint tracking, and staff coordination.

Important functional aspects of the system are:

* + User Authentication & Role Management: Independent access levels for students, wardens, and administrators.
  + Room Allotment & Management: Room allocation, tracking availability, and hostel room reassignment.
  + Student Complaints Handling: Students can report complaints which are directed to relevant staff or department.
  + Profile Management: Users can edit and maintain their personal information.
  + Notifications and Announcements: Admins can publish notices accessible to particular roles.

**Cloud Deployment Scope**

* + To facilitate scalability, accessibility, and high availability, the system is hosted on Amazon Web Services (AWS). This involves:EC2 (Elastic Compute Cloud): Hosting the backend (Node.js + Express) and frontend (React.js) components.
  + **MongoDB Atlas / Amazon DocumentDB**: To host the database of the application on the cloud.
  + **S3 (Simple Storage Service)**: To host static assets and uploaded content by users (if required).
  + **Route 53 / Elastic Load Balancer**: To route DNS and manage traffic.
  + **IAM Roles & Security Groups**: To apply secure access control and manage permission across AWS services.

Through utilizing AWS cloud infrastructure, the project guarantees:

* + **Scalability:** To handle growing user loads smoothly without performance decline.
  + **Reliability:** Less downtime with high availability architecture.
  + **Security:** Improved data security and secure access by AWS best practices.
  + **Cost-effectiveness**: Pay-as-you-go model with efficient use of resources.

**System Analysis**

**Existing System:**

Traditional practices of hostel management continue to use today:

* + Paper Registers for marking attendance and room assignment.
  + Cash Payments processed manually with receipts.
  + Manual Grievance Resolution where students need to physically come to the office.
  + Physical Notices on walls for announcements.

**Limitations of Current Systems:**

* + No Remote Access: Parents and students cannot access services remotely.
  + Error-prone: Hand entry of data increases chances of misassignments and miscommunication.
  + Vulnerable Data: No backup systems; physical records are easily damaged.
  + Delayed Response: Problems and grievances take more time to settle.
  + No Real-time Tracking: No real-time view of hostel occupation or fee status.
  + Resource Wastage: Ineffective utilization of available rooms.

**Scope of Proposed System:**

* + A Fully Web-Based System with 24x7 access.
  + Real-Time Room Booking and availability monitoring.
  + Online Complaint Handling system.
  + Digital Fee Collection coupled with future online payment facilities.
  + Role-Based Dashboards for admin and students.
  + Data Backup and Disaster Recovery using AWS.
  + Analytics and Reporting tools for management insights.
  + Security Implementation with encryption, RBAC, and secure communication.

**Project Features:**

The Hostel Management System consists of the following key features:

**1. User Registration & Login**

* Secure student, warden, and admin sign-up and login system.
* Role-based access control.

**2. Room Allotment**

* Students can request hostel rooms.
* Admins can allocate and reassign rooms.
* Real-time view of room availability

.

**3.Complaint Management System**

* Students can lodge complaints (e.g., maintenance, electricity, water).
* Staff can view and address issues.
* Status tracking of complaints.

**4.Profile Management**

* Users can view and edit their personal information.
* Admins can manage user data.

**5. Notice Board / Announcements**

* Admins can put up notices for staff and students.
* Students can see notices pertaining to their hostel.

**6. Dashboard Interface**

* Role-specific dashboards (e.g., students view room and complaint status; admins view system-wide stats).

**7. Cloud Integration (AWS)**

* Hosted on AWS for improved availability, performance, and security.

**Project Feasibility:**

**1. Technical Feasibility**

* Built with the MERN stack which is open-source and highly supported.
* Deployable on AWS services like EC2, S3, and MongoDB Atlas with ease.
* Can be horizontally scaled by adding server instances or a load balancer.

**2. Economic Viability**

* Open-source solutions save on development expenses.
* AWS's free tier along with the pay-as-you-go model keeps deployment costs low.
* Affordable for colleges or institutions that want digital hostel operations.

**3. Operational Feasibility**

* + Easy-to-use UI/UX facilitates easy interaction for non-technical users.
  + Can be incorporated into the hostel's day-to-day workflow.
  + Reduces paperwork and accelerates communication and record-keeping.

**Requirement Analysis:**

Functional Requirements

**1. User Roles**

* + Admin, Student, and Warden roles with defined permissions**.**

**2. Authentication**

* + Sign-up and sign-in feature.
  + Password protection and session management.

**3. Room Management**

* + View available rooms.
  + Allot or vacate rooms.
  + Update room status.

**4. Complaint Handling**

* + Register, track, and resolve complaints.
  + Categorized by type and urgency.

**5. Notices/Announcements**

* + Add, edit, or delete announcements.
  + View by all or chosen user roles.

**6. Profile Features**

* Edit personal information.
* Display hostel-related details.

1. **Performance Requirements: Response Time**

* It should respond to user requests within 2 seconds.
* Real-time room availability and complaint status updates

1. **Concurrent Users**

* It should accommodate a minimum of 100+ concurrent users during busy hours.

1. **Cloud Performance**

* AWS auto-scaling and load balancer provide consistent uptime and responsiveness.

**Security Requirements:  
1. Authentication and Authorization**

* Role-based authorization.
* Hashed passwords using encryption (e.g., bcrypt).

**2. Data Protection**

* Role-based authorization.
* Hashed passwords using encryption (e.g., bcrypt).

**3. Cloud Security**

* AWS IAM roles and security groups manage access to services.  
  n Database not publicly accessible; only accessible via server.

**4. Session Management**

* JWT-based or session-based authentication to avoid unauthorized access.

**System Design**

The system design describes the Hostel Management System components' structure, behavior, and interaction. It confirms that the system satisfies the functional and non-functional requirements and still remains scalable, efficient, and secure.

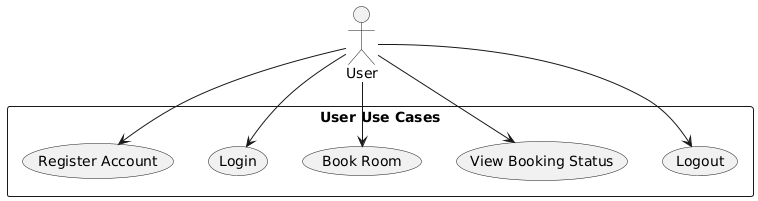
**Design Constraints**

|  |  |
| --- | --- |
| Constraint Type | Details |
| Hardware Constraints | Minimum 4 GB RAM, 2-core processor, 20 GB disk (for EC2 instance) |
| Software Constraints | Node.js v18+, MongoDB, Docker, Jenkins, React.js |
| Platform Constraints | Web-based application, optimized for desktop and mobile browsers |
| Database Constraints | Use of MongoDB with schema for users, rooms, complaints, and notices |
| Network Constraints | Requires internet connectivity to access AWS EC2 APIs and MongoDB |
| Security Constraints | HTTPS access, CORS policy, JWT-based user authentication |
| Development Constraints | Project developed using MERN stack and deployed via AWS Cloud infrastructure |

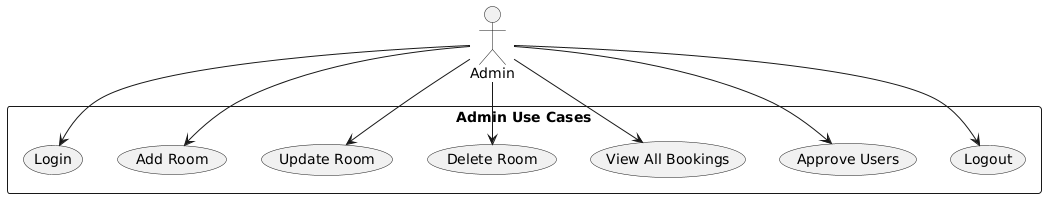
**System Model: UML Diagrams and ER Diagram**

**Use Case Diagram**

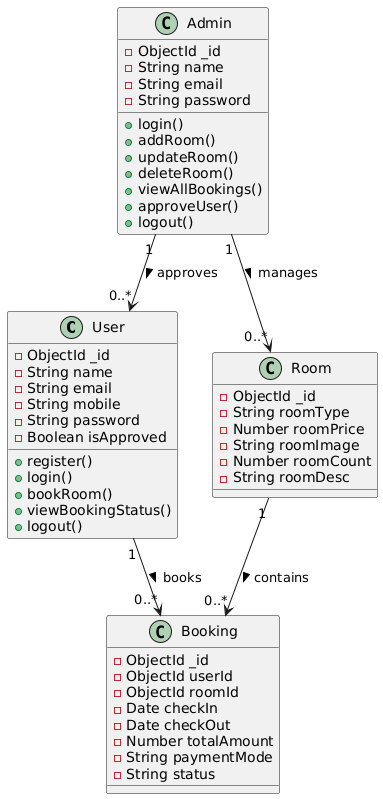
* **User Side**



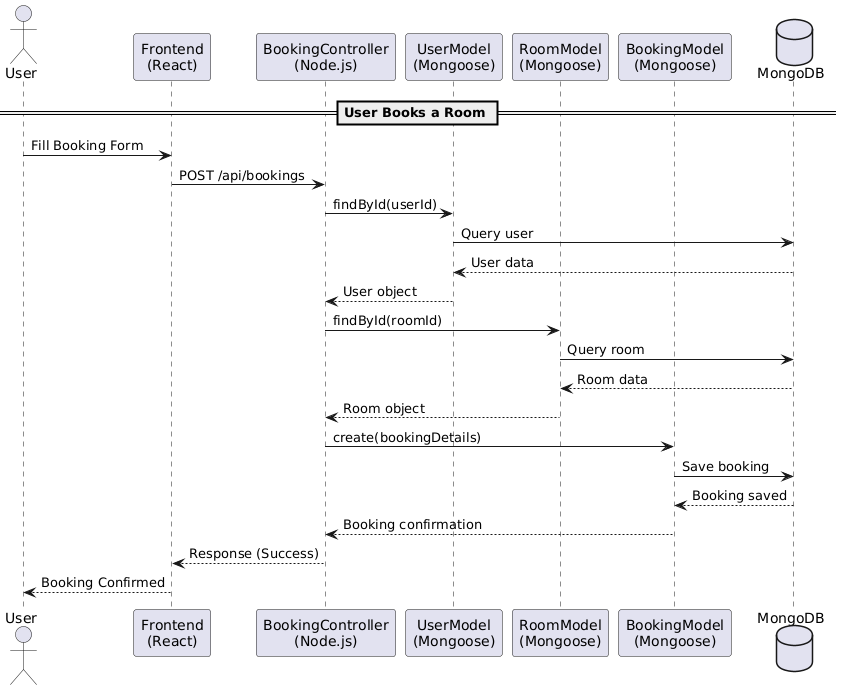
* **Admin side**



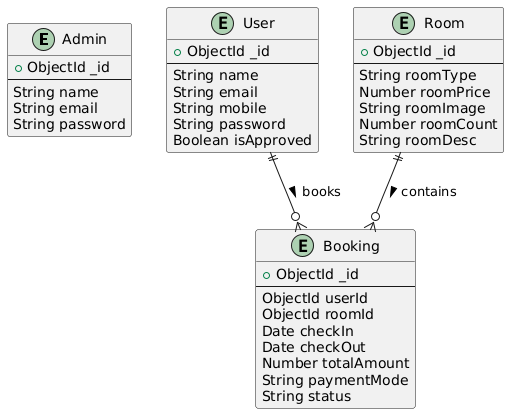
**B. Class Diagram**



**C. Sequence Diagram**

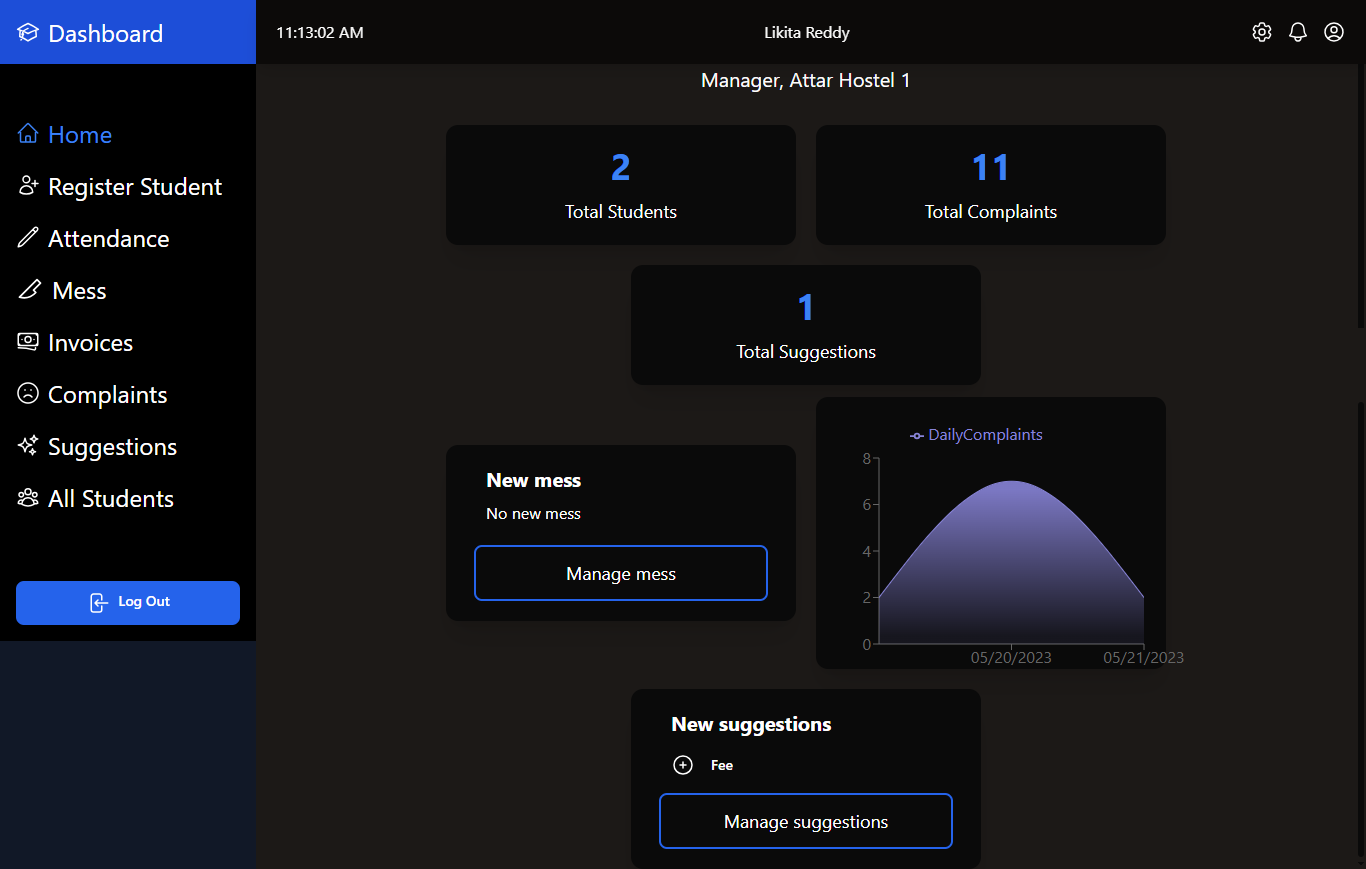


**D. ER Diagram (Entity Relationship Diagram)**

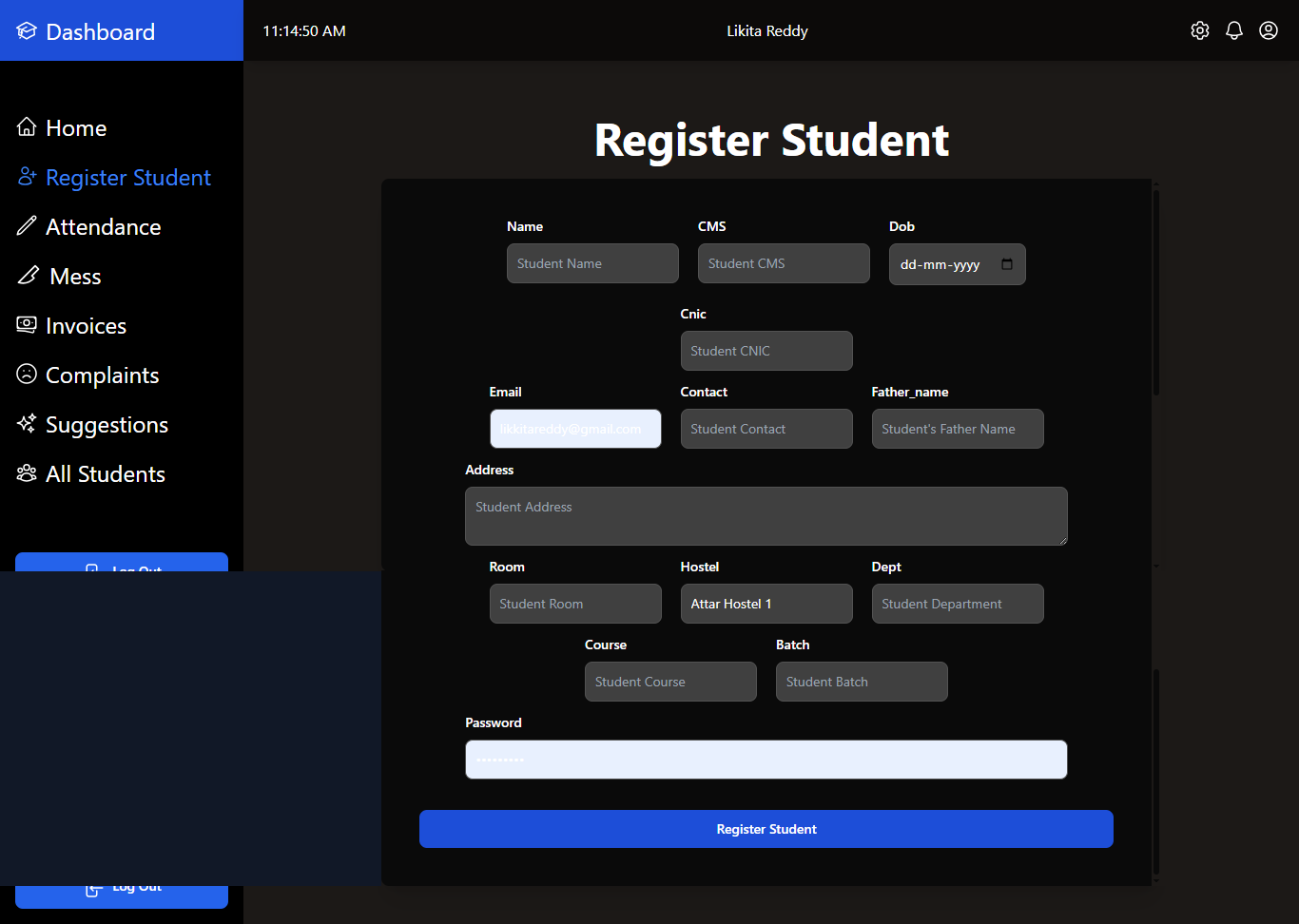


**INPUT & OUTPUT SCREENS:**

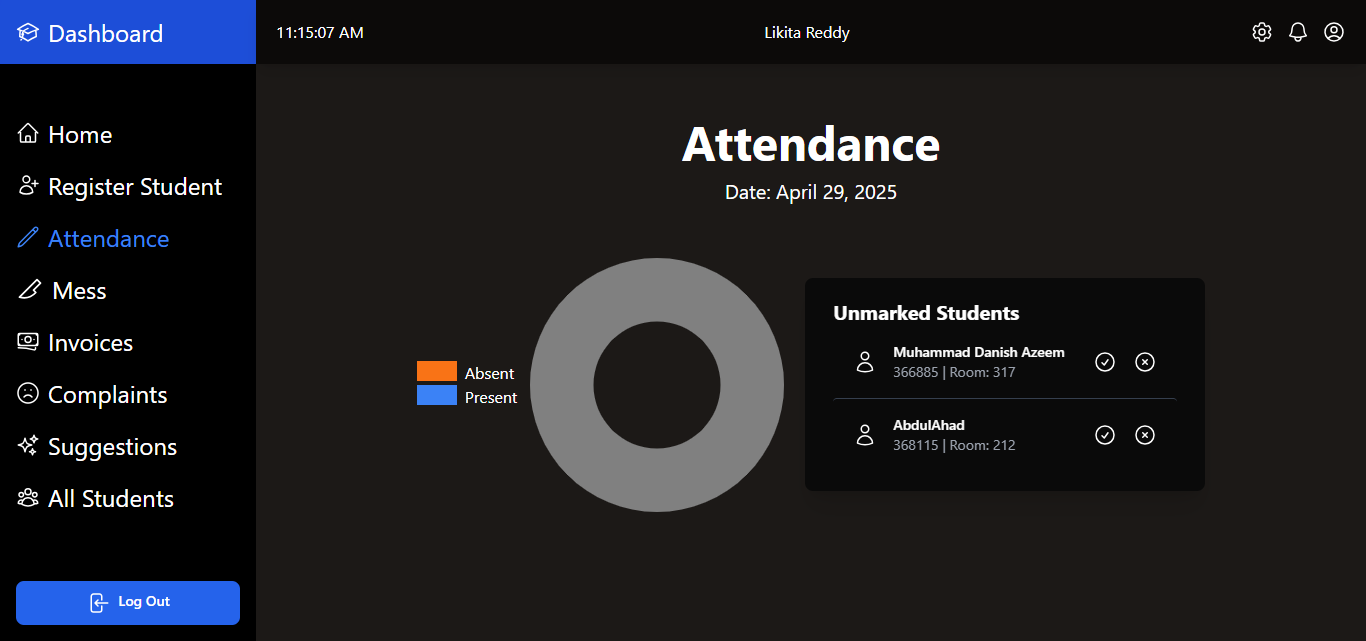
Admin Dashboard:

****

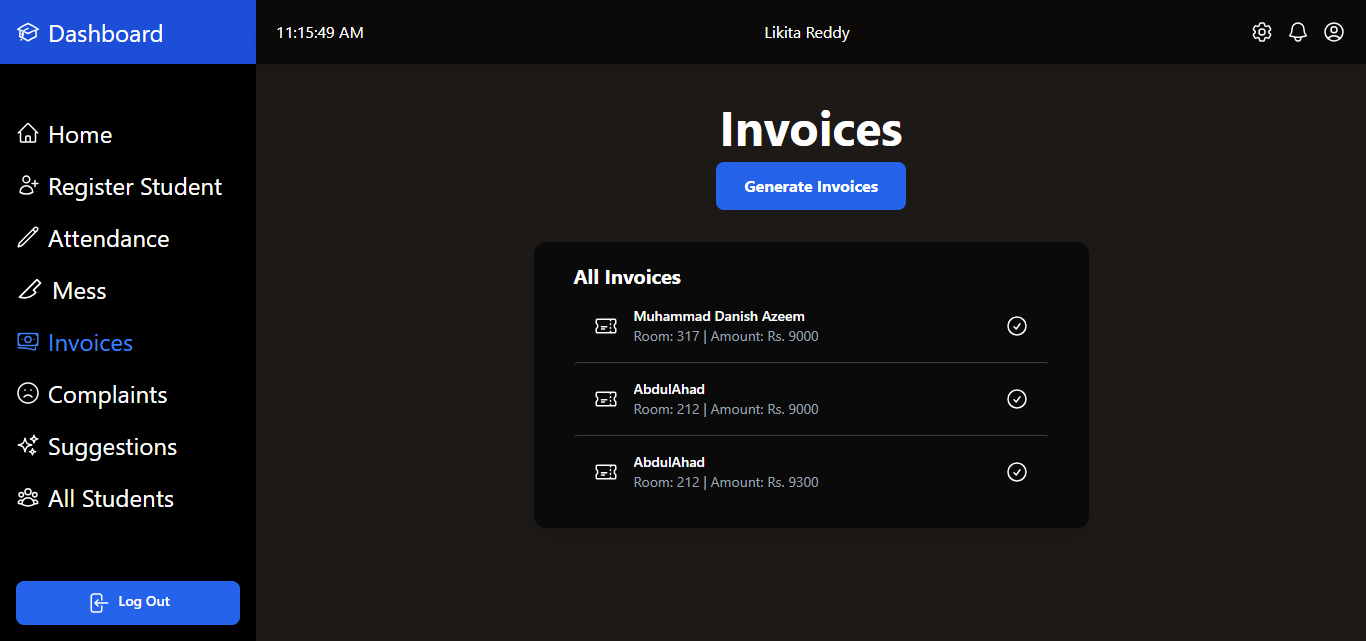
Student Registration :

****

Attendance dashboard:

****

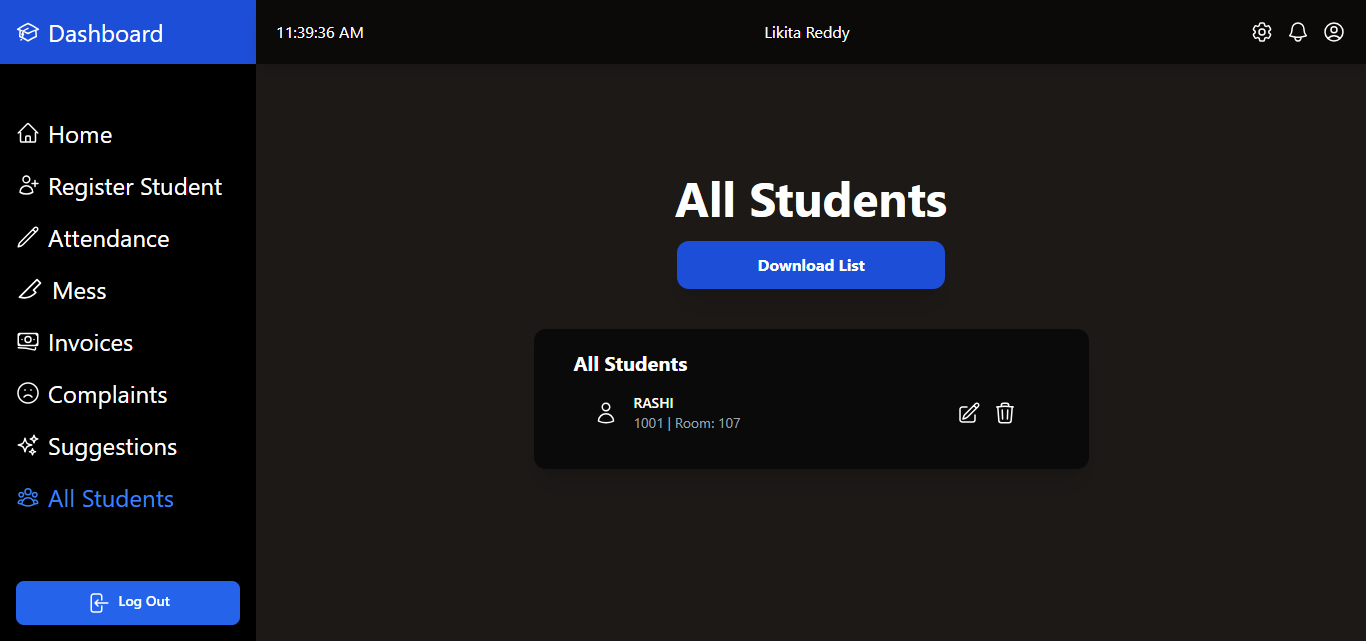
Invoices:

****

Complaints:

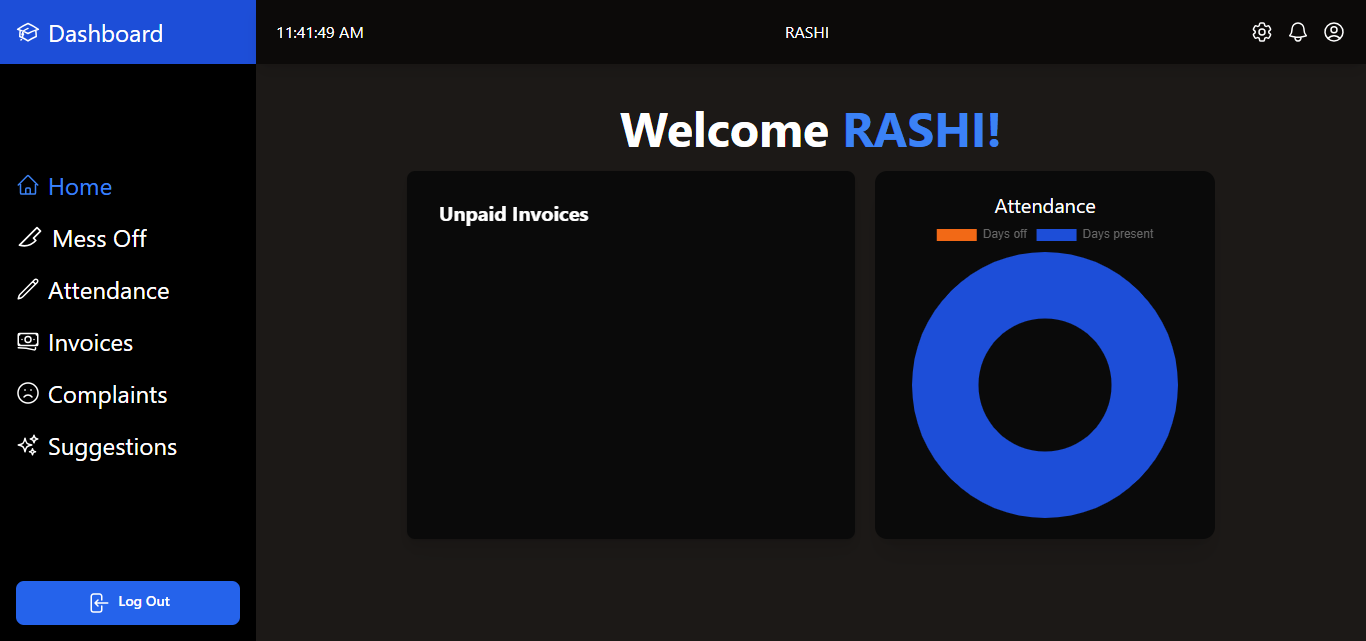
****

Students List:

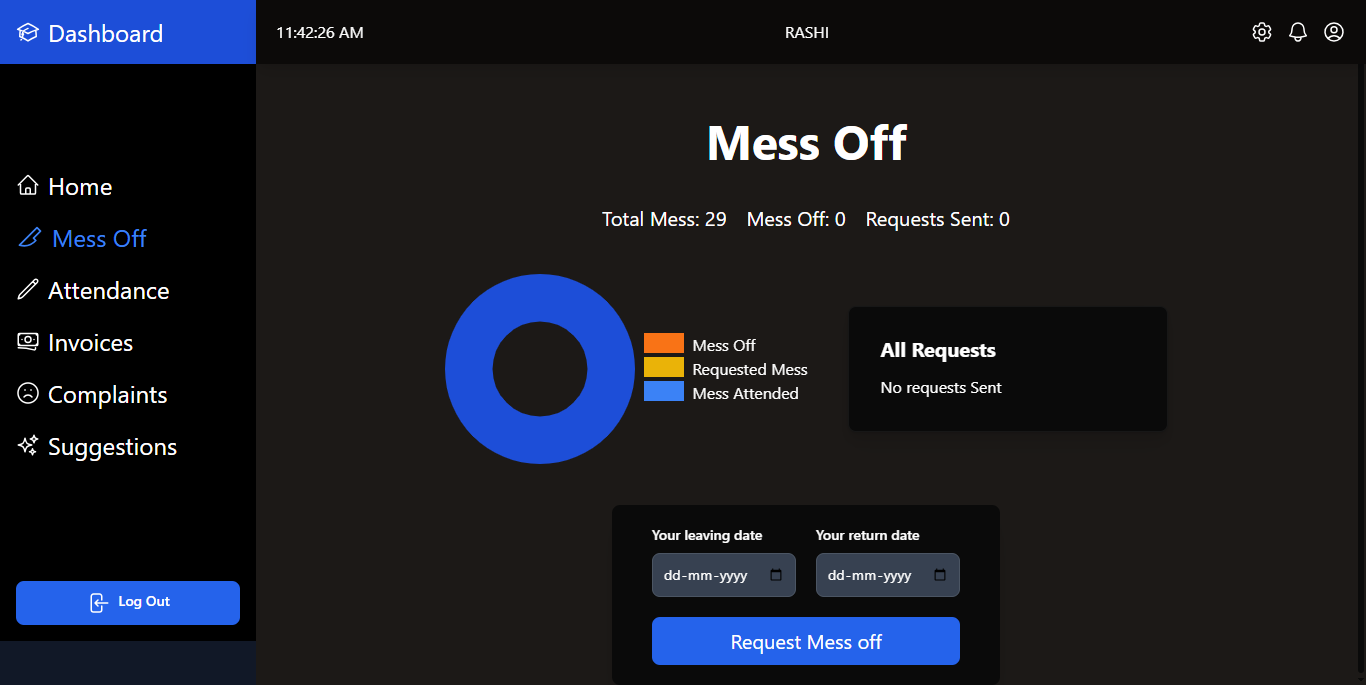
****

**STUDENTS SIDE:**

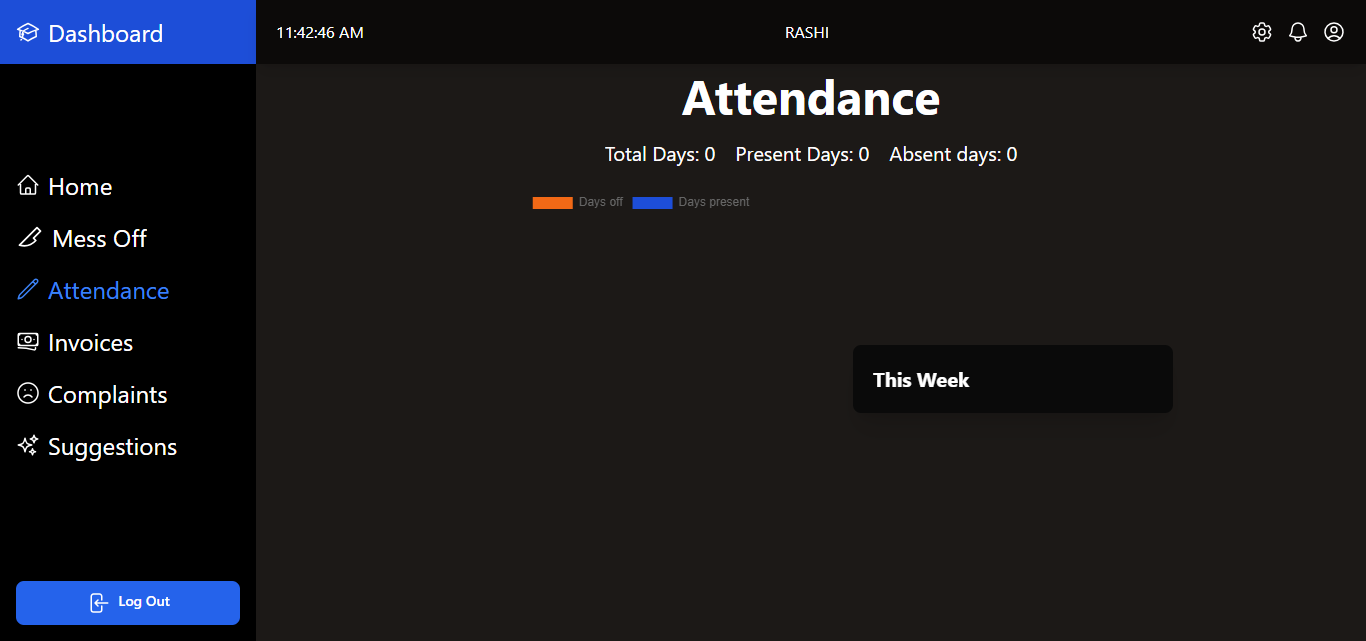
Student’s Dashboard:

****

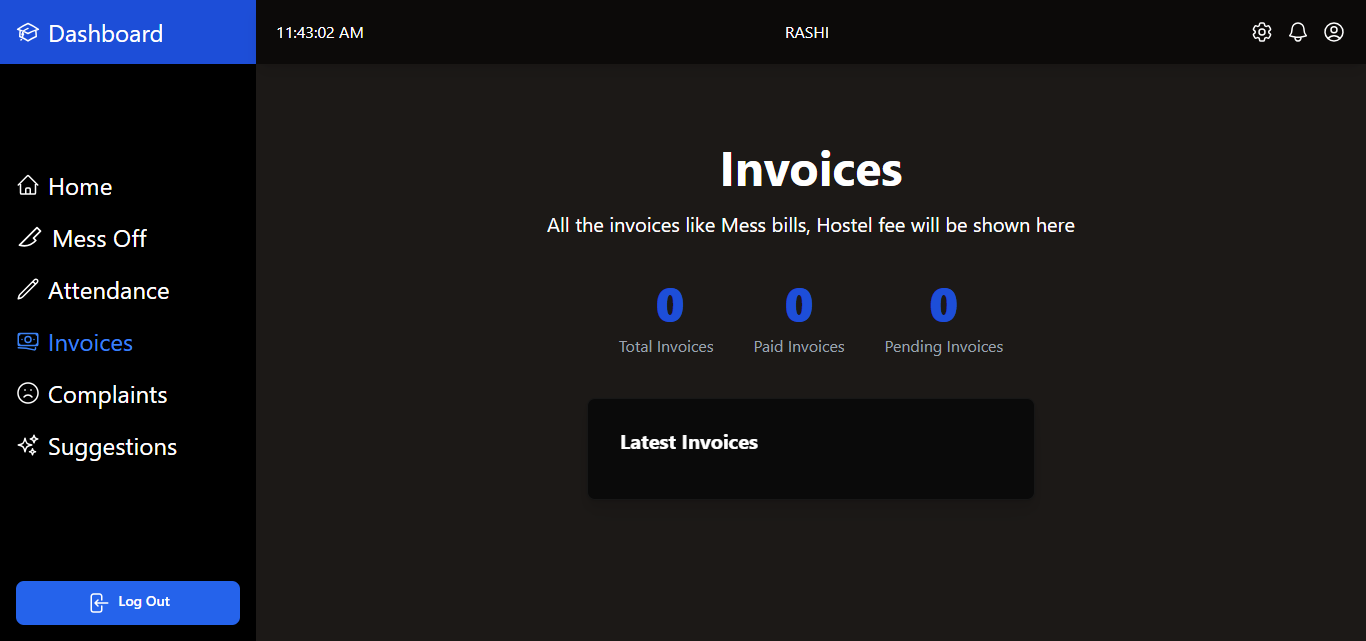
Mess Details:

****

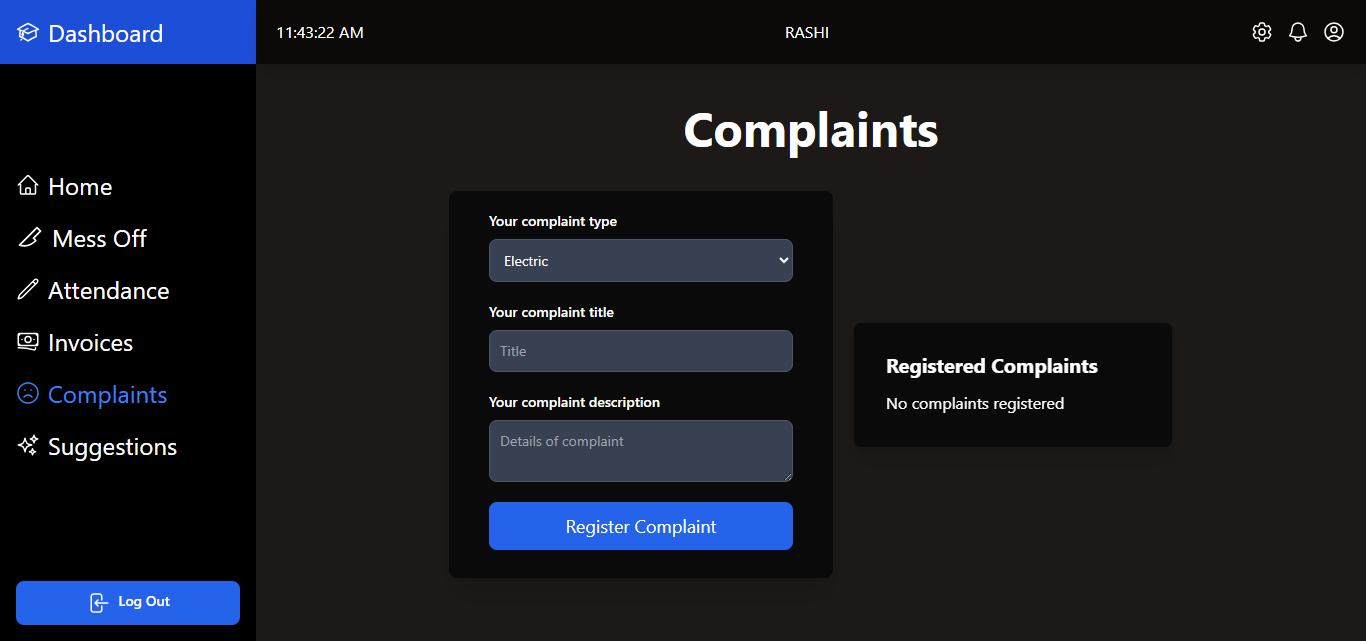
Attendance:

****

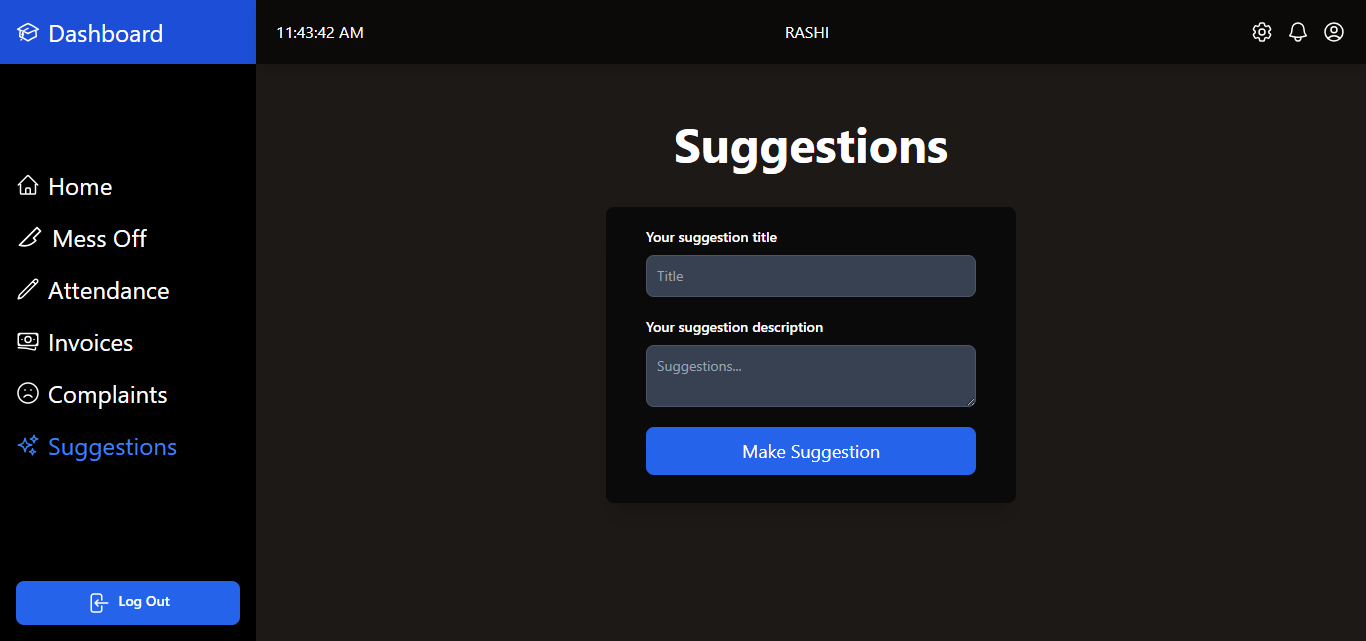
Invoices:

****

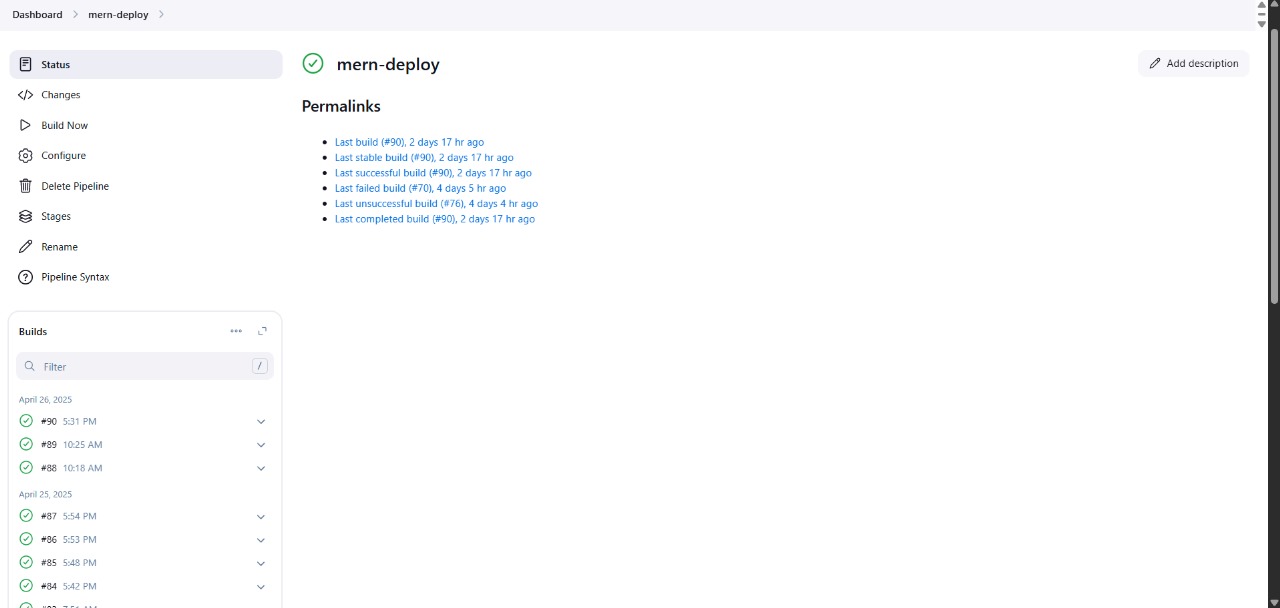
Complaints:

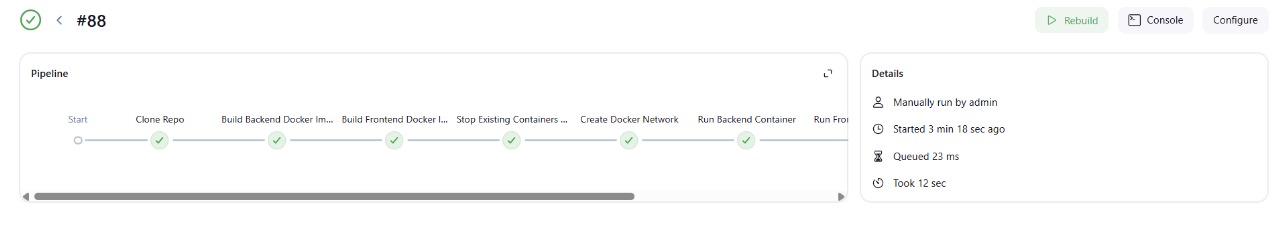
****

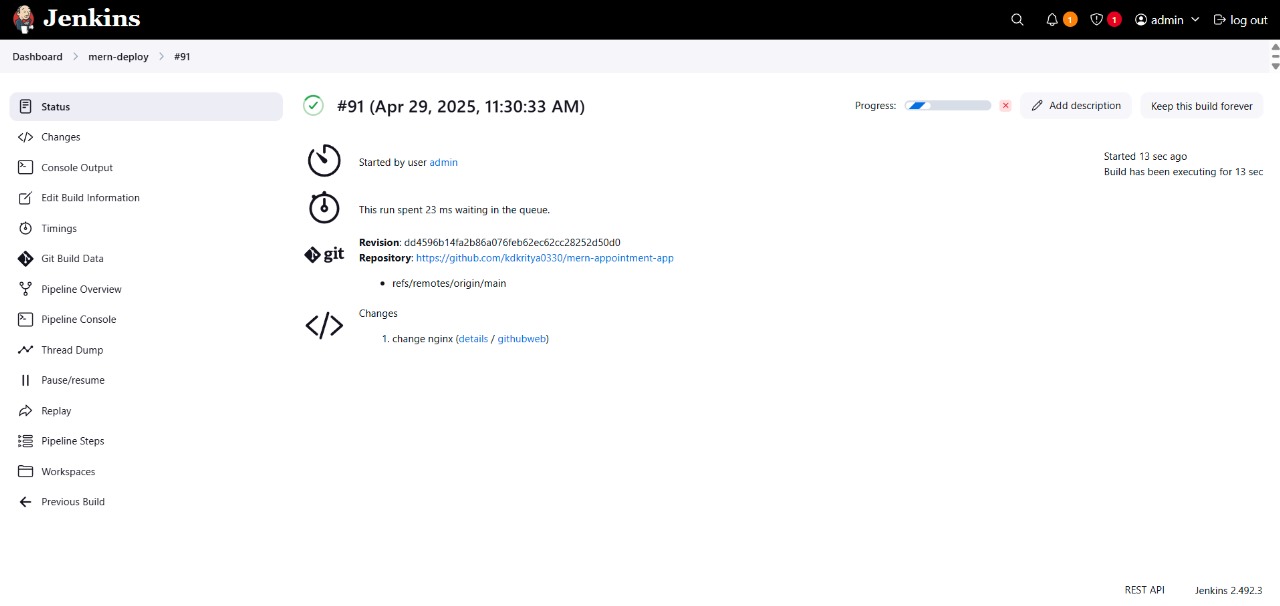
Suggestions:

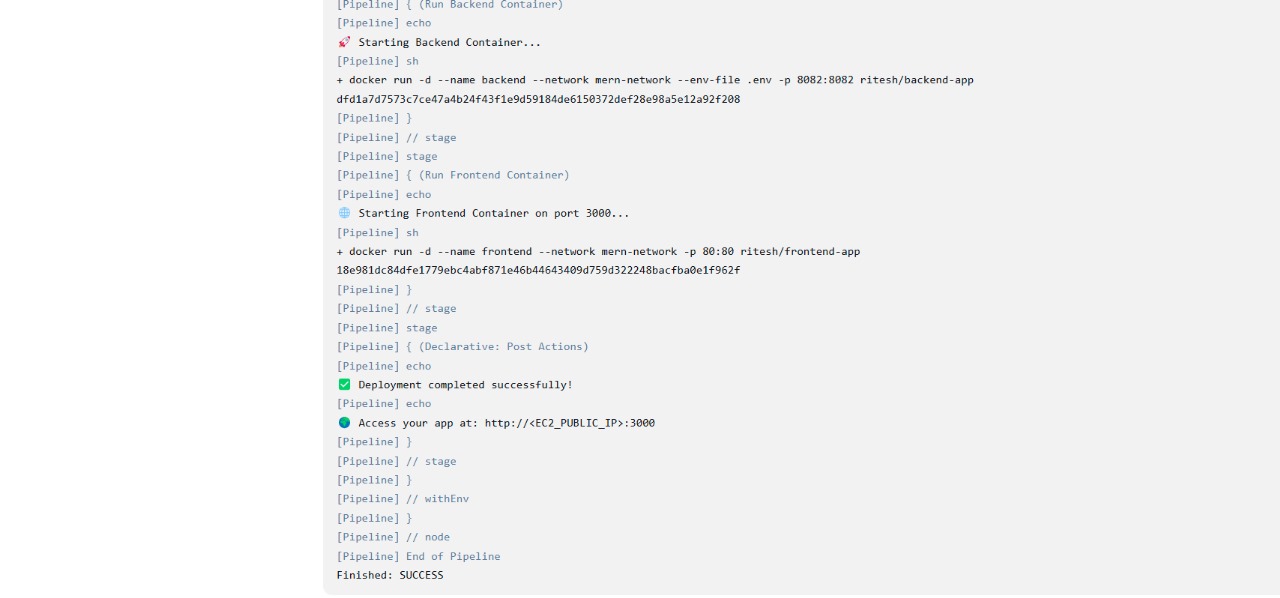
****

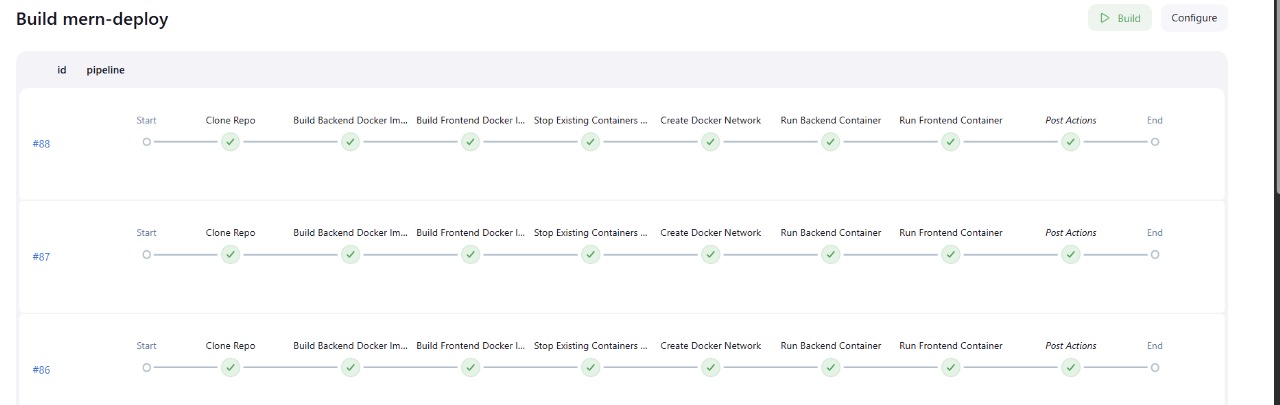
**Cloud Deployment**











**Reports Testing**

**Black Box Testing**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test Case** | **Description** | **Input** | **Expected Output** | **Status** |
| Login Valid | Valid email & password | admin@example.com / admin123 | Redirect to dashboard | ✅ Pass |
| Login Invalid | Incorrect credentials | Wrong email/password | Error message | ✅ Pass |
| Room Allotment | Admin allots room | Student ID, Room No | Room assigned message | ✅ Pass |
| Complaint Submit | Student files complaint | Complaint form | Success message | ✅ Pass |
| View Attendance | Student views attendance | Logged-in user | Attendance table | ✅ Pass |
| Logout | Session logout | Click logout | Redirect to login page | ✅ Pass |

White Box Testing

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Test Case** | **Module** | **Feature Tested** | **Description** | **Expected Behavior** | **Status** |
| API Auth Check | Auth Middleware | checkAuth(req, res) | Verifies user is authenticated prior to access | If token valid, Proceeds | ✅ Pass |
| POST /complaints | Complaints Route | submitComplaint() | Saves complaint to DB | Saves complaint with timestamp | ✅ Pass |
| Room Availability ADMIN Controller | checkRoomAvailability() | Fetches DB for available rooms | Returns list of empty rooms |  | ✅ Pass |
| Mongo Connection | DB Config | mongoose.connect() | Links to MongoDB Atlas | Returns successful connection | ✅ Pass |
| Docker Build | CI/CD | Dockerfile and docker-compose.yml | Constructs containers for app | App operates in containers | ✅ Pass |

**Cloud Architecture**

This project is hosted on Amazon Web Services (AWS) with a secure and scalable cloud architecture. The architecture provides high availability, performance, and security with low cost. The following is a summary of the fundamental cloud components:

**Virtual Private Cloud (VPC)**

A VPC is utilized to establish an isolated virtual network in the AWS cloud.

• Subnets: Split into Public and Private subnets.

* Public Subnet: Runs the frontend (React app) and backend (Node.js/Express) on EC2.
* Private Subnet: Runs the MongoDB database (if Amazon DocumentDB is used instead of MongoDB Atlas).

• Internet Gateway: Provides communication between public subnet resources and the internet.

• NAT Gateway (optional): Provides access for private subnet instances to reach the internet securely for updates.

**Compute:**

**EC2 (Elastic Compute Cloud):**

* Home to the Node.js backend and React frontend.
* Enrolled with Auto Scaling to adapt to fluctuating traffic.
* Attached to the VPC's public subnet.

**Elastic Load Balancer (ELB):**

* Routes incoming traffic across EC2 instances for load balancing and availability.

**Storage:**

• MongoDB Atlas:

* A cloud-hosted NoSQL database employed to save all user information, room information, grievances, etc.
* Hosted on a cloud hosting system and accessed securely through backend APIs.

• Amazon DocumentDB:

* Employed if database to be hosted directly within AWS.
* Hosted in the private subnet for added security.

• Amazon S3 (Simple Storage Service):

* Employed for storing static resources such as images, documents, and notifications.
* Offers versioning and lifecycle management.

**Security:**

• IAM (Identity and Access Management):

* + Defines policies and roles to manage access to AWS services and resources.
  + Ensures least-privilege principle for secure operation.

• Security Groups:

* + Similar to a firewall regulating traffic to and from EC2 instances.
  + Only necessary ports (e.g., 80, 443, 3000) are exposed.

• SSL/TLS Encryption:

* + Ensures HTTPS communication between users and the application to be secure.

• Database Security:

* + MongoDB Atlas or DocumentDB set up with IP whitelisting and encryption at rest.

**CLOUD SERVICES**

**Amazon EC2 (Elastic Compute Cloud)**

Definition:

Amazon EC2 is a web service that offers secure, resizable compute capacity in the cloud. In simpler terms, EC2 allows us to rent virtual machines (called instances) where we can install and run our own software, just like on a physical computer.

Why EC2 is used:

* We require a cloud-based server on which we can execute our application as well as automation tools (such as Jenkins).
* We require complete environmental control to set up Docker and deployment.

Usage in Project:

• Two instances of EC2 are created:

* EC2 Instance 1: The Production Server.
* executes backend Docker containers for (Node.js + Express), frontend Docker container for (React), and MongoDB database.
* EC2 Instance 2: Executes Jenkins for handling automatic CI/CD pipelines.
* Jenkins pulls code, builds Docker containers, and hosts the application onto Instance 1.

EC2 Configuration Example:

* Ubuntu Server 20.04 LTS
* Instance Type: t2.medium (2 vCPUs, 4 GB RAM) for medium workload

Security Groups set to allow:

* SSH (port 22) – administrator access
* HTTP/HTTPS (ports 80/443) – public access to the application
* Custom ports for Docker containers if necessary

Advantages:

* Scalable: Can resize instance as traffic increases.
* Reliable: Supported by AWS infrastructure.
* Secure: Can set up firewalls, key-pairs, and role-based access.

**Docker:**

Definition:

Docker is an open-source system that aims to automate the deployment of applications as light, portable containers that contain everything required to run the software.

Why Docker is used:

* To bundle the MERN application (frontend, backend, database) into isolated containers.
* To make the application run in exactly the same manner in development, testing, and production.

Use in Project:

• Each component of the MERN stack is containerized:

* mongo: MongoDB database container.
* backend: Node.js + Express API container.
* frontend: React app container hosted on Nginx or Node server.

• Docker Compose is employed for defining and executing several containers simultaneously.

Advantages:

• Rapid deployment and rollback.

• Simple scalability and container orchestration.

• Reduces dependency and compatibility problems.

**Jenkins:**

Definition:

Jenkins is an open-source automation server that helps automate parts of the software development process, primarily building, testing, and deploying code.

Why Jenkins is utilized:

• To facilitate Continuous Integration: Whenever a developer commits code to GitHub, Jenkins would automatically build the latest code.

• To automate Continuous Deployment: Upon building the Docker images, Jenkins would deploy the containers to the EC2 production server.

Utilization in Project:

* Jenkins is installed on a different EC2 instance.
* It integrates with your GitHub repository through webhooks.
* It executes the following CI/CD pipeline:

1. Catch code changes via webhook.

2. Clone the changed repository.

3. Build new Docker images for backend and frontend.

4. Push images to DockerHub (optional).

5. SSH into production EC2 server and restart containers with docker-compose.

Advantages:

• Zero manual steps for deployment.

• makes the application always up to date.

• can be extended with tests, notifications, and monitoring.

**Deploying API for the application**

The Hostel Management System API is built using Express.js and Node.js, and it acts as the central backend service responsible for processing all business logic, database operations, and HTTP requests from the frontend. For ensuring reliability, scalability, and automation, the API is containerized via Docker and deployed on an Amazon EC2 instance via Jenkins CI/CD pipelines.

**API Configuration and Environment:**

The API has several endpoints for:

* Registration and login of students and staff
* Assignment of rooms and tracking of vacant rooms
* Making complaints and resolving them
* Notifications and announcements

The backend is linked to a MongoDB database, either through a Docker container or MongoDB Atlas. Environment variables are utilized to secure database URIs and secrets.

**Dockerizing the API:**

To make the API portable and deployment-capable, a Dockerfile is employed:

FROM node:18-alpine

WORKDIR /app

COPY package\*.json./

RUN npm install

COPY.

EXPOSE 5000

CMD ["node", "server.js"]

This builds a light Docker image that can be executed reliably on any server or machine.

**3. Docker Compose Setup:**

For multi-service deployment (e.g., MongoDB + API), a docker-compose.yml file is employed:

version: '3'

services:

mongo:

image: mongo

container\_name: hostel-mongo

ports:

- "27017:27017"

api:

build:./backend

container\_name: hostel-api

ports:

- "5000:5000"

depends\_on:

- mongo

environment:

MONGO\_URI=mongodb://mongo:27017/hostelDB

This setup will start MongoDB and the API service simultaneously in interconnected containers.

**4. Deploying on AWS EC2 with Jenkins:**

In order to host the API on AWS, the following are done:

1. EC2 Instance Setup:

* An Ubuntu-based EC2 instance is launched.
* Docker and Docker Compose are set up.
* Security groups permit traffic on port 5000 (API), SSH (22), and MongoDB (optional/internal).

2. Jenkins Automation:

* Jenkins is installed onto another EC2 instance.
* A GitHub webhook calls the Jenkins pipeline whenever code is pushed.
* Jenkins pulls the code, builds the Docker image, and transfers the files to the EC2 host through SCP or SSH.
* Jenkins runs docker-compose up -d --build remotely to deploy the API

**Drawbacks and Limitations**

• While RoomieHub is far better than traditional hostel management systems and utilizes current cloud infrastructure, there are a few disadvantages and limitations in the current version.

The primary limitation is that the backend server is running on a single Amazon EC2 instance. The design has one point of failure. If the EC2 instance goes down because of hardware failure, server crash, or scheduled maintenance downtime, the whole system is not available to users until manual intervention to resume services. In a production-level environment, such downtime is undesirable.

• Scalability is another very important limitation. RoomieHub presently relies on manual scaling processes. In case the user base significantly increases, particularly around the admission periods when room applications peak, the existing system would need to be manually upgraded with servers or new instances deployed. The absence of automatic scaling means there is a possibility of compromised performance, slower response times, or even server crashes when there is high load.

• Moreover, RoomieHub does not have a load balancing feature. All the traffic of the users goes to a single server, resulting in unequal load distribution and higher chances of server overload. During a traffic surge, without an Elastic Load Balancer, the server can become unresponsive or slow, impacting user experience and system stability.

• Monitoring and alerting are not extensive as of now. While the backend has PM2 monitoring process health, no advanced monitoring system such as AWS CloudWatch is fully set up. This leads to a reactive system where things are only fixed after getting complaints from users or doing manual checks, as opposed to detecting performance bottlenecks or running out of resources beforehand.

• Security controls, while robust at a fundamental level, can be further bolstered. There is no Web Application Firewall (WAF) to provisionally defend the system against frequent web assaults such as SQL injection, cross-site scripting (XSS), and distributed denial-of-service (DDoS) attacks. Additionally, SSL certificates for HTTPS encryption are not automatically provisioned using AWS Certificate Manager but rather manually planned.

Backup and disaster recovery strategies need to be improved as well. While MongoDB Atlas offers automated database backup, there are no automatic backups of EC2 server data, application logs, or configuration files. In the event that RoomieHub is inadvertently terminated by the user, re-deployment of RoomieHub would require additional effort and result in increased downtime.

• Lastly, although the React frontend is successfully hosted on Amazon S3, it is simply static and does not support dynamic rendering features such as Server-Side Rendering (SSR). This is a limitation to search engine optimization (SEO) and dynamic serving of personalized content, which may be important for future scaling.

• Generally, although RoomieHub brings notable improvements, solving these limitations would further enhance the system to be even stronger, scalable, secure, and production-ready.

**Conclusion and Recommendations**

The effective development and deployment of RoomieHub represent a major milestone towards the modernization of hostel management systems in academic institutions. With the inclusion of cloud-native architectures and the capability of the MERN stack, RoomieHub reconditions the conventional, laborious, and error-probable manual hostel operations to a seamless, secure, and highly accessible digital platform.

The system supports vital operations seamlessly, including student enrollment, room assignment, complaint handling, tracking of fees, and notice posting. It is deployed in the cloud on AWS for high availability, strong security, and worldwide accessibility, providing an end-to-end, seamless experience to hostel administrators and students alike.

RoomieHub secures sensitive data through role-based access control, SSL/TLS encryption, and secured APIs to ensure stringent security practices are followed.

Even with its present strengths, RoomieHub's original version has areas for improvement, especially in scalability, automated monitoring, and disaster recovery planning. Recognizing these weaknesses creates a roadmap of future improvements that can make the system more robust, scalable, and enterprise-capable.

Finally, RoomieHub not only succeeds in its initial project goals but also sets the stage for growth in the future. Its modular design, cloud-first approach, and user-centric design principles establish it as a solid foundation upon which to build future development, whether in bringing in mobile applications, AI-based predictive features, serverless architecture, or multi-regional deployment.

**Future Scope**

• In order to make RoomieHub become a very robust and enterprise-class solution, the following future upgrades are suggested.

• Most importantly, hosting RoomieHub as part of an Auto Scaling Group on AWS is essential. Auto-scaling enables automatic adjustment of server capacity with respect to incoming traffic. During high usage hours, additional instances would start automatically, providing smooth performance, while lowering costs during low-usage hours.

• Implementing an Elastic Load Balancer (ELB) would load balance incoming traffic across multiple EC2 instances. Not only does this provide high availability, but it also stops any one instance from being overwhelmed, enhancing overall reliability and user experience.

• Advanced monitoring and alert mechanisms must be implemented with AWS CloudWatch. The establishment of custom metrics and alarms for CPU usage, memory, disk I/O, and API latency would enable proactive responses to possible concerns before they impact end users.

• Security should be improved by invoking AWS Web Application Firewall (WAF) in order to defend against known attacks, and AWS Shield for mitigation against DDoS attacks. Automatic management of SSL/TLS certificates with the help of AWS Certificate Manager would provide uninterrupted secure communication without periodic manual renewal.

• Backup and disaster recovery measures can be enhanced by setting up scheduled EC2 instance snapshots and Amazon Machine Images (AMIs). Such backups would enable recovery of the application instantly in the event of data loss or failure of the server.

• Migrating the backend to serverless architecture based on AWS Lambda Functions and API Gateway is another solid future possibility. A serverless model eliminates the necessity to manually manage servers, provides automatic scaling, and bills only for actual usage of resources, considerably lowering operational costs.

• Furthermore, deploying RoomieHub to multiple AWS regions may also be an option. Having RoomieHub hosted in multiple AWS regions would ensure regional redundancy such that even in case of the whole AWS region shutdown by way of natural disaster or outages, RoomieHub is still running using another region.

• On the frontend, switching to a Server-Side Rendering (SSR) library such as Next.js would greatly enhance SEO, page load speed, and support for serving dynamic content, making for increased discoverability and user interaction.

• On the user experience side, potential updates might include mobile app integration, support for multiple languages, predictive analytics for room availability forecasting with AI, and real-time notification capabilities through services such as Firebase Cloud Messaging or AWS SNS.

• Through the adoption of these improvements, RoomieHub can become a genuinely world-class, robust, and future-proof hostel management platform.

**Bibliography and References**

**Books:**

* Alex Banks and Eve Porcello, Learning React: Modern Patterns for Developing React Apps, O'Reilly Media.
* Mario Casciaro and Luciano Mammino, Node.js Design Patterns, Packt Publishing.
* Kristina Chodorow, MongoDB: The Definitive Guide, O'Reilly Media.
* Roger S. Pressman, Software Engineering: A Practitioner's Approach, McGraw-Hill.
* Srinivasan Desikan and Gopalaswamy Ramesh, Software Testing Principles and Practices, Pearson Education.

**Web References:**

* Official Documentation of AWS (<https://aws.amazon.com/documentation/>)
* MongoDB Atlas Documentation (<https://www.mongodb.com/cloud/atlas>)
* Official Documentation of React.js (<https://react.dev/>)
* Official Documentation of Node.js (<https://nodejs.org/>)
* PM2 Process Manager (<https://pm2.keymetrics.io/>)
* AWS EC2 User Guide
* AWS S3 Static Website Hosting Guide
* AWS CloudWatch Metrics and Alarms Guide
* AWS Elastic Load Balancer Documentation
* AWS Auto Scaling Documentation