HOSTEL MANAGEMENT SYSTEM

Submitted by

Bansari Gupta

For the award of the degree of

Bachelors Of Technology & Computer Science

Under the supervision of
Mr. Sunil Kumar

Supervisor
Mr. Sunil Kumar, Scientist-E, CDAC - New Delhi

Faculty of Mathematics and Computing
Banasthali Vidyapith

Banasthali - 304022

Session: 2025

प्रगत संगणन विकास केंद्र
CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING

इलेक्ट्रॉनिकी और सूचना प्रौद्योगिकी मंत्रालय की वैज्ञानिक संस्था, भारत सरकार
A Scientific Society of the Ministry of Electronics and Information Technology, Government of India

सी डैक
CDAC

भूखंड सं. 20, एफसी-33,
संस्थानिक क्षेत्र, जसोला,
नई दिल्ली - 110 025, भारत
Plot No. 20, FC-33,
Institutional Area, Jasola,
New Delhi – 110 025, India
फोन/ Ph.: +91-11-2694 0239
www.cdac.in

CDACD/EGSG/2024/TRAINEES/2911202417
November 29, 2024

TO WHOMSOEVER IT MAY CONCERN

This is to certify that **Ms. BANSARI GUPTA** D/o Mr. ATUL KUMAR GUPTA (Enrollment No. 2021/389) pursuing B.Tech 4th year (7th Semester) from "Banasthali Vidyapith, Jaipur, Rajasthan" has undergone internship training during the period from **01/07/2024 to 30/11/2024** successfully at "CENTRE FOR DEVELOPMENT OF ADVANCED COMPUTING (C-DAC), NEW DELHI".

During the internship training, she has successfully **designed and developed "Hostel Management System"** using **Spring Boot, MongoDB, Java, React etc.**

During the period of her internship training with us, she was found sincere and hardworking.

Sunil Kumar
(Scientist-E)

Abstract

The **Hostel Management System** is a comprehensive, web-based application designed to streamline the management of hostel operations. It automates essential processes such as student and warden registration, student room allocation, student feedback collection, hostel notice board and other hostel facilities. Built using Java Spring Boot and MongoDB, the system ensures scalability and efficient data handling.

The system allows students to register, log in, and view their room allocation details, while administrators and wardens can manage student records, room availability, and allocation. A feedback module enables students to rate the hostel's cleanliness, food, staff, and facilities, providing valuable insights for service improvements.

Additionally, the system features a Notice Board where important announcements and updates are posted for students, ensuring effective communication between hostel management and residents.

The login and logout functionality enhances security by recording user activity and ensuring proper authentication. By automating and optimizing various hostel management tasks, this system not only saves time but also enhances the overall experience for both students and administrators, making hostel operations more organized and efficient.

Acknowledgement

We would like to express our sincere gratitude to **Sunil Kumar Sir**, our mentor, for his unwavering support, guidance, and encouragement throughout the development of the **Hostel Management System** project. His expertise, constructive feedback, and insightful suggestions were instrumental in the successful completion of this project.

Sunil Kumar Sir's constant availability for discussions, clarification of concepts, and his valuable input during each phase of the project helped us overcome various challenges and enhanced the overall quality of our work. His mentorship has played a pivotal role in shaping our learning and improving our technical and problem-solving skills.

We are extremely grateful for his dedication and the time he invested in helping us achieve the objectives of this project. His mentorship has been a key factor in making this project a success.

I express my sincere gratitude to **Prof. CK Jha** , Dean Dept of Mathematics and Computing, Banasthali Vidyapith and **Dr. Rajiv Singh** , Head of the Department of Computer Science,AIM & ACT, Banasthali Vidyapith, Rajasthan and members of the staff of my university for their constant encouragement during the project work.

Table of Contents

Objective (Problem Statement)

In recent years, the rapid increase in educational institutions has led to a corresponding rise in the number of hostels required to accommodate the growing student population. As a result, managing these hostels has become increasingly complex, putting significant strain on administrators. Traditionally, hostel management has relied on manual processes, which are time-consuming and prone to errors. This project aims to address these challenges by introducing a software solution that automates and streamlines hostel management tasks, reducing the burden on administrators and minimizing the risk of human error.

The Hostel Management System is a website designed to address these challenges by automating key processes such as:

1. **User Registration and Authentication**: Simplifying the process of user enrollment and ensuring secure access through login/logout functionality.
2. **Room Allocation**: Providing an organized approach to allocate rooms to students based on availability and requirements.
3. **Feedback Collection**: Enabling students to provide feedback on various aspects such as cleanliness, food, staff, and facilities, ensuring continuous improvement.
4. **Notice Board**: Offering a digital platform for administrators to post important announcements and updates for students.
5. **Admin and Warden Access**: Allowing hostel administrators and wardens to manage student records, track room availability, and perform administrative tasks effectively.

Requirement Analysis

2.1 Requirement Specification

2.1.1 System Features

1) User Authentication

Description: Allows users to log in with a unique username and password.

Priority: High

Priority Components:

- Benefit: 9
- Penalty: 8
- Cost: 4
- Risk: 5

Stimulus/Response Sequences:

1. User Action: User enters username and password.
   System Response: Validates credentials and logs the user in.
2. User Action: User enters incorrect username or password.
   System Response: Displays an error message and prompts for re-entry.
3. User Action: User logs out.
   System Response: Ends the session and redirects to the login page.

Functional Requirements:

- REQ-1: The system must validate the username and password against stored credentials.
  Error Handling: If credentials are invalid, the system should display an error message.

- REQ-2: The system must manage user sessions securely.
  Error Handling: If session management fails, the system should log out the user and prompt to log in again.
- REQ-3: The system must lock the account after a predefined number of failed login attempts.
  Error Handling: If the account is locked, the system should notify the user and provide instructions to unlock it.
- REQ-4: The system must allow password reset via email verification.
  Error Handling: If email verification fails, the system should prompt the user to retry or contact support.

2) Student Room Allocation

Description: Enables wardens to allot hostel rooms to students, viewable by students and admins.

Priority: High

Priority Components:

- Benefit: 8
- Penalty: 7
- Cost: 5
- Risk: 4

Stimulus/Response Sequences

1. User Action: Warden updates hostel details.
   System Response: Saves and reflects the updated details.
2. User Action: Student or admin views hostel details.
   System Response: Displays the current hostel details.

Functional Requirements

- REQ-1: The system must allow the warden to update room allotment.
  Error Handling: If update fails, the system should notify the warden and provide error details.
- REQ-2: The system must display hostel details to students and admin.
  Error Handling: If retrieval fails, the system should notify the user and retry.
- REQ-3: The system must ensure data integrity and security for hostel information.
  Error Handling**:** If data integrity is compromised, the system should alert the admin.
- REQ-4: The system must provide real-time updates of hostel details.
  Error Handling: If real-time updates fail, the system should notify the user and retry.

3) Hostel Notice Board

Description: Facilitates communication by allowing wardens to update notices for admins and students.

Priority: Medium

Priority Components:

- Benefit: 7
- Penalty: 6
- Cost: 4
- Risk: 4

Stimulus/Response Sequences

1. User Action: Warden posts or updates a notice.
   System Response: Updates the notice board with the latest information.
2. User Action: Admin or student views the notice board.
   System Response: Displays the notices posted by wardens.

Functional Requirements

- REQ-1: The system must allow wardens to post, edit, and delete notices.
  Error Handling: Notify the warden if posting fails and provide error details.
- REQ-2: The system must display notices in chronological order.
  Error Handling: Notify the user if display fails and retry.
- REQ-3: The system must ensure notices are viewable by admins and students.
  Error Handling: Alert the admin if access control fails.
- REQ-4: The system must manage access control to restrict notice updates to wardens.
  Error Handling: Notify the admin if access control fails and restrict updates.

4) Student Feedback System

Description**:** Allows students to submit feedback regarding hostel facilities or services.

Priority: Medium

Priority Components:

- Benefit: 6
- Penalty: 5
- Cost: 3
- Risk: 3

Stimulus/Response Sequences

1. User Action: Student submits feedback.
   System Response: Stores the feedback and notifies admin and wardens.
2. User Action: Admin or warden views feedback.
   System Response: Displays the feedback submitted by students.

Functional Requirements

- REQ-1: The system must allow students to submit feedback.
  Error Handling: Notify the student if submission fails and retry.
- REQ-2: The system must store feedback securely.
  Error Handling: Notify the admin if storage fails and provide error details.
- REQ-3: The system must notify admin and wardens of new feedback submissions.
  Error Handling: Retry notifications and alert the admin if failures persist.
- REQ-4: The system must provide a feedback viewing interface for admin and wardens.
  Error Handling: Notify the user if viewing fails and retry.

5) User Home Page:

Description**:** Allows users to choose which task they want to perform.

Priority: Medium

Priority Components:

- Benefit: 7
- Penalty: 5
- Cost: 3
- Risk: 3

Stimulus/Response Sequences

1. User Action: Student will choose whether they will view their room number or view the hostel notice board or give feedback.
   System Response: Redirect to the required function.
2. User Action: Warden will choose whether they will allot rooms or view the room list or add a notice or view student feedbacks.
   System Response: Redirect to the required function.

3. User Action: Admin will choose whether they will view allotted rooms, hostel notice board or feedback.
   System Response: Redirect to the required function.

Functional Requirements

- REQ-1: The system must allow students to submit feedback.
  Error Handling: Notify the student if submission fails and retry.
- REQ-2: The system must store feedback securely.
  Error Handling: Notify the admin if storage fails and provide error details.
- REQ-3: The system must notify admin and wardens of new feedback submissions.
  Error Handling: Retry notifications and alert the admin if failures persist.
- REQ-4: The system must provide a feedback viewing interface for admin and wardens.
  Error Handling: Notify the user if viewing fails and retry.

2.2 Non-functional Requirements

1) Performance Requirements

- Response Time: The system should respond to user requests within 95% of transactions during peak usage times.
- Throughput: The system should support at least 1000 concurrent users without performance degradation.
- Scalability: The system should scale to handle up to 10,000 users with minimal changes to the underlying architecture.
- Data Processing: Batch processes, such as generating monthly reports, should complete efficiently without user interaction.
- Real-Time Updates: Changes in student information should be reflected across all modules in real time.

Rationale: These requirements ensure the HMS provides a smooth and efficient user experience, even during peak times, and can accommodate future demand.

2) Safety Requirements

- Data Backup: Regular backups of the database must be performed daily to prevent data loss. Backup data should be securely stored off-site.
- Error Handling: The system should include robust error handling to prevent data corruption and ensure system stability.
- User Safety: Sensitive student information, such as personal details or health records, should only be accessible to authorized personnel.
- Regulatory Compliance: The system should comply with relevant safety regulations, such as GDPR, to ensure legal adherence.

Rationale: These safety measures protect user data and system integrity, reducing the risk of loss or harm.

3) Security Requirements

- Authentication: Users must authenticate using a unique username and password. Multi-factor authentication should be implemented for administrative accounts.
- Authorization: Role-based access control (RBAC) should ensure users only access data and perform actions appropriate to their role.
- Data Encryption: Sensitive data, whether in transit or at rest, must be encrypted using industry-standard techniques.
- Audit Trails: The system should maintain an audit log of all user activities, including logins, data modifications, and access to sensitive information, for a minimum of one year.
- Security Compliance: The system must adhere to relevant security standards, such as ISO/IEC 27001, and local data security regulations.

Rationale: These measures protect user data from unauthorized access and ensure compliance with legal standards.

4) Software Quality Attributes

- Usability: The system should have an intuitive user interface to facilitate ease of use for all user classes.
- Reliability: The system should achieve an uptime of 99.9% to ensure continuous availability.
- Maintainability: The system should be modular and well-documented to support updates and maintenance efficiently.
- Interoperability: The system should integrate with other educational and administrative systems using standard protocols.
- Adaptability: The system should accommodate changes in policies or requirements without significant rework.
- Testability: Automated testing tools should be used to ensure high-quality releases.
- Portability: The system should be platform-independent and run on various operating systems and devices.

Rationale: These attributes ensure the system is robust, reliable, and adaptable to future needs.

5) Business Rules

- Role-Based Access: Only authorized personnel, such as hostel managers and administrators, can perform critical functions like room allocation, fee management, and disciplinary actions.
- Disciplinary Actions: Records of disciplinary actions, such as warnings or fines, must only be accessible to authorized users.
- Emergency Protocols: The system should provide quick access to student contact information and emergency procedures.

Rationale: These rules ensure the system operates within defined guidelines, maintains order, and supports the safety and well-being of users.

2.3 User Interface Requirements

1) Login System Interface

Description: Interface for users to log in with their credentials.
GUI Standards: Standard username and password input fields, "Login" button, "Forgot Password" link for password recovery.
Error Message Display: Display error messages as a popup when he user enters an incorrect username or password during login.

2) Room Allotment Interface

Description: Interface for wardens to allot hostel rooms to the students and for students/admin to view hostel details.
GUI Standards: Editable fields, update/save buttons for warden and read-only fields for students/ admin.
Error Message Display: Display error messages inline when there is a failure to update room allotment due to data validation issues.

3) Notice Board Interface

Description: Interface for wardens to post latest notices for admin/students to view them.
GUI Standards: Text area for posting notices, edit/delete options for wardens and read-only fields for students/admin.
Error Message Display: Display error messages inline.

4) Feedback System Interface

Description: Interface for students to submit feedback and for admins/wardens to view it.

GUI Standards: Text area for submitting feedback, list view for displaying feedback for admin/ wardens.

Error Message Display: Display error messages inline when there's a problem storing feedback due to database connectivity issues.

5) Hostel Dashboard Interface

Description: Interface providing specific hostel-related information for admins, wardens, and students.

GUI Standards:Tabs for choosing different hostel dashboards.

Error Message Display: Display error messages in a notification panel if data aggregation fails when loading dashboard information.

2.4 H/w and S/w Requirements

2.4.1 Hardware Requirements

Server-Side:

RAM: Minimum 8 GB, with the capability to scale up as the user base grows.

Hard Disk: Minimum 100 GB or more, depending on data storage needs.

Processor: Minimum 2.5 GHz quad-core or better for efficient processing.

Client-Side:

RAM: Minimum 4 GB or more to handle browser-based interactions smoothly.

Hard Disk: Minimum 20 GB or more to store cache, data, and temporary files.

Processor: Minimum 1.5 GHz dual-core or better for general operations.

2.4.2 Software Requirements

Server-Side:

Operating System (OS): Windows 10 or newer versions recommended.

Web Server: Node.js built-in HTTP for handling backend and web requests.

Database Server: MongoDB for managing relational databases.

Client-Side:

Operating System (OS): Windows 10 or newer versions recommended.

Web Browser: Compatibility with modern web browsers such as:

1. Google Chrome
2. Mozilla Firefox
3. Safari
4. Microsoft Edge

2.4.3 Technologies Used

Front-End Development:

1. React (for component-based UI development)
2. JavaScript (for dynamic interactivity)
3. CSS (for styling)

Back-End Development: Spring Boot (Java Framework)

Database Server: MongoDB

Development Tool: Microsoft Visual Studio Code for designing and coding.

2.4.4 Communication Requirements

Communication Functions: Web Browsers

Communication Standards: HTTP, HTTPS

Server-Side Framework and API: Node.js with its built-in API handles server-side scripting, enabling dynamic content generation and database interactions.

2.5 Feasibility Study

2.5.1 Technical Feasibility

- Technologies Used: The HMS employs web-based technologies such as React, CSS, JavaScript for the front-end, and Springboot with MongoDB for the back-end. These are widely used and support scalability.
- Hardware and Software Requirements:
  - Server: At least 8 GB RAM, 100 GB storage, and a 2.5 GHz processor.
  - Client: At least 4 GB RAM and modern web browsers like Chrome or Firefox.
- Integration: The system uses modular architecture and standard communication protocols (HTTP, HTTPS) for seamless integration.
- Constraints: Compliance with GDPR and FERPA regulations is required.

Conclusion: Technically feasible with no major barriers due to accessible technology stack and infrastructure compatibility.

2.5.2 Operational Feasibility

- Ease of Use: Intuitive UI and role-based features for admin, wardens, and students make the system user-friendly.
- Adoption: Basic English literacy and computer knowledge are required for users, which is reasonable for the target audience.
- Training and Documentation: Comprehensive user guides and support documentation are planned.

Conclusion: Operationally feasible with minimal training requirements.

2.5.3 Economic Feasibility

- Cost Factors:
  - Development: Uses common open-source tools (Node.js, Mongodb).

- Maintenance: Modular design ensures reduced long-term costs.

- Value Added:
  - Automation of room allocation and feedback systems reduces administrative overhead.
  - Centralized data management enhances operational efficiency.

Conclusion: Economically viable due to lower costs associated with open-source tools and reduced administrative effort.

2.6 Product Functions

A hostel management website for a university can offer a wide range of functionalities aimed at streamlining and automating the various administrative and logistical tasks associated with managing student accommodation, it focuses on simplifying the task of managing the hostel. Right now, all the relevant data is being managed in the written form i.e., on pen and paper, which we aim to digitalise.

The Hostel Management System will perform following major functions:-

2.6.1 User Registration and Authentication:

- Admin Registration: Admins can be registered and onboarded by the system super admin.
- Warden Registration: Wardens can be registered and assigned to specific hostels.
- Student Registration: Students can be registered by the warden or self-register with approval from the warden. They can also be assigned to specific rooms during registration.

2.6.2 Room Allocation Process:

- Allocate Rooms: The warden can assign rooms to students based on availability. The warden can view the current status of all rooms (occupied, available, under maintenance).

- Update Room Assignments: The warden can update room assignments, such as reallocating rooms when students move out or switch rooms.
- View Room Status: The student view the details about their allocated room.

2.6.3 Notice Board Notifications:

- Post Notices: The warden can post notifications on the digital notice board, including important updates, events, and announcements.
- Update Notices: The warden can update or remove notices as needed.
- View Notices: Students can view the latest notices and stay informed about hostel-related updates.

2.6.4 Student Feedback:

- Submit Feedback: Students can submit feedback about the hostel facilities, services, or any other concerns they have.
- View Feedback: The warden can view all feedback submitted by students to identify areas for improvement.

2.6.5 Student Records Management:

- Store Student Records: The system stores detailed records for each student, including personal information, room assignment, contact details.
- View Student Records: The admin can view the consolidated student records for all hostels, allowing for centralized oversight.

2.6.6 Administrative Functions:

- View Student Records: The admin has access to view the student records for all hostels, enabling effective monitoring and management.
- View Feedback: The admin can view all feedback provided by students across all hostels, facilitating the identification of common issues and the implementation of improvements.

2.7 Use-case Diagram

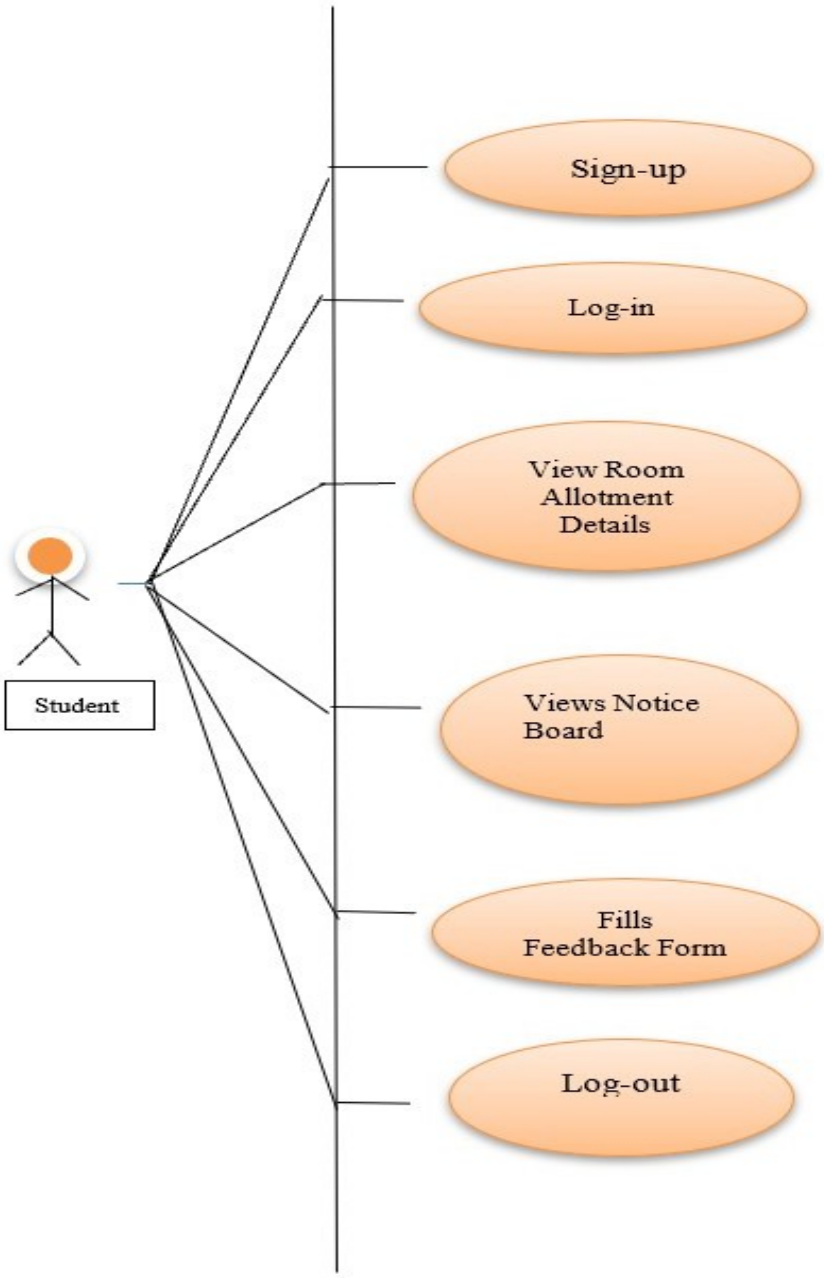
There are three users who can use this product include administrator, wardens , and the students. Each user is required to have a basic knowledge of compute and net browser. Each user should have basic knowledge of English.

1. Administrators: The admin oversee system operations and ensure compliance. Approve the users and maintain the registers of each user.

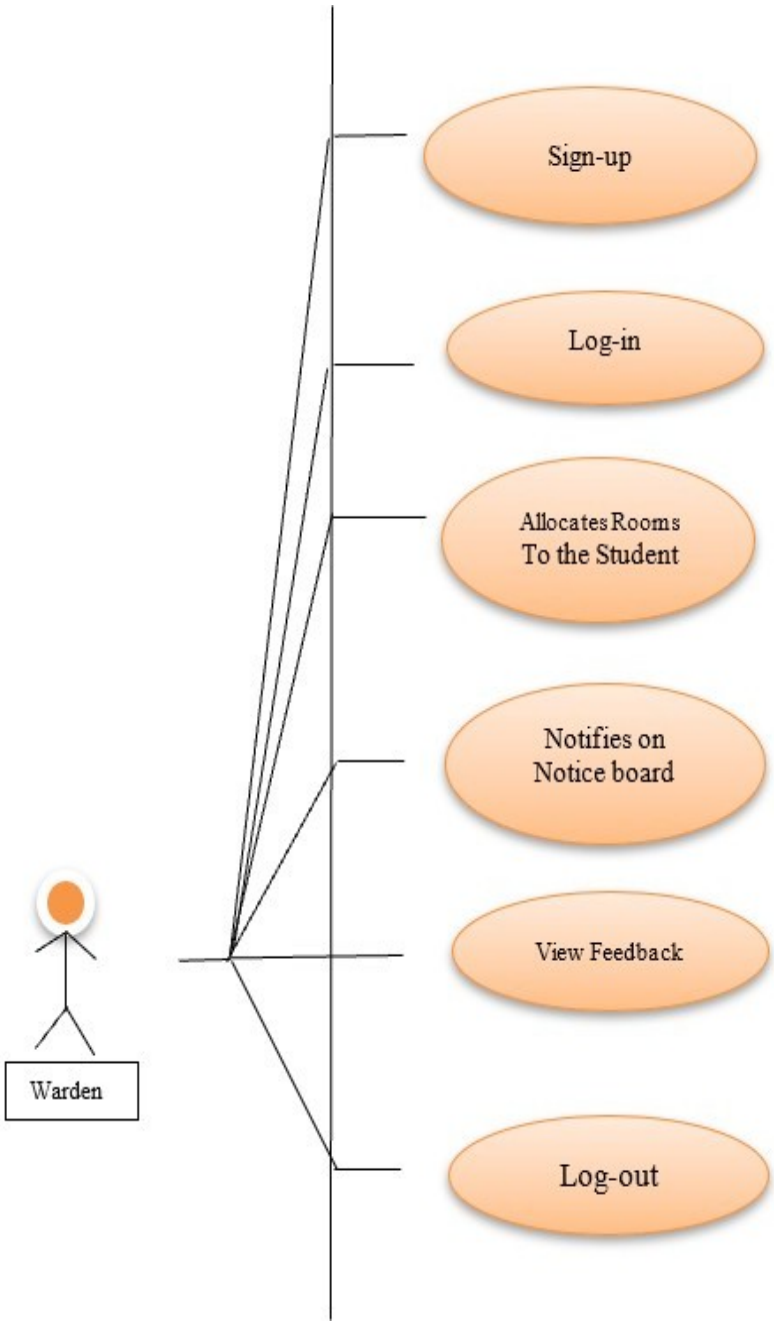
2. Wardens: The warden can allocate rooms to the students. They can also notify the students with important information. They can view the necessary details about the hostel rooms (total capacity, availability etc).

3. Students: The students can view the required information about their allocated room. They can also provide feedback regarding the hostel services.

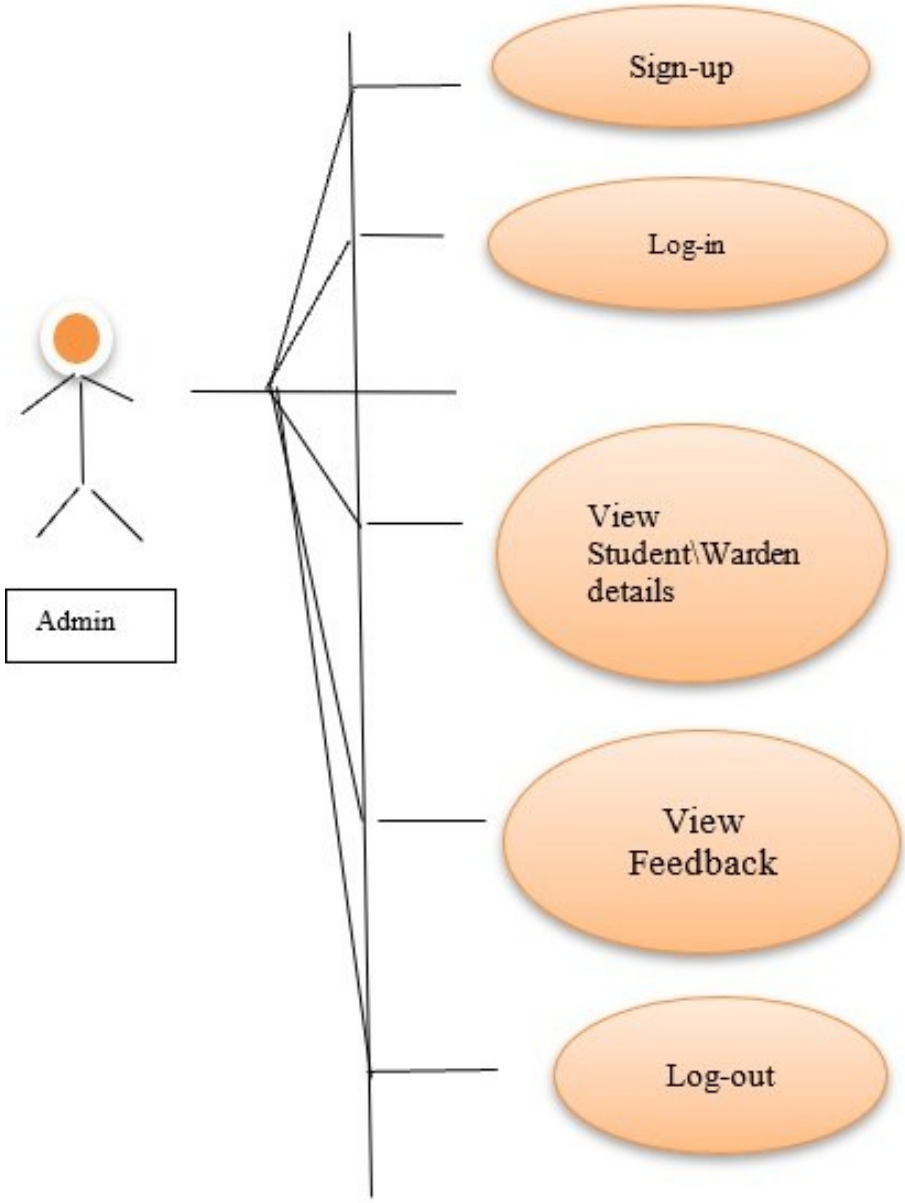
2.7.1 Use-case Diagram for Student:



2.7.2 Use-case Diagram for Warden:

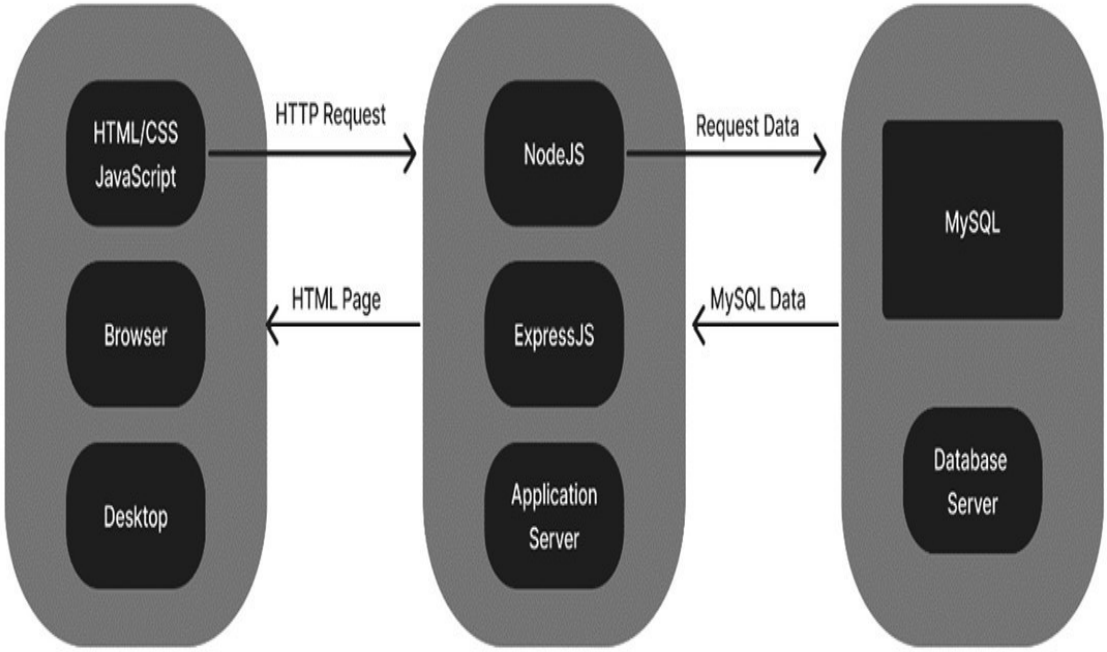


2.7.3 Use-case Diagram for Admin:

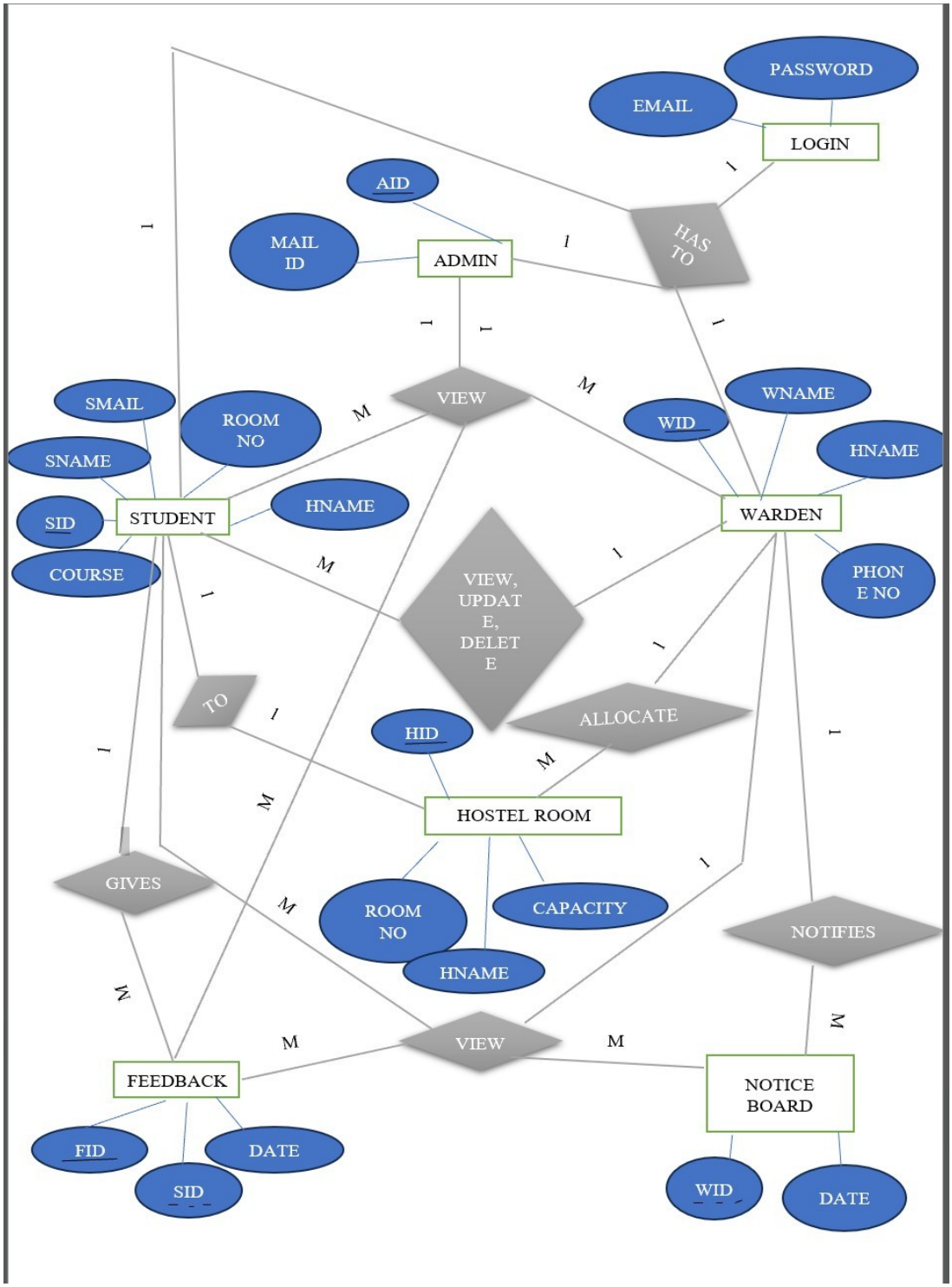


System Design (SDS)

3.1 High-level Diagram



3.2 ER Diagram



3.3 Database Design

3.3.1 Users Table (For Login/Registration):

Attribute	Type	Constraints
user_id	INT	PRIMARY KEY, AUTO_INCREMENT
first_name	VARCHAR(100)	NOT NULL
Last_name	VARCHAR(100)	NOT NULL
email	VARCHAR	NOTNULL, UNIQUE
Password	VARCHAR	NOT NULL
role	ENUM('admin', 'student', 'warden')	NOT NULL

3.3.2 Room Allocation Table:

Attribute	Type	Constraints
allocation_id	INT (PK)	PRIMARY KEY, AUTO_INCREMENT
Hostel_name	VARCHAR(100)	NOT NULL
Course_name	VARCHAR(100)	NOT NULL
student_id	INT	FOREIGN KEY REFERENCES Student(student_id), NOT NULL
student_name	VARCHAR(100)	NOT NULL
Room_number	INT	NOT NULL

3.3.3 Notice Board Table:

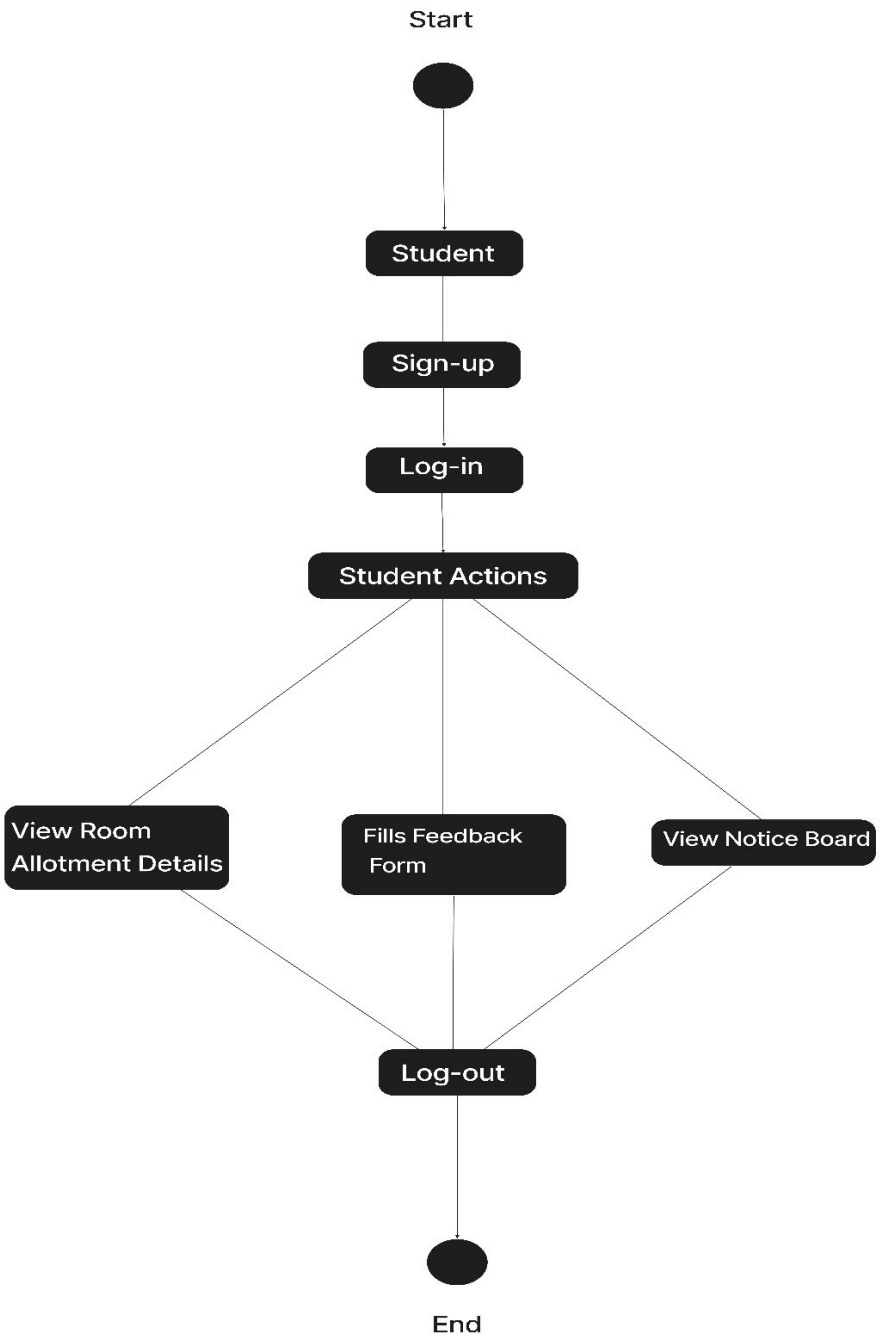
Attribute	Type	Constraints
notice_id	INT (PK)	PRIMARY KEY, UNIQUE, NOT NULL, AUTO_INCREMENT
title	VARCHAR(255)	NOT NULL
notice	TEXT	NOT NULL

3.3.4 Feedback Table:

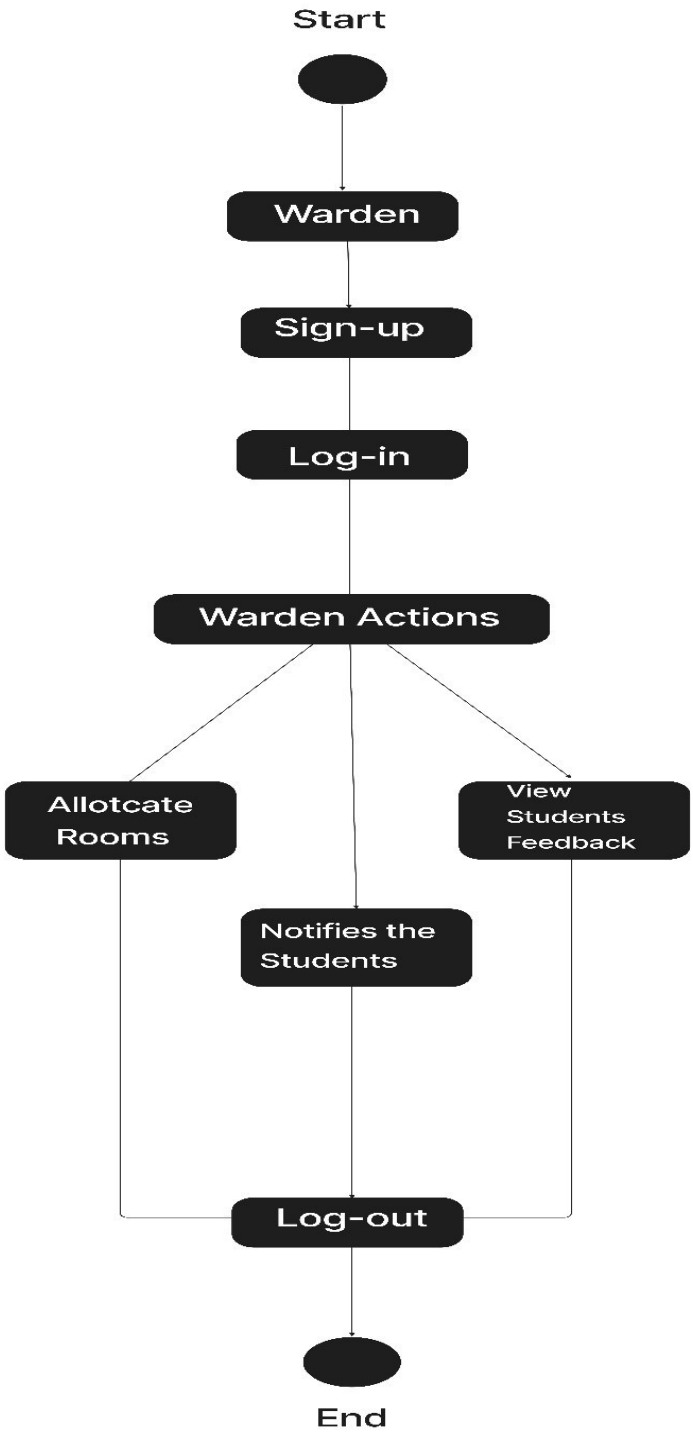
Attribute	Type	Constraints
feedback_id	INT	PRIMARY KEY, AUTO_INCREMENT
full_name	VARCHAR(100)	NOT NULL
email	VARCHAR(100)	NOT NULL
hostel_name	VARCHAR(100)	NOT NULL
cleanliness_rating	ENUM('Excellent', 'Very Good', 'Good', 'Average', 'Below Average')	NOT NULL
food_rating	ENUM('Excellent', 'Very Good', 'Good', 'Average', 'Below Average')	NOT NULL
facilities_rating	ENUM('Excellent', 'Very Good', 'Good', 'Average', 'Below Average')	NOT NULL
staff_rating	ENUM('Excellent', 'Very Good', 'Good', 'Average', 'Below Average')	NOT NULL
safety_rating	ENUM('Excellent', 'Very Good', 'Good', 'Average', 'Below Average')	NOT NULL

3.4 Activity Diagram

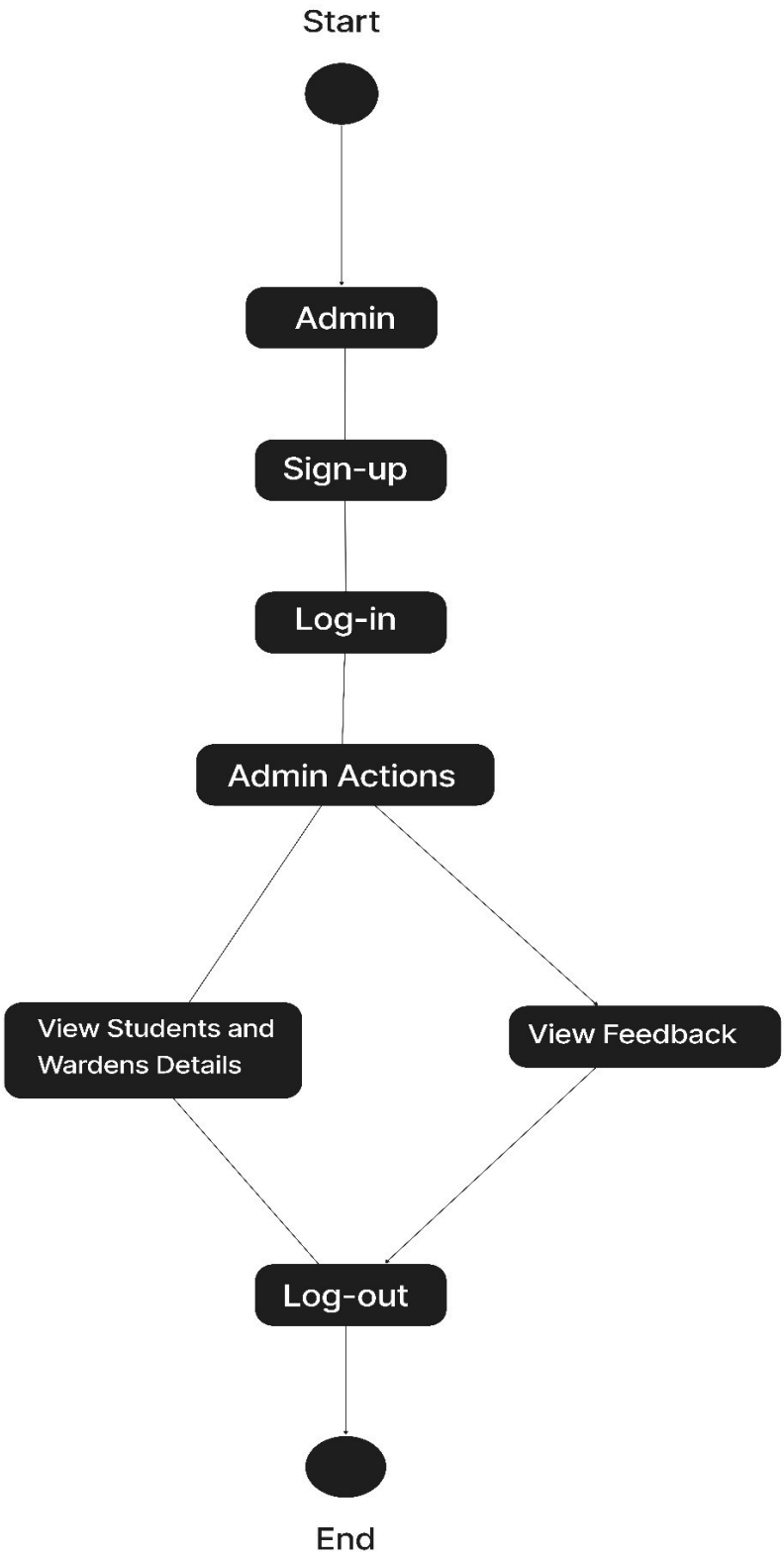
3.4.1 Activity Diagram for Student:



3.4.2 Activity Diagram for Warden:

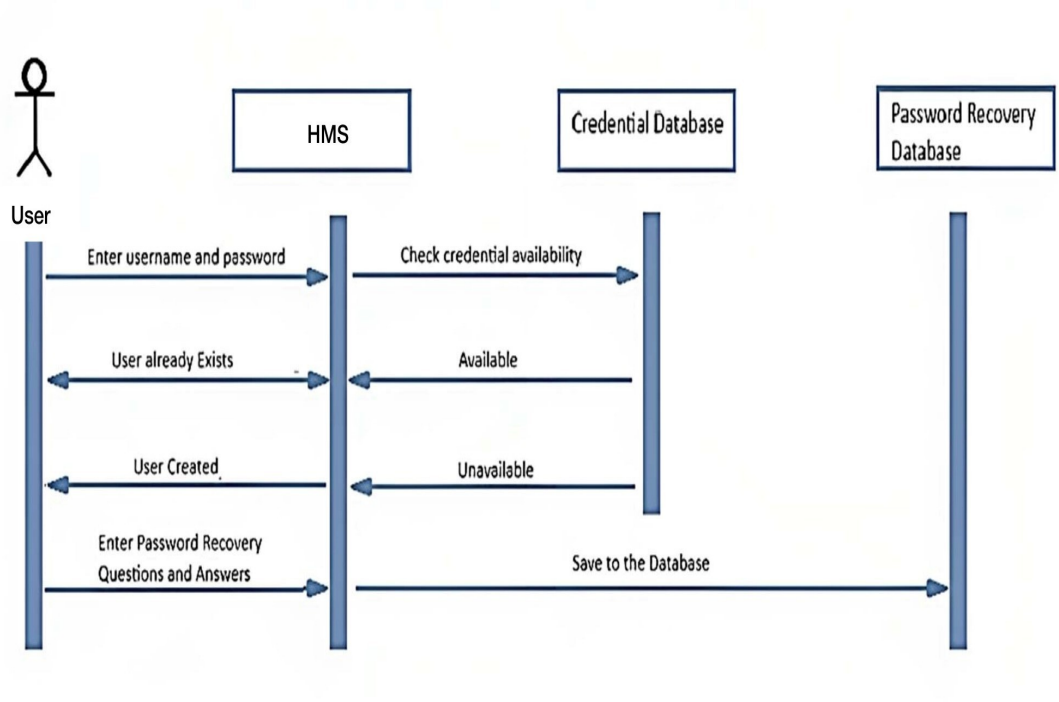


3.4.3 Activity Diagram for Admin:



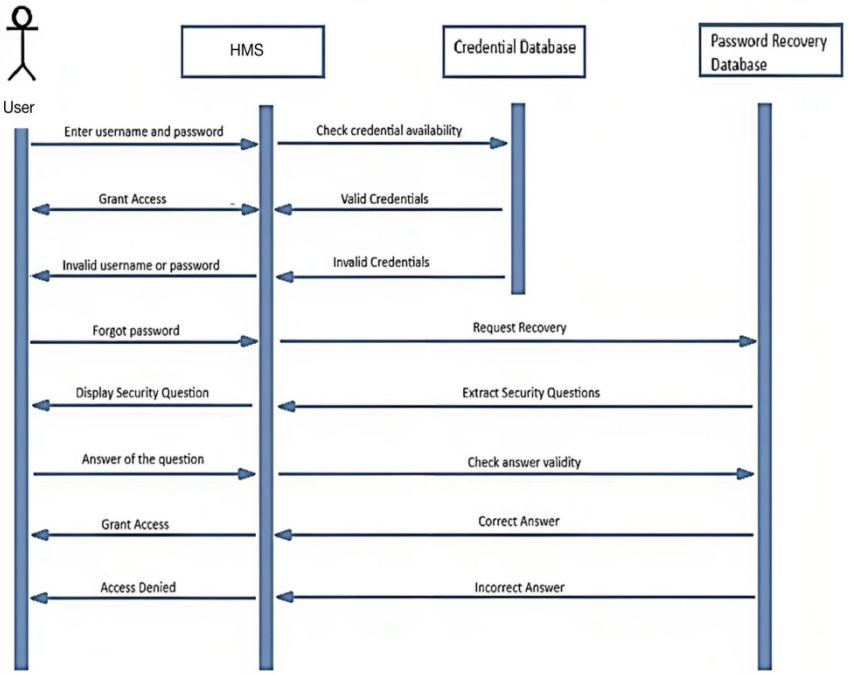
3.5 Sequence Diagram

3.5.1 Sign up:

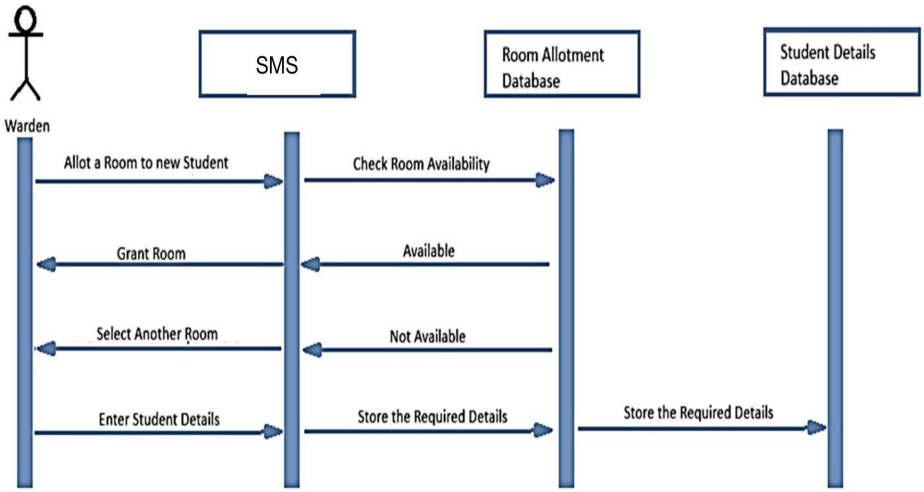


3.5.2 Log In & Password Recovery:

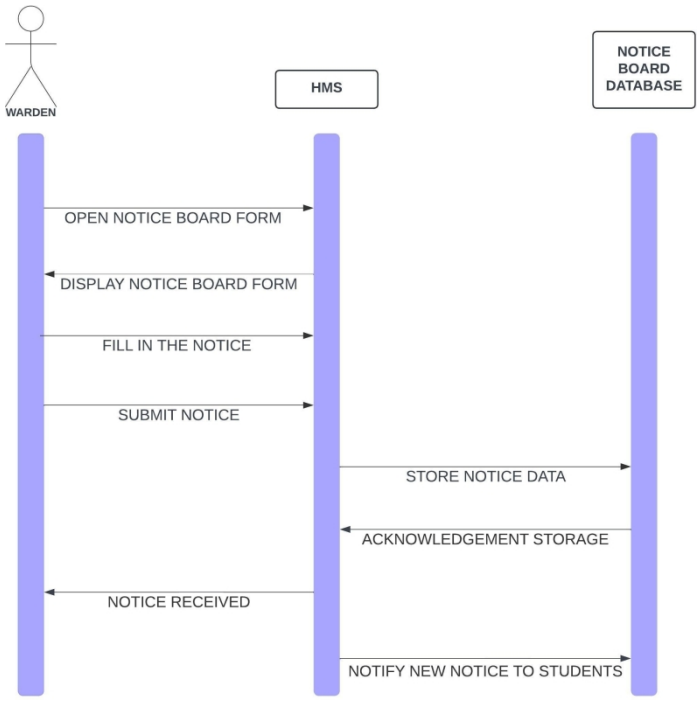
\



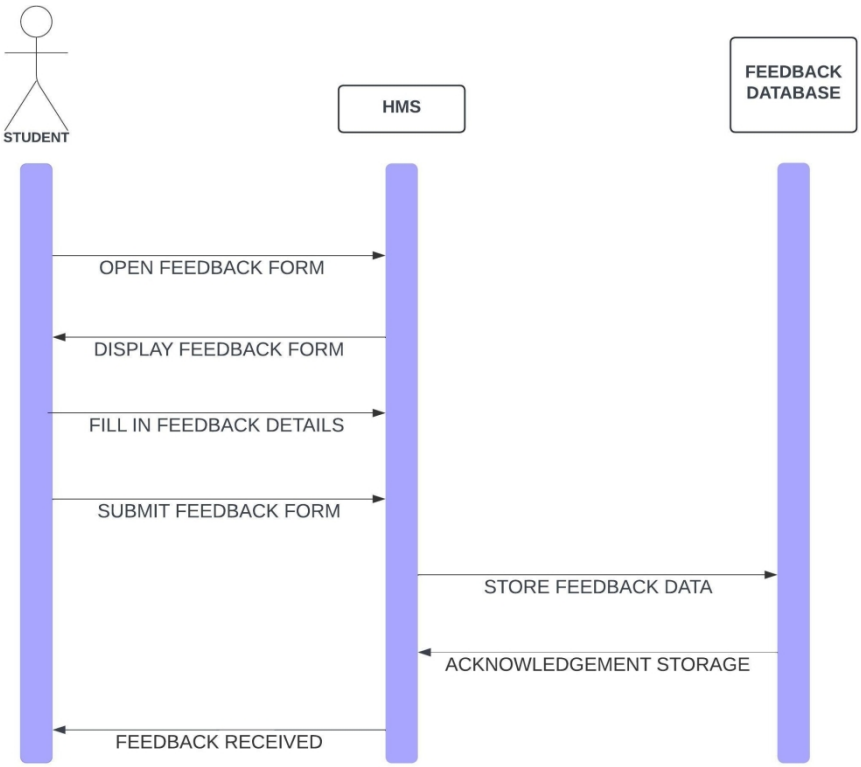
3.5.3 Room Allotment:



3.5.4 Notice Board:



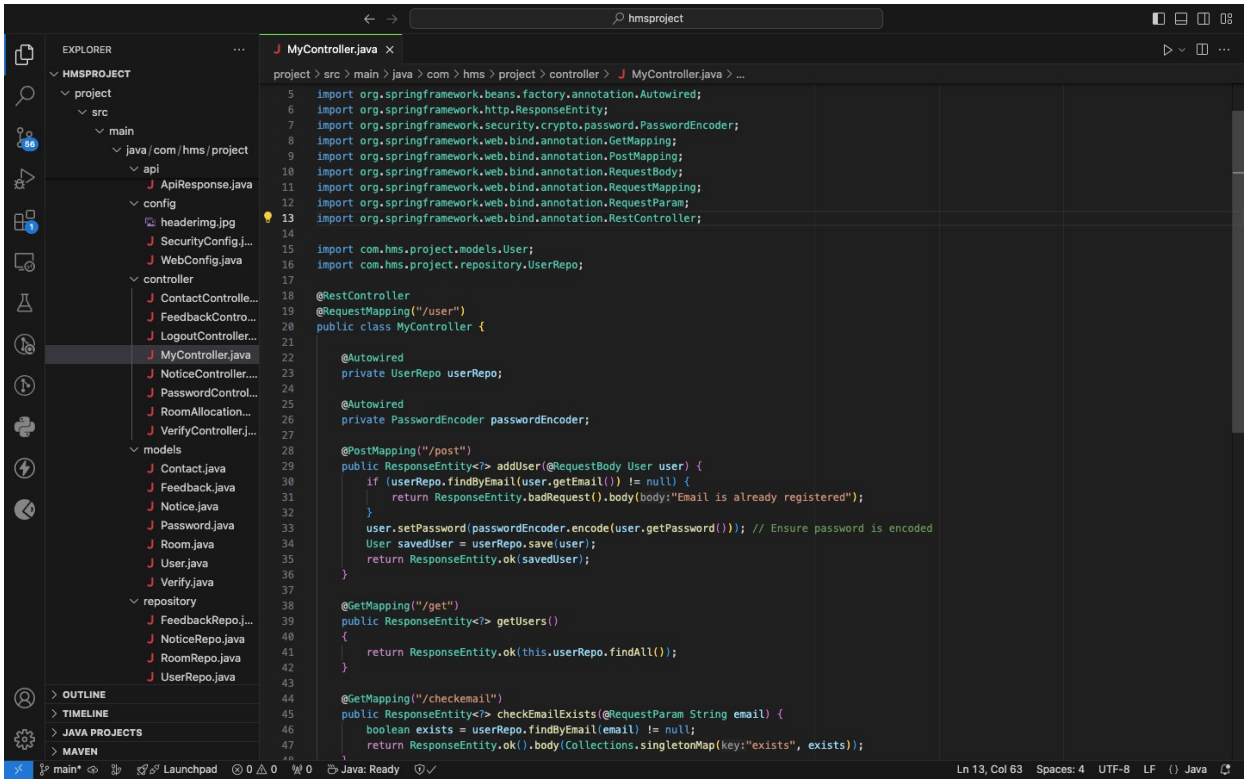
3.5.5 Feedback:



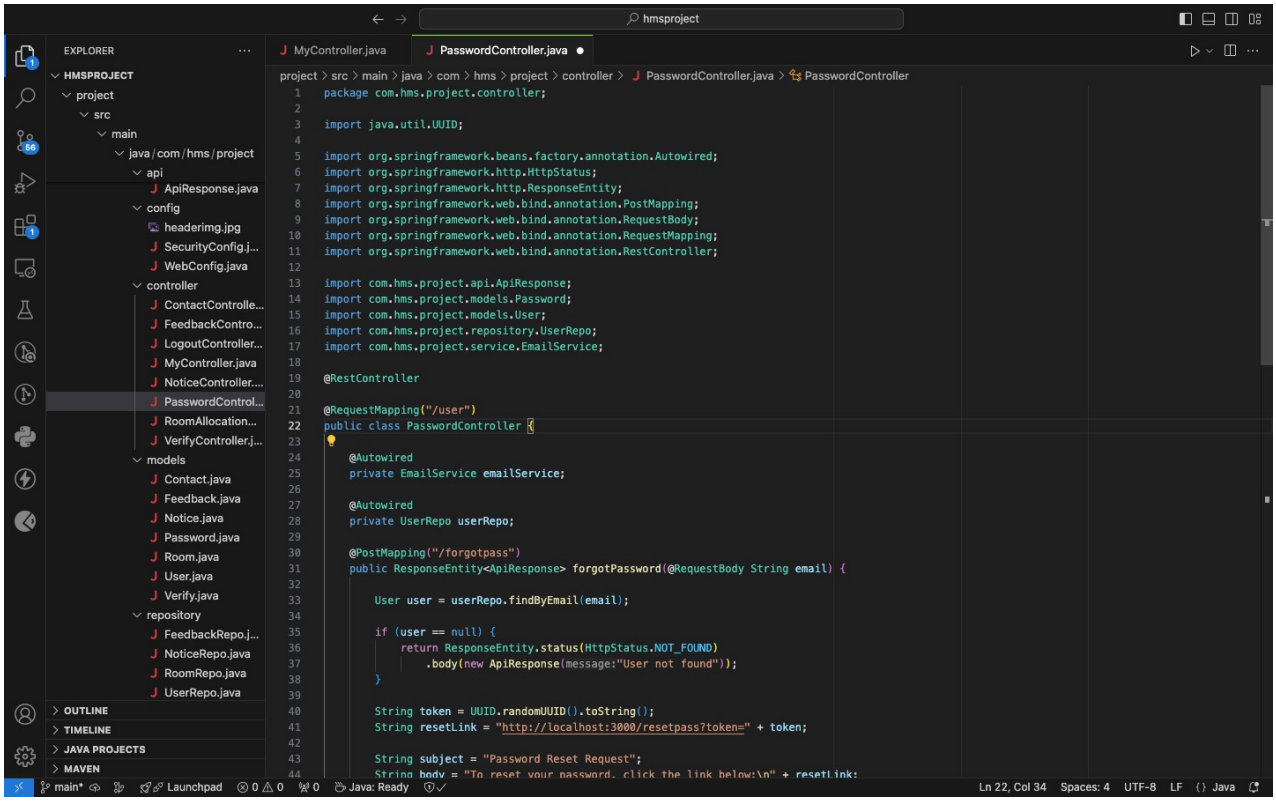
Coding

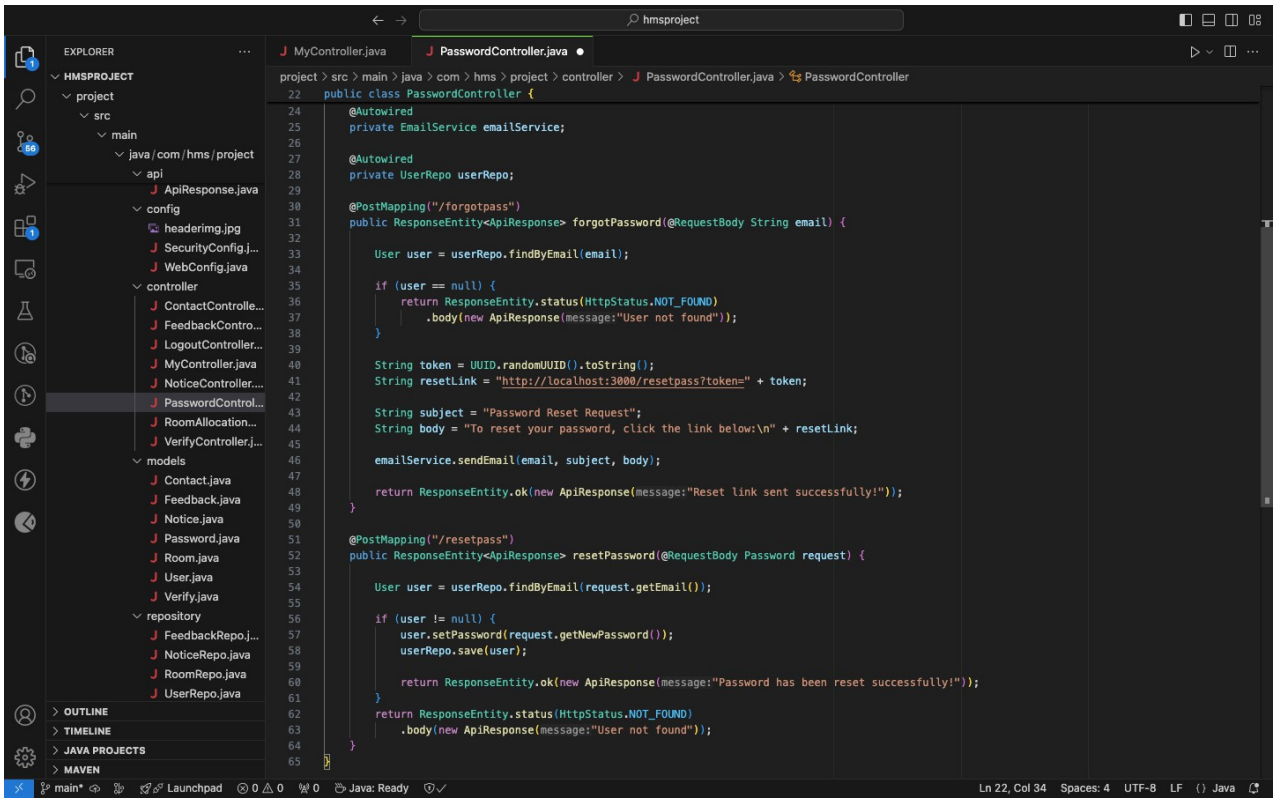
4.1 Backend

4.1.1 Controllers

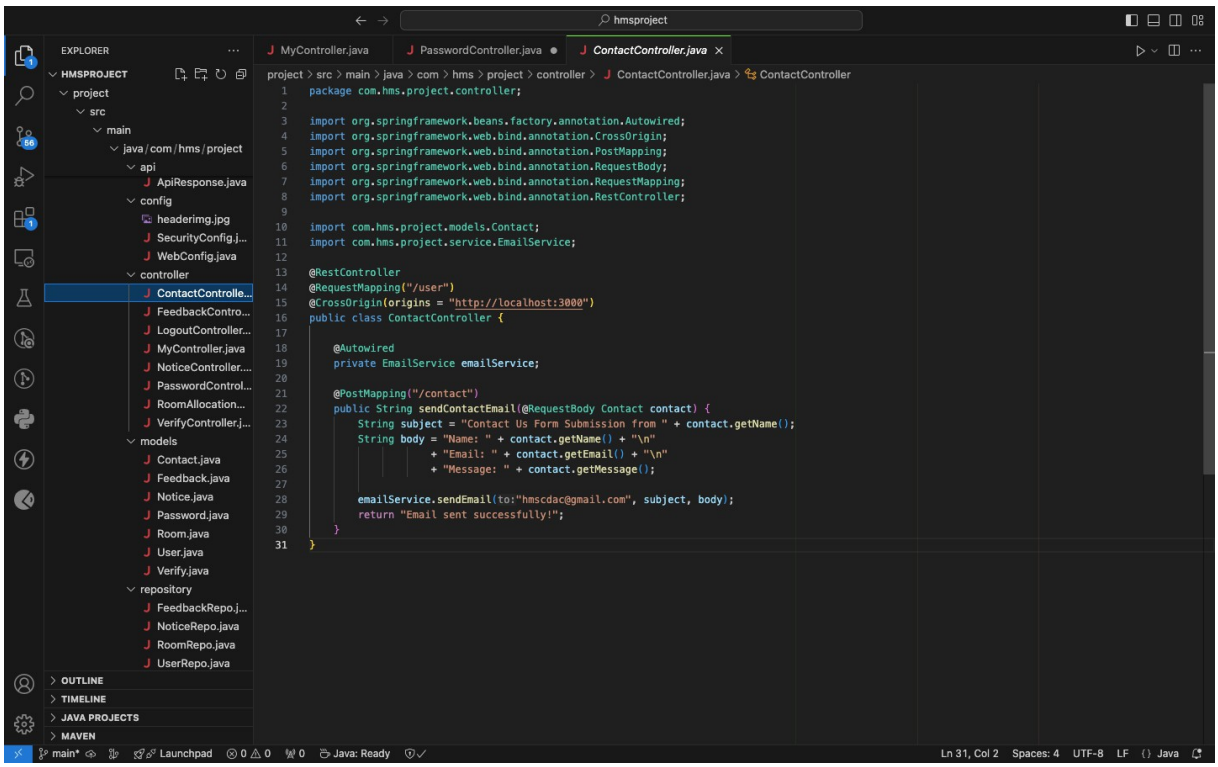


MyController.java

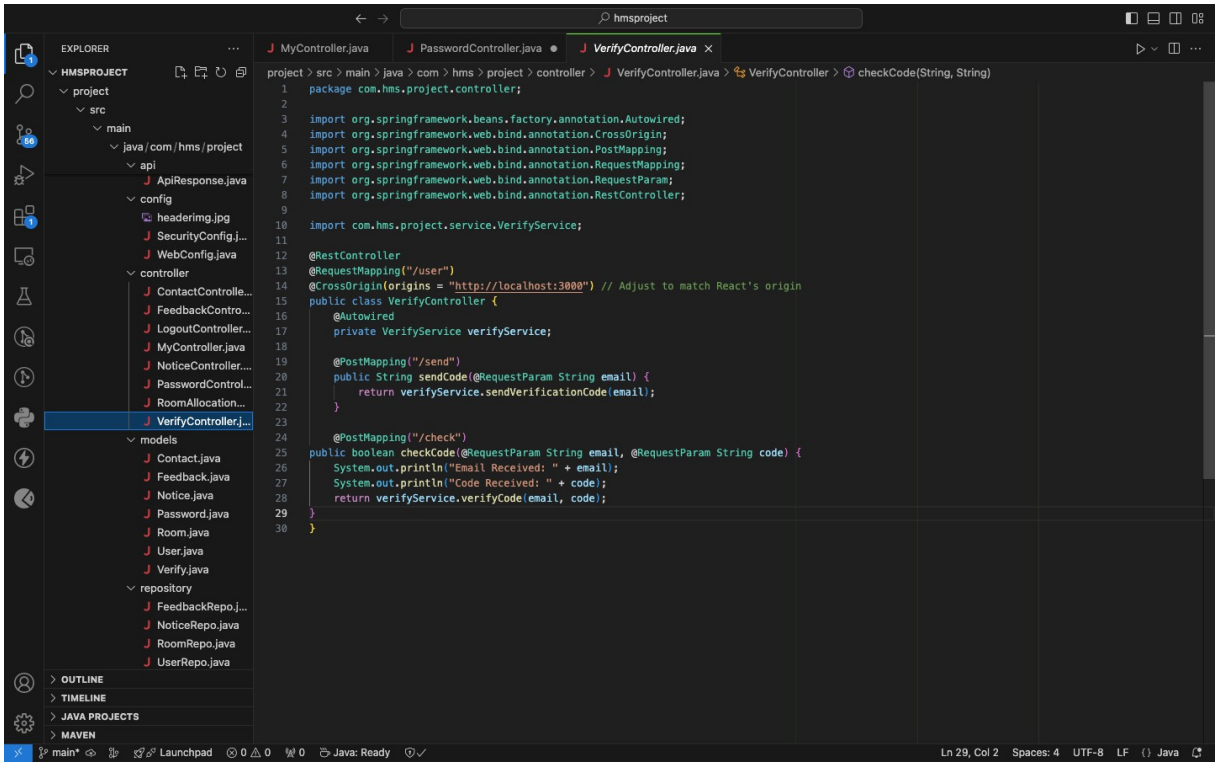




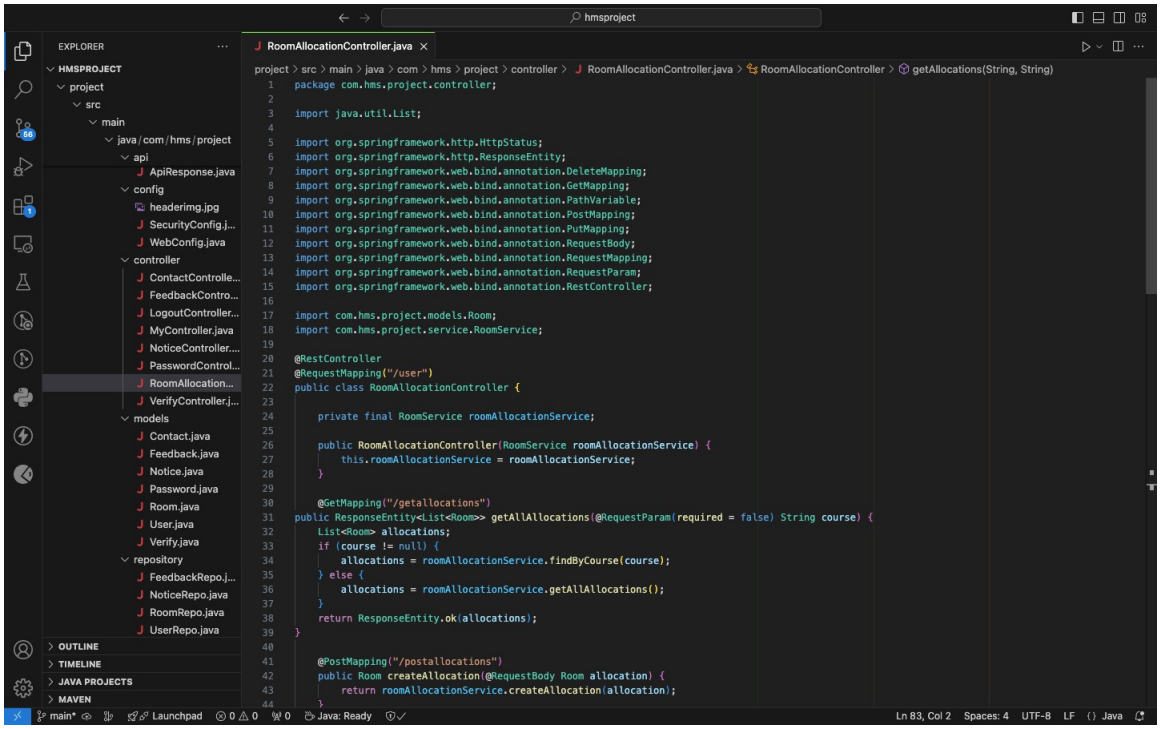
PasswordController.java

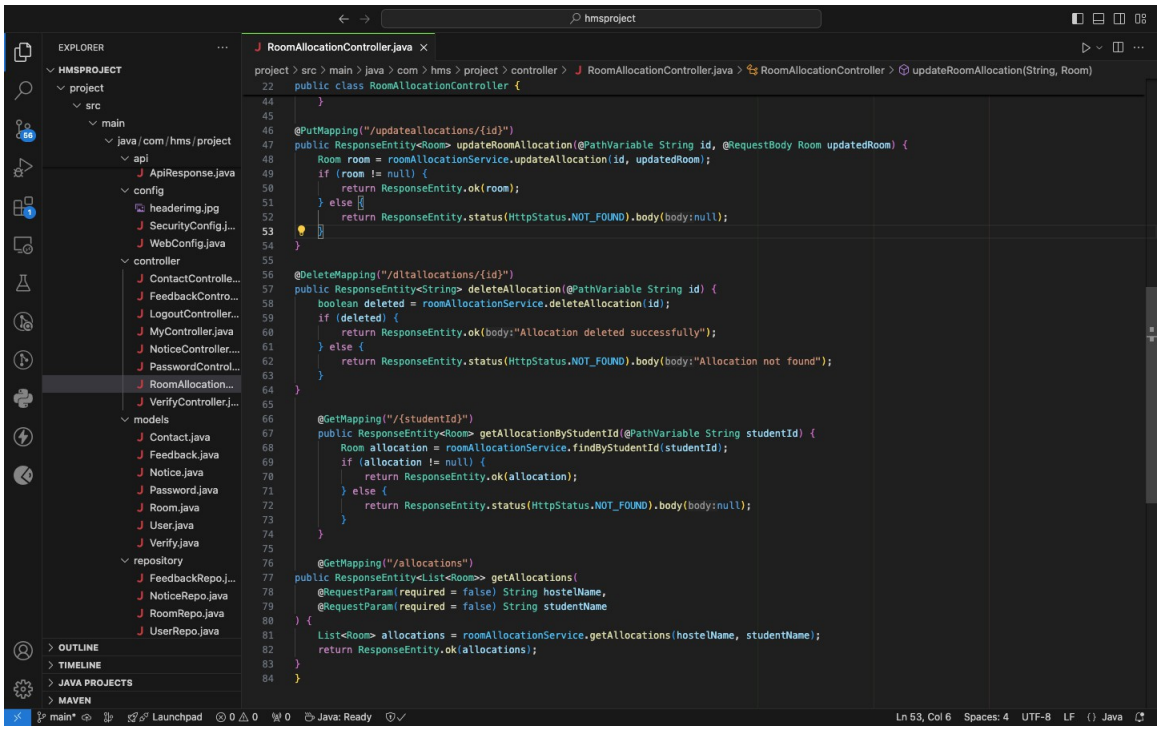


ContactController.java

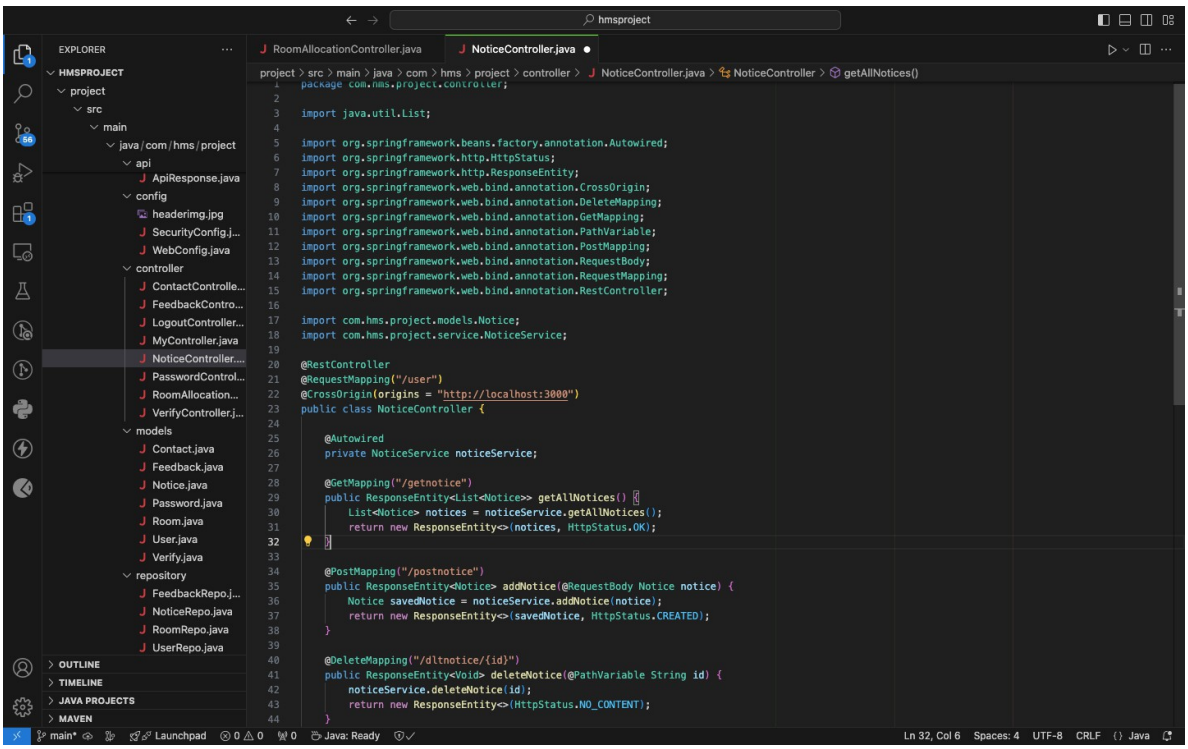


VerifyController.java

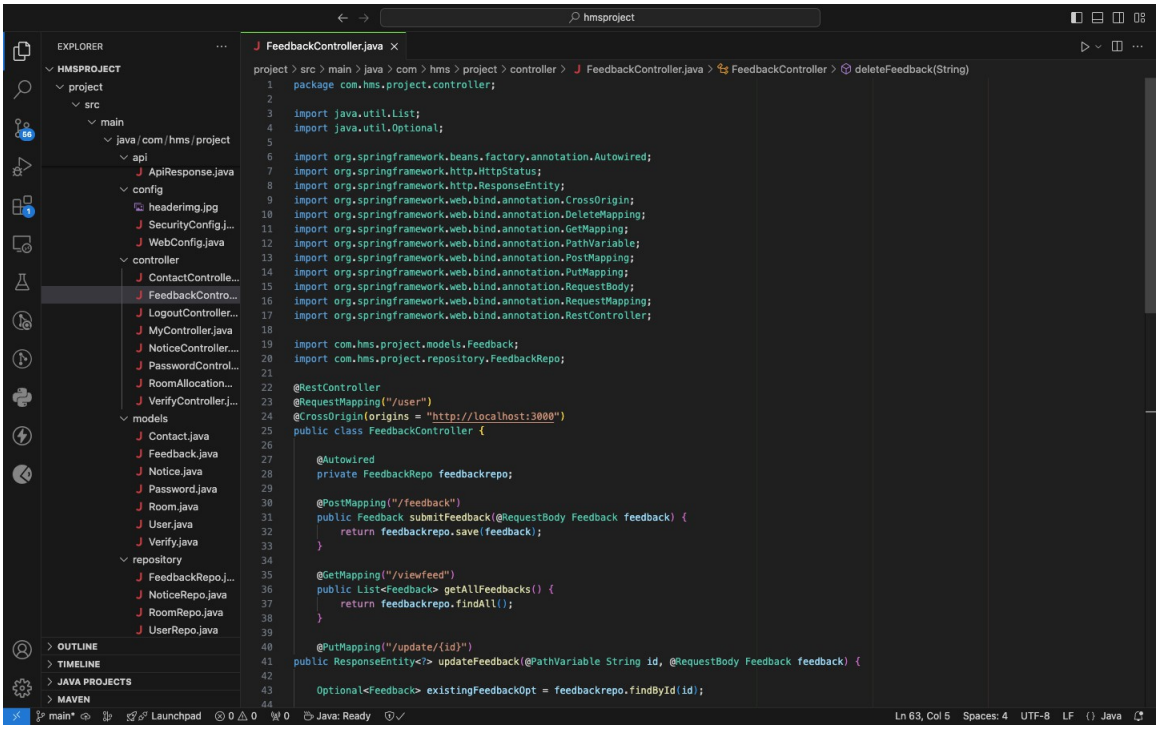


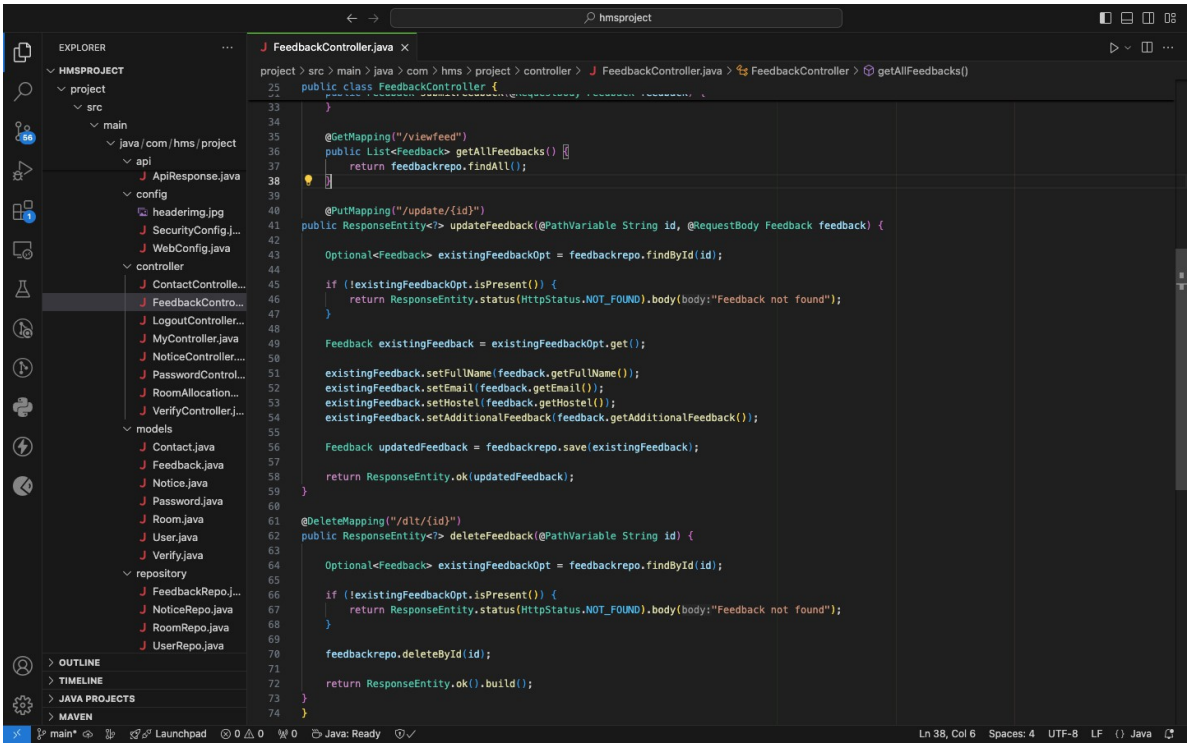


RoomController.java

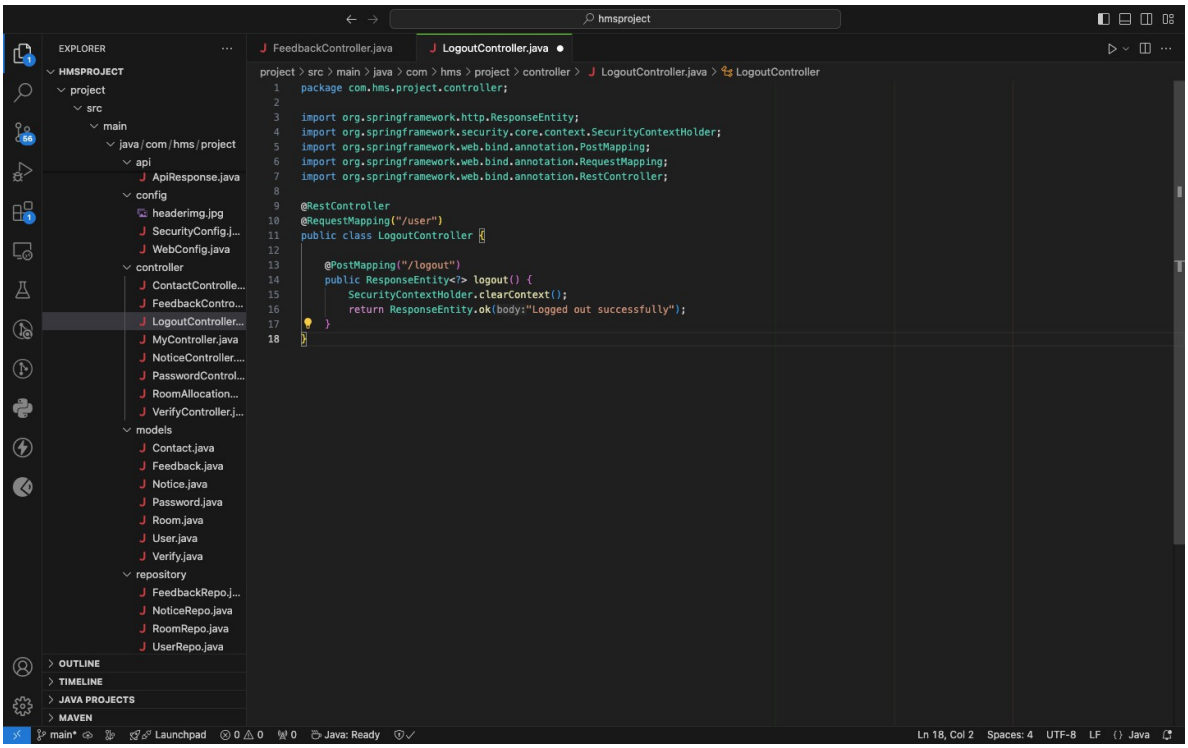


NoticeController.java



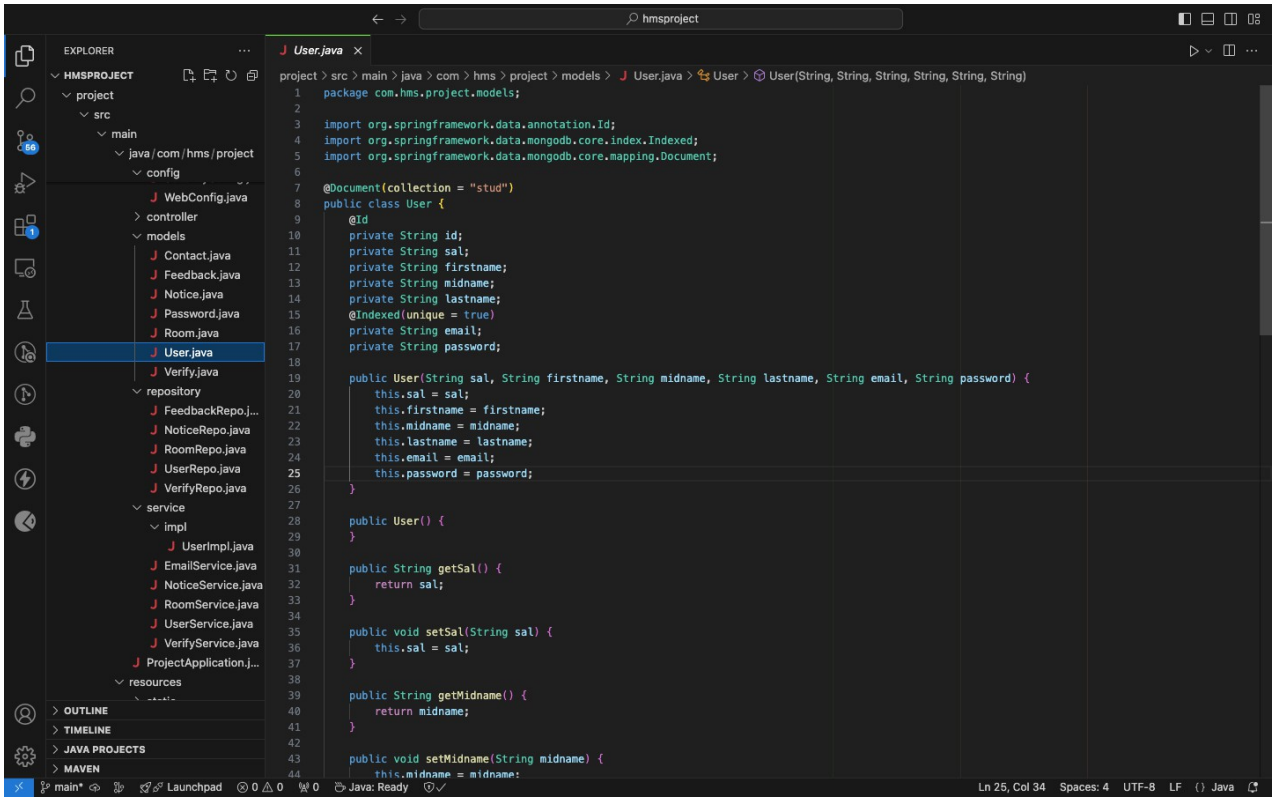


FeedbackController.java

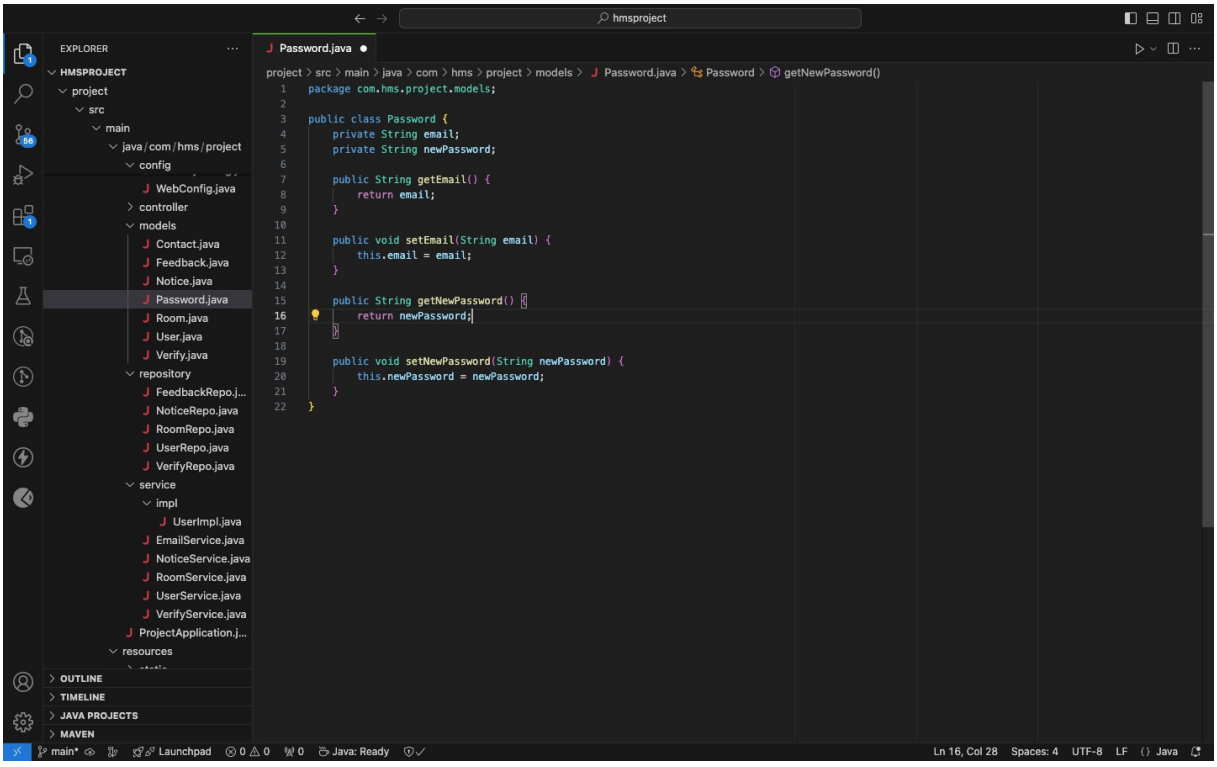


LogoutController.java

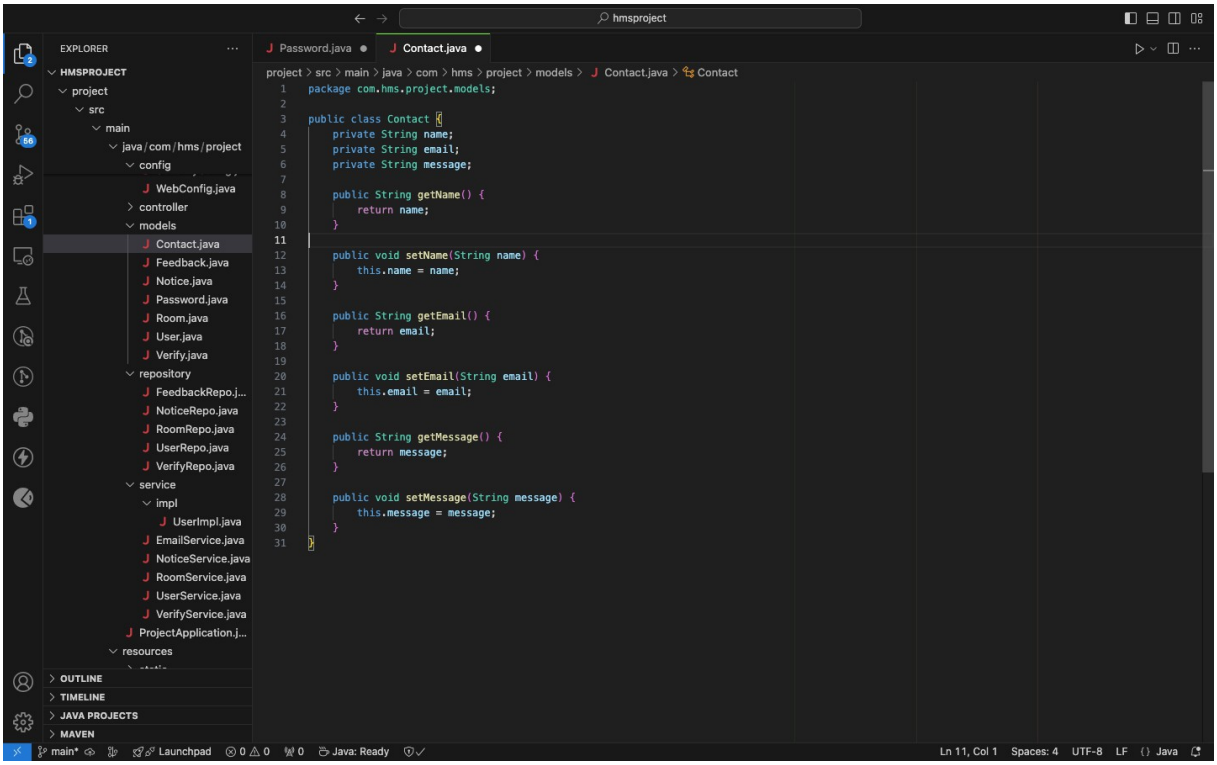
4.1.2 Models



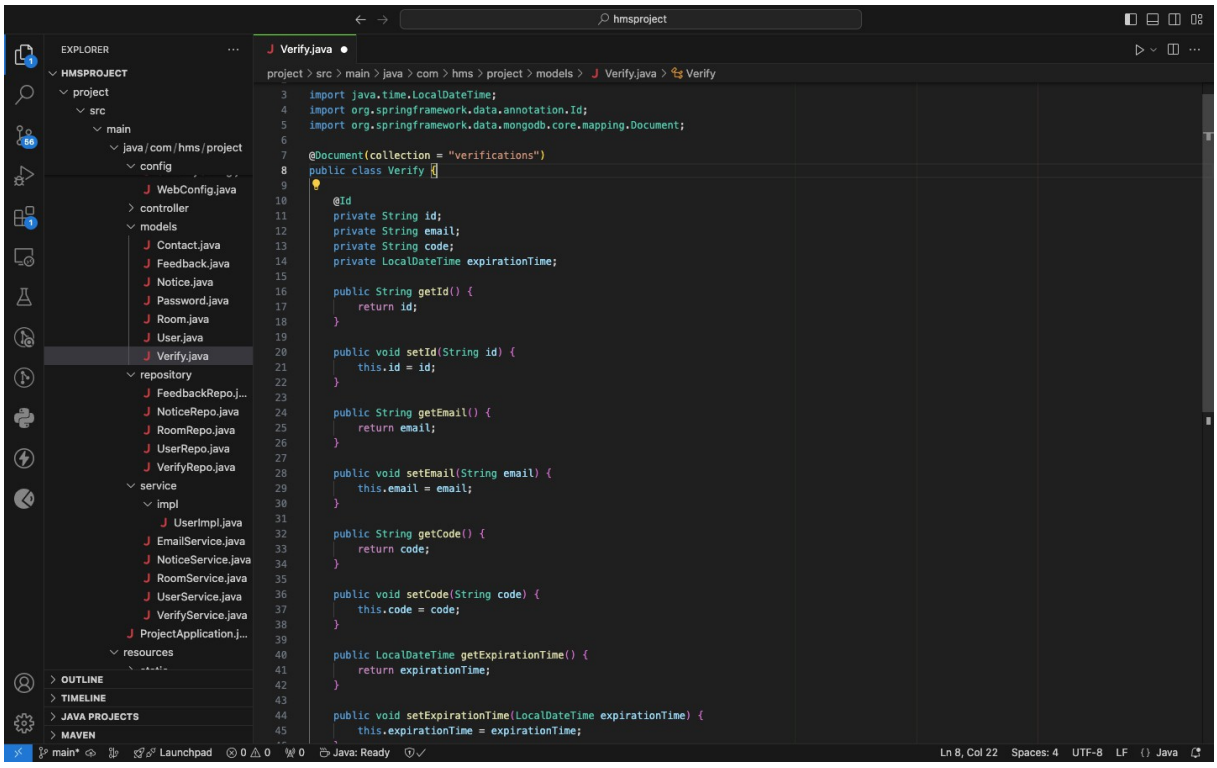
User.java



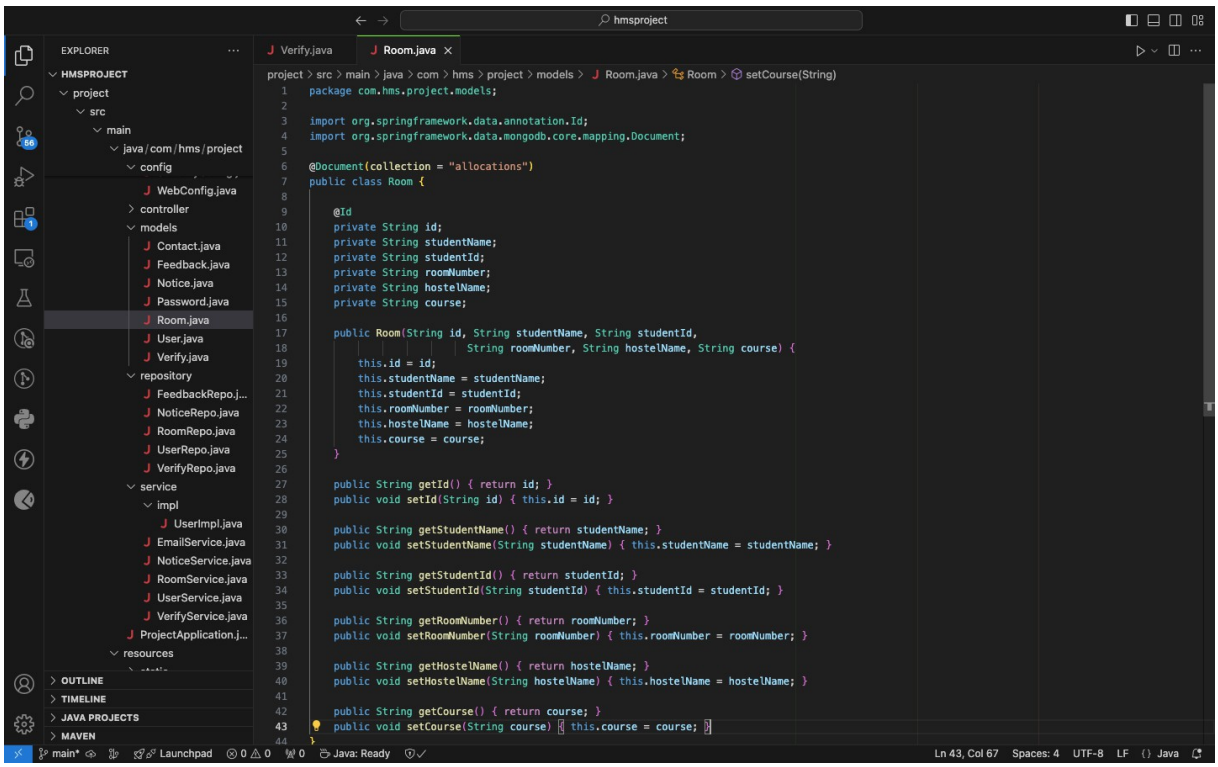
Password.java



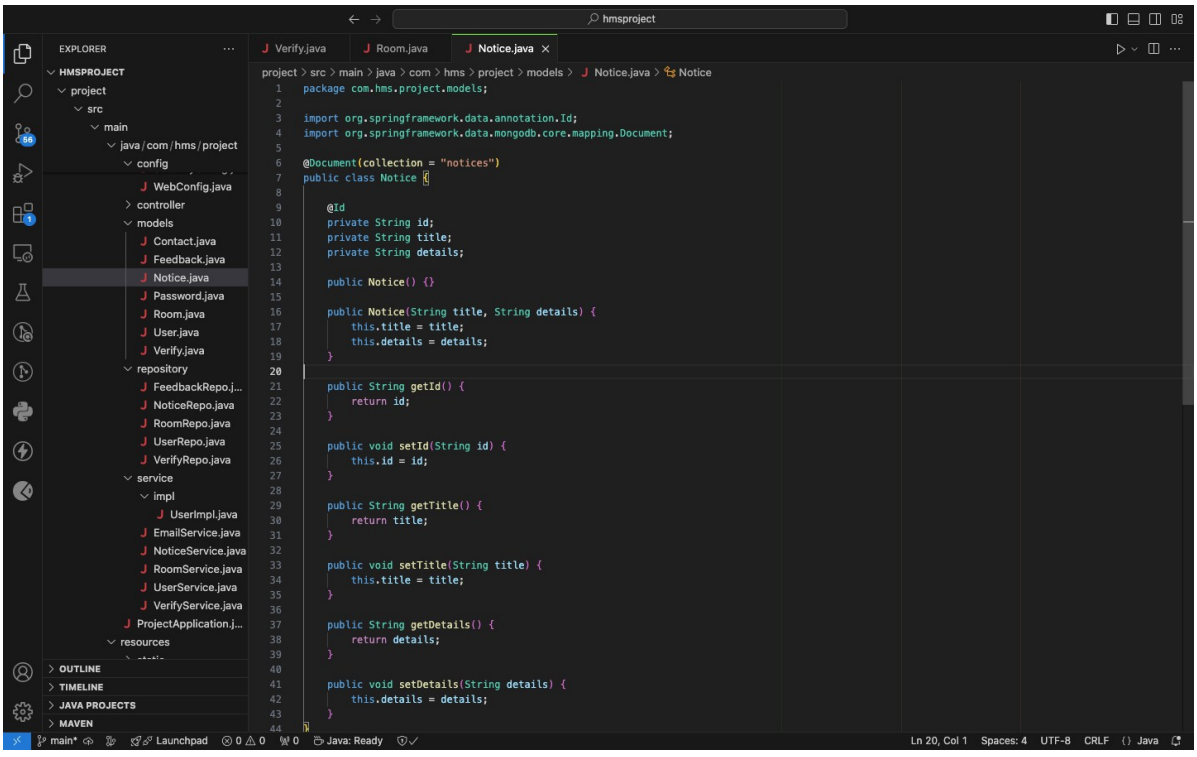
Contact.java



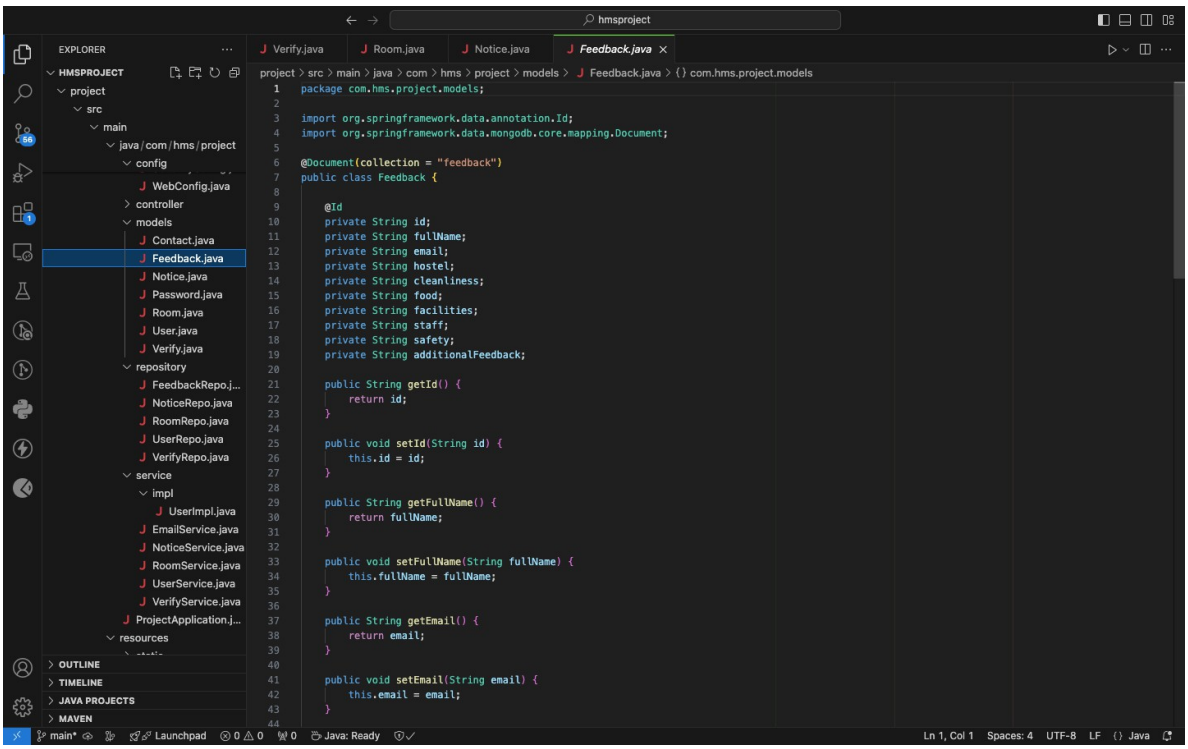
Verify.java



Room.java

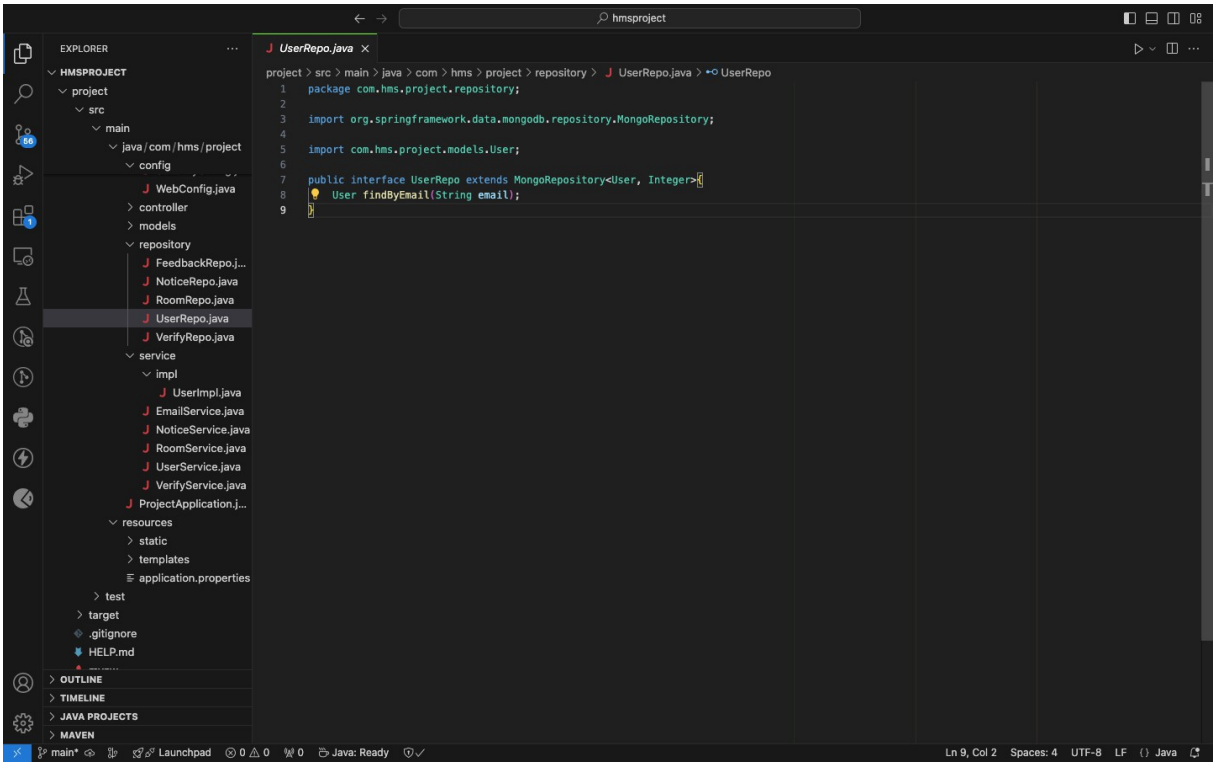


Notice.java

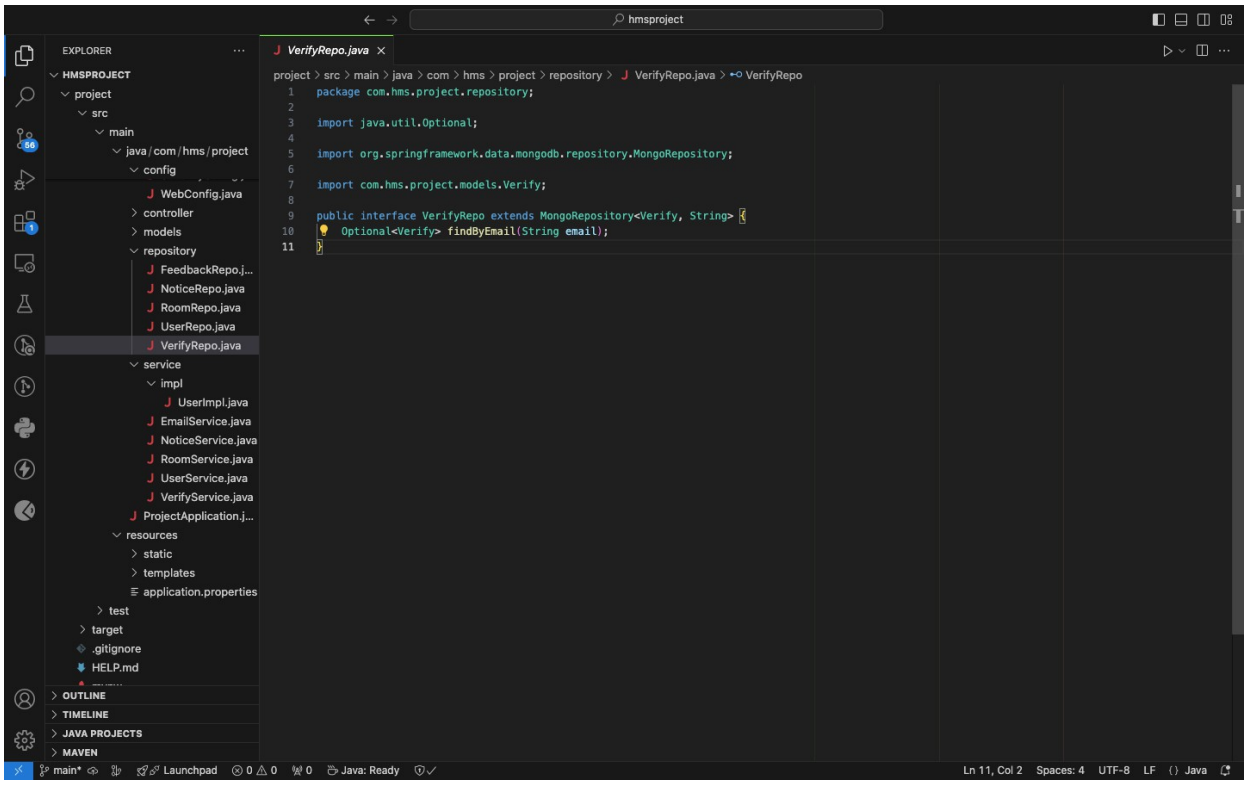


Feedback.java

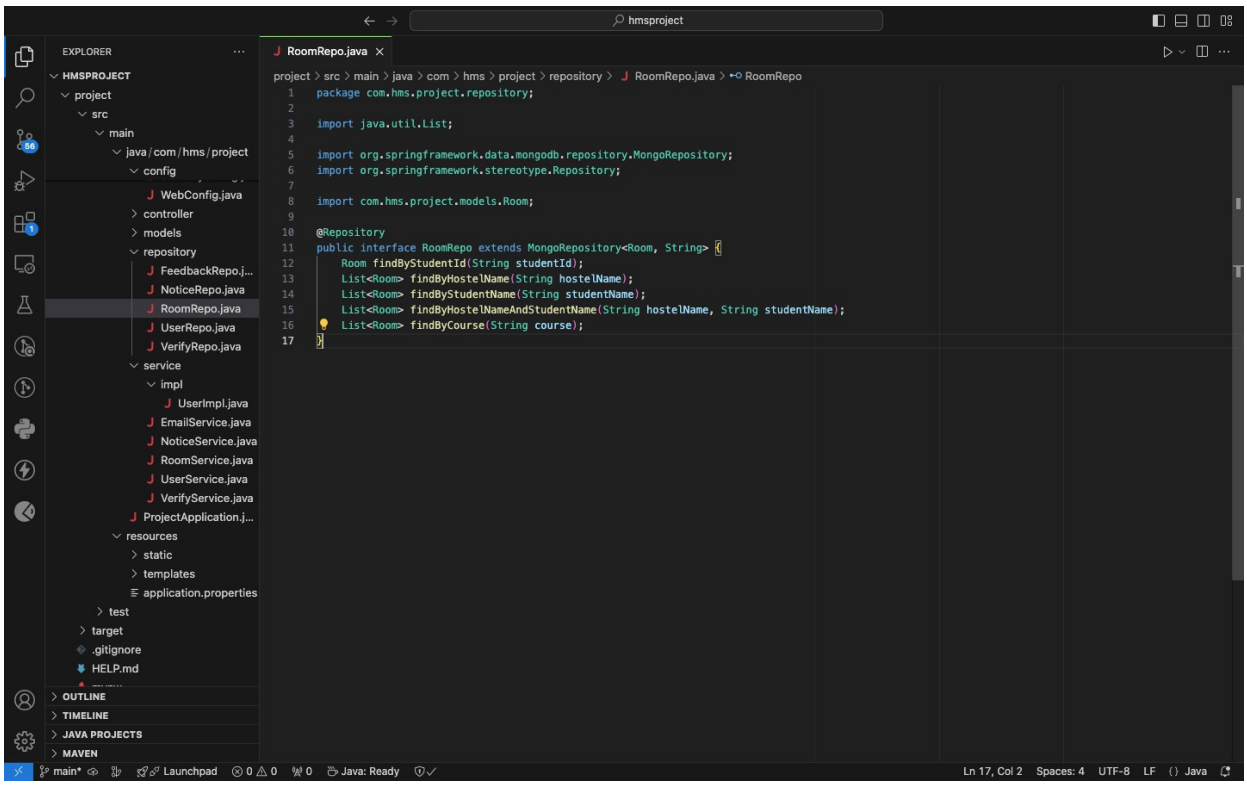
4.1.3 Repository



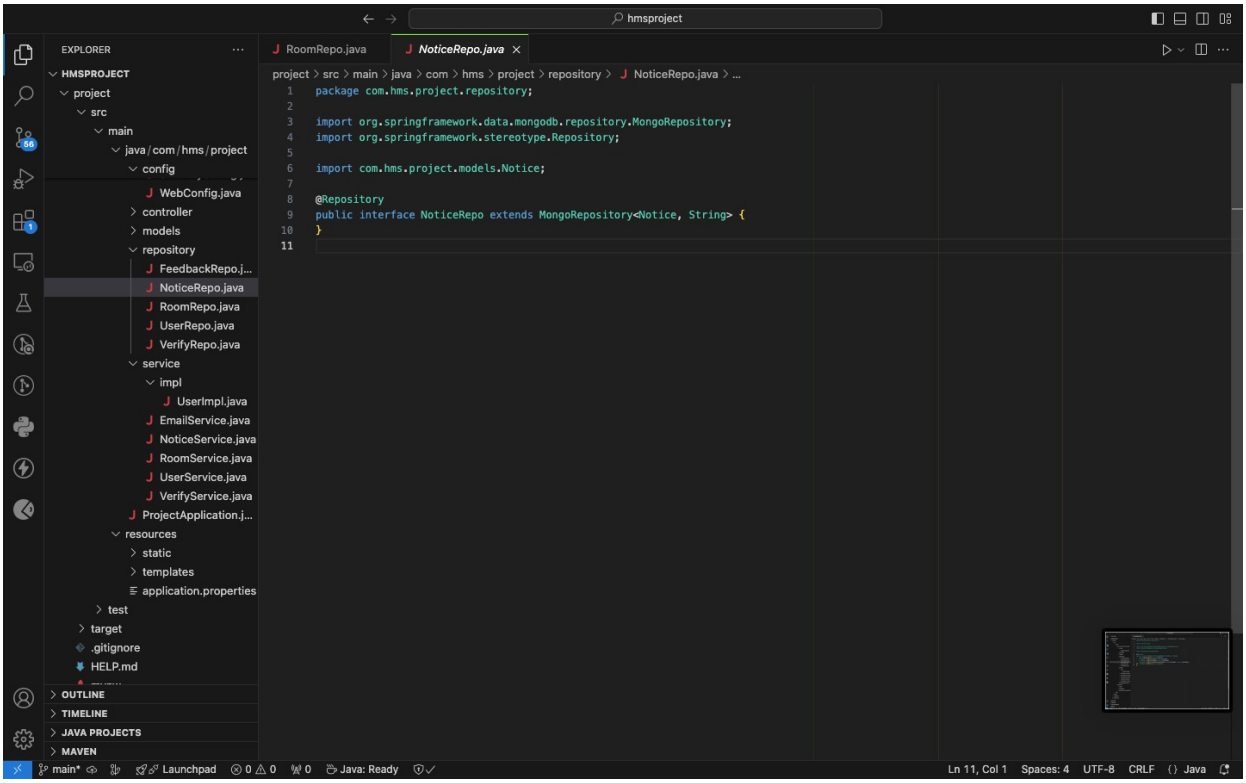
UserRepo.java



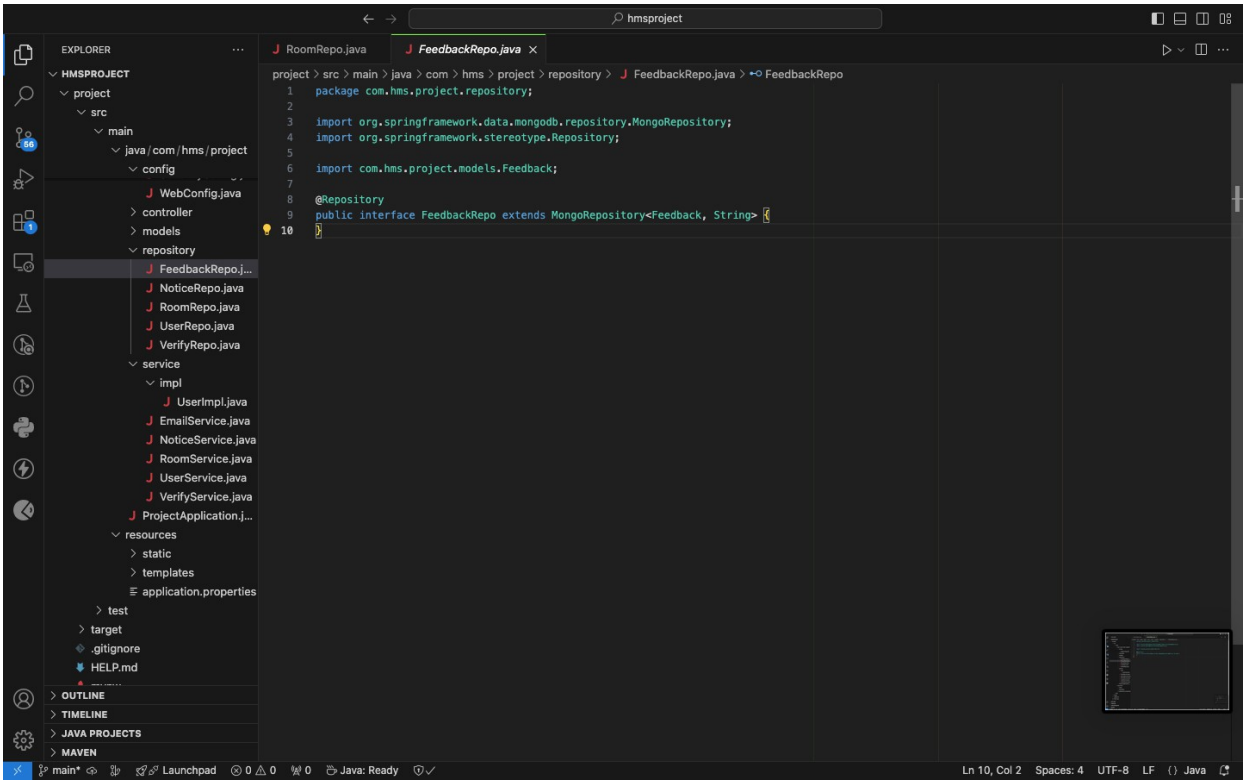
VerifyRepo.java



RoomRepo.java

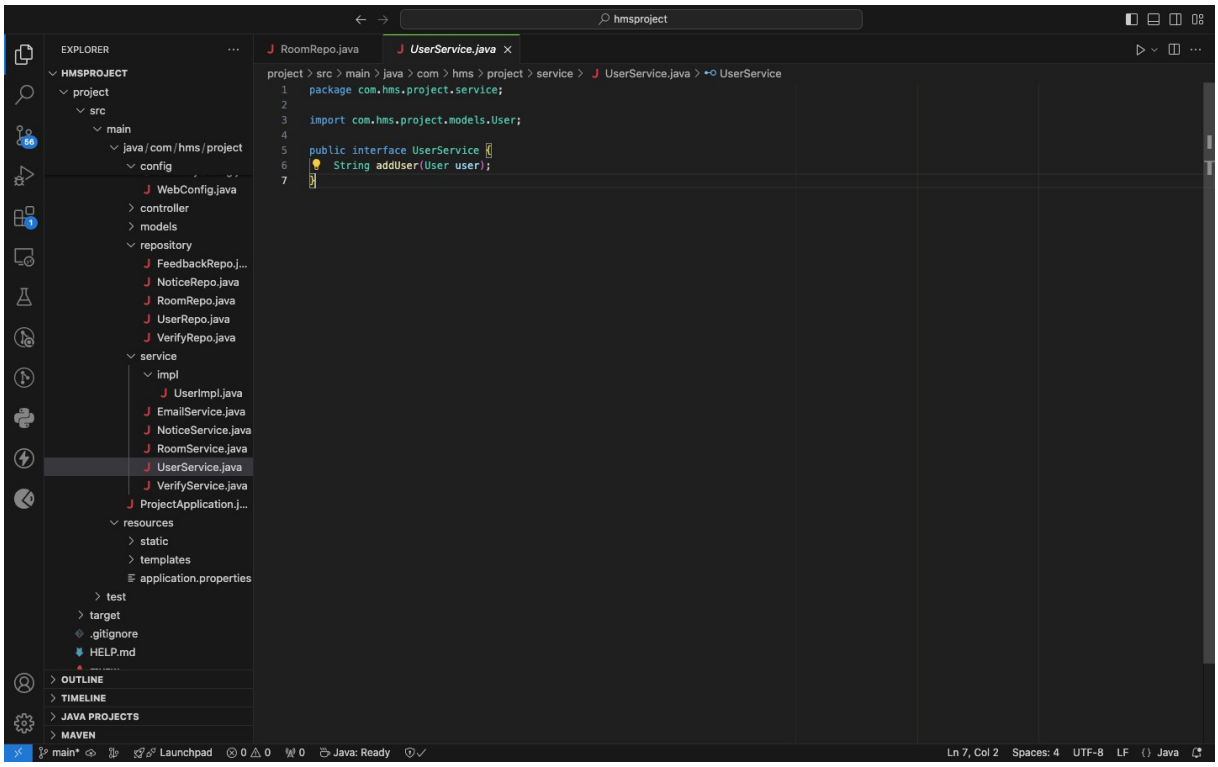


NoticeRepo.java

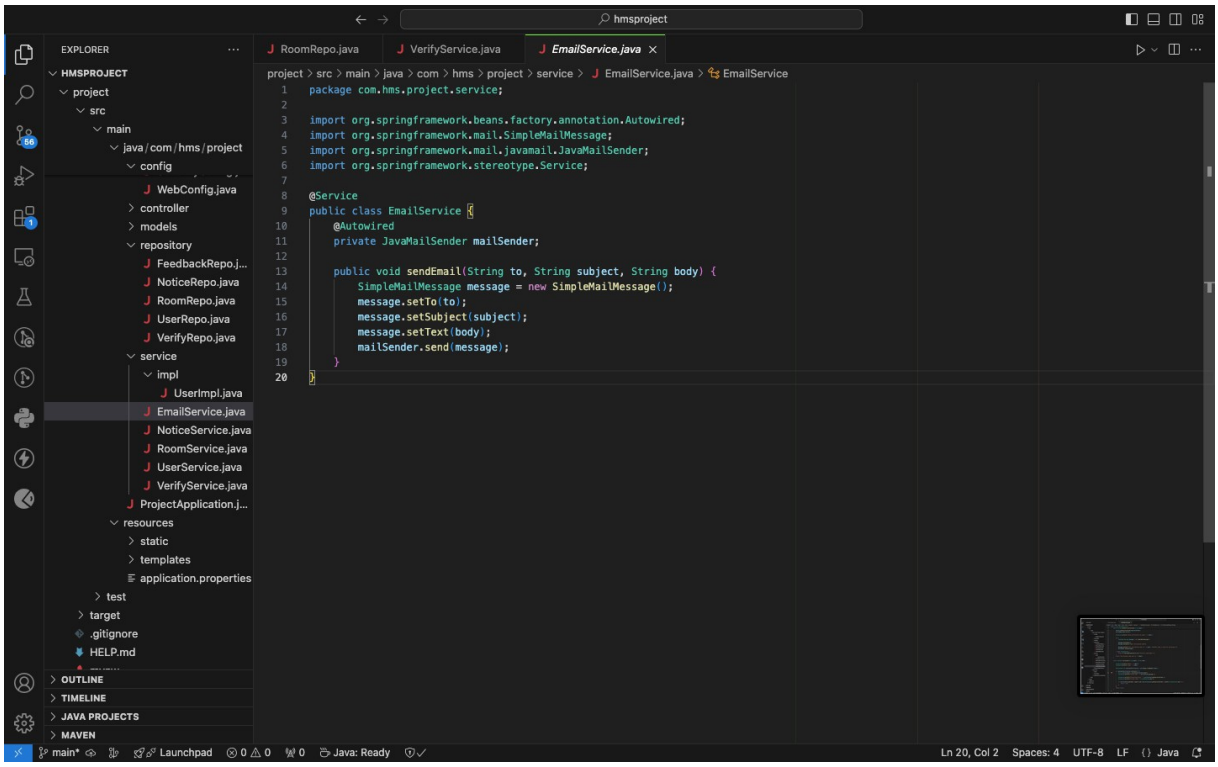


FeedbackRepo.java

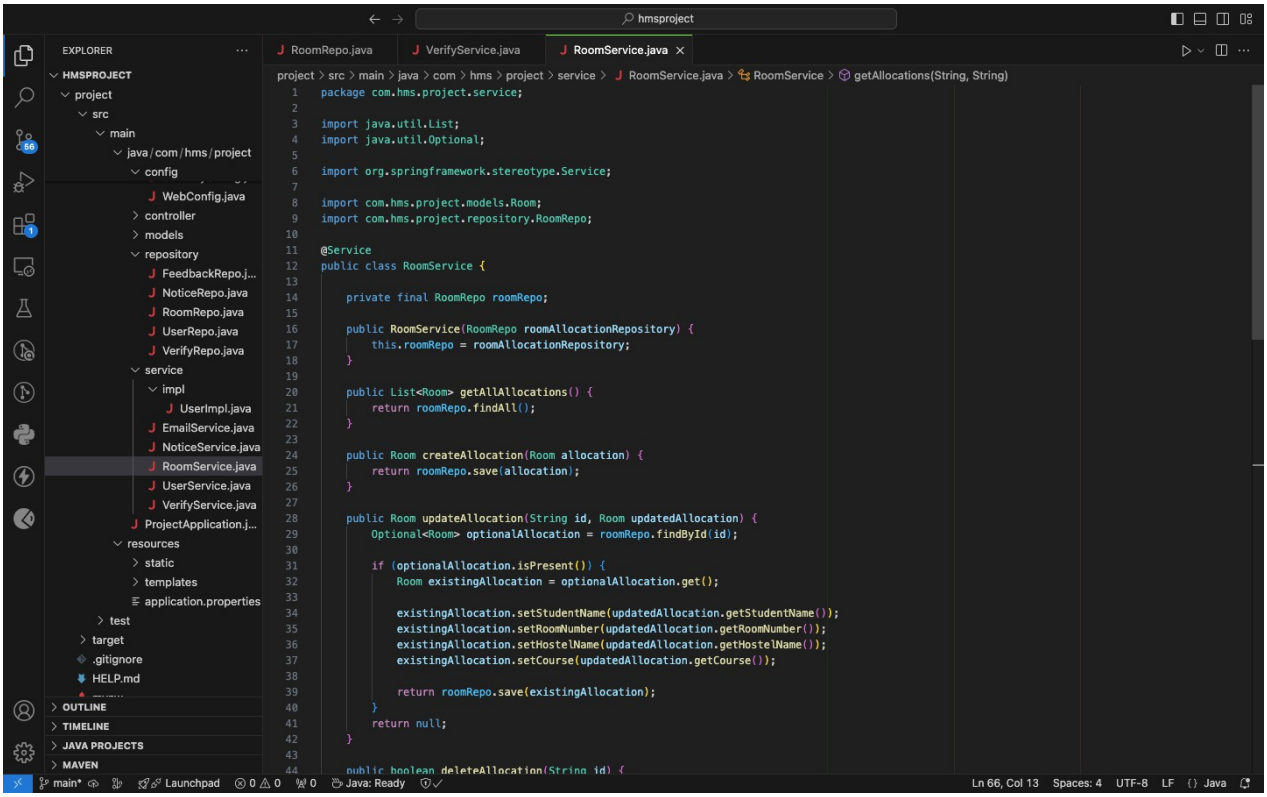
4.1.4 Service

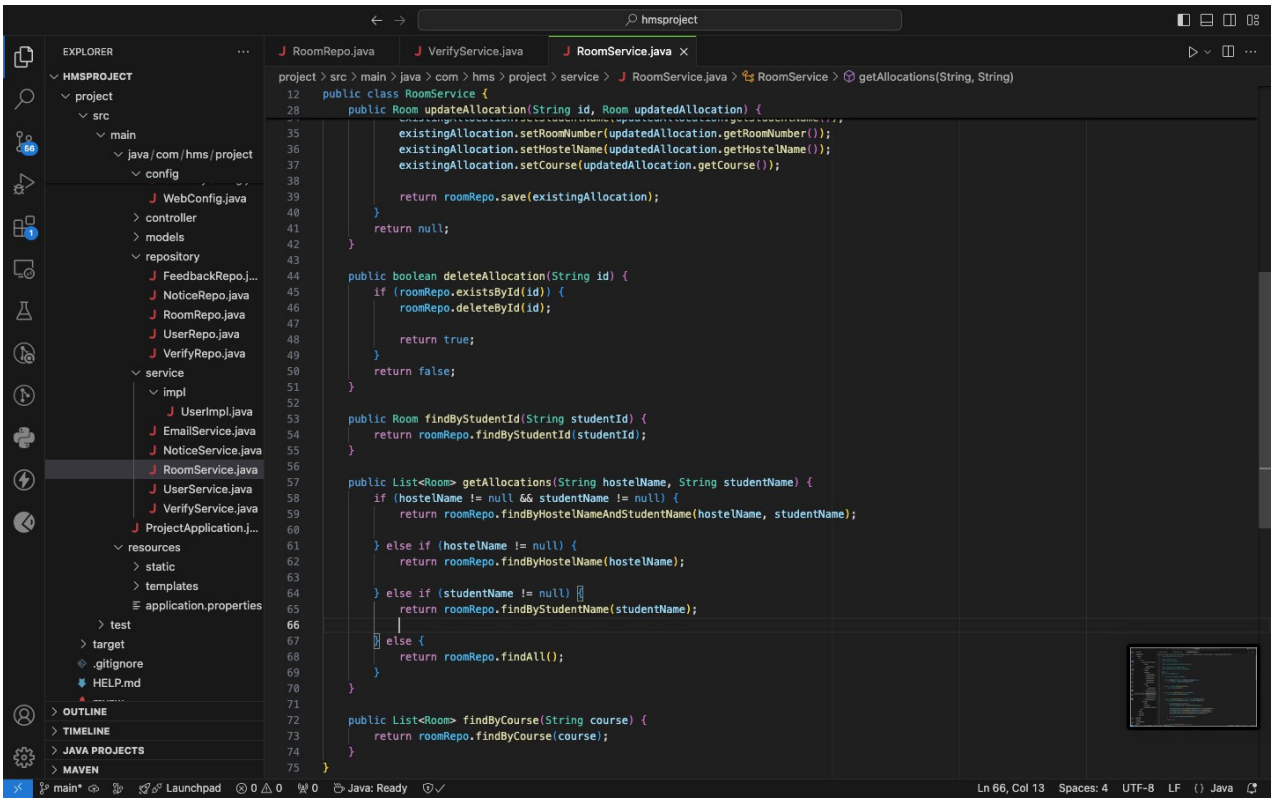


UserService.java

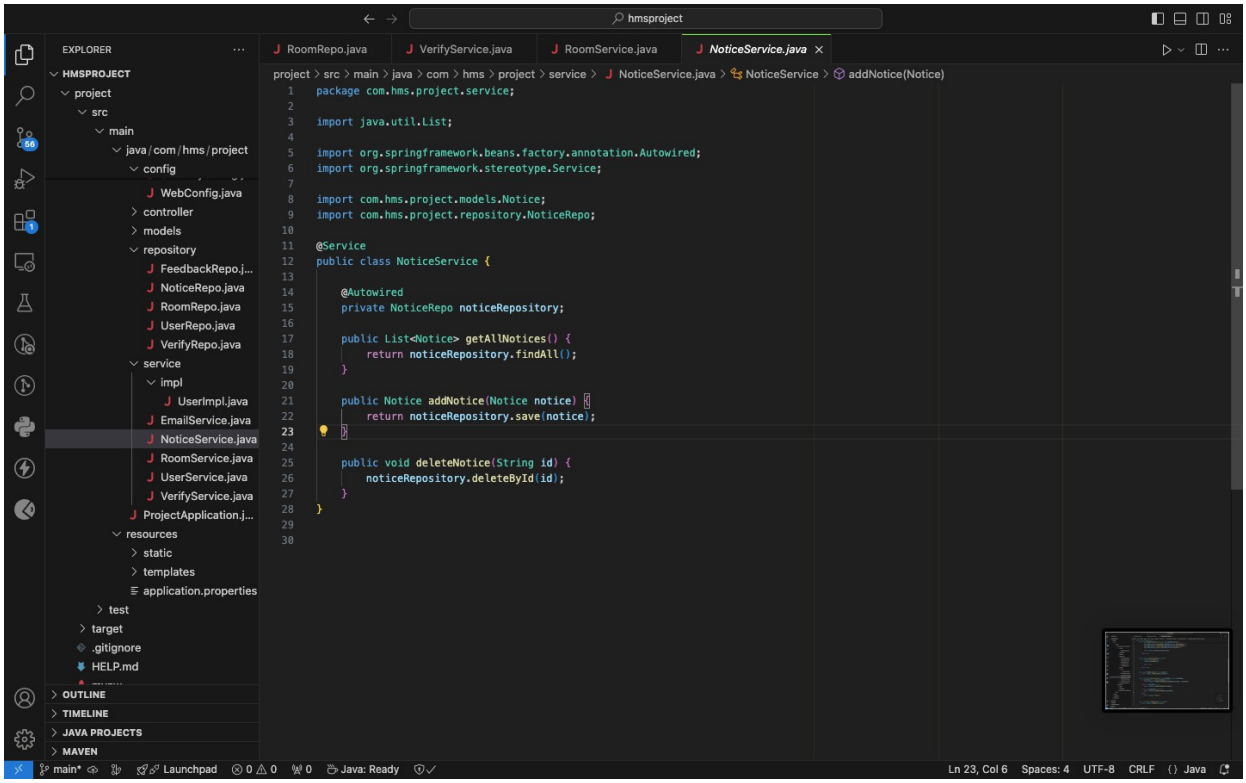


EmailService.java



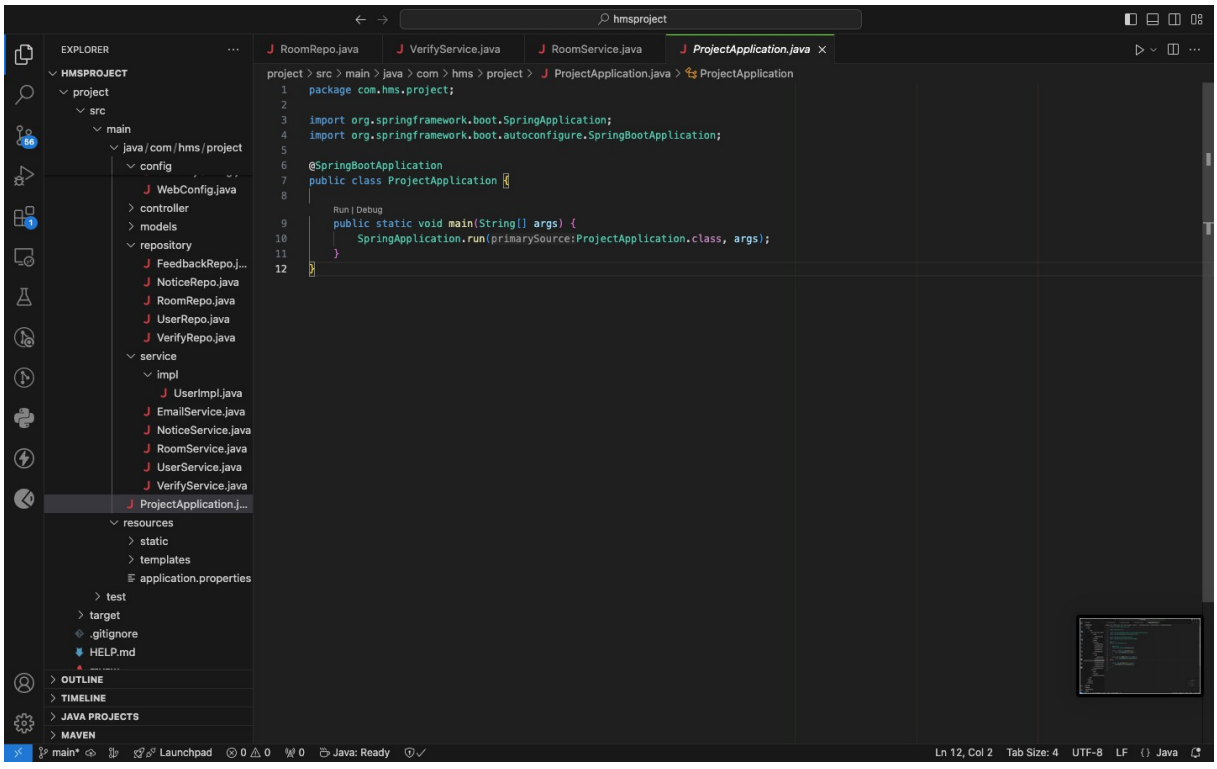


RoomService.java



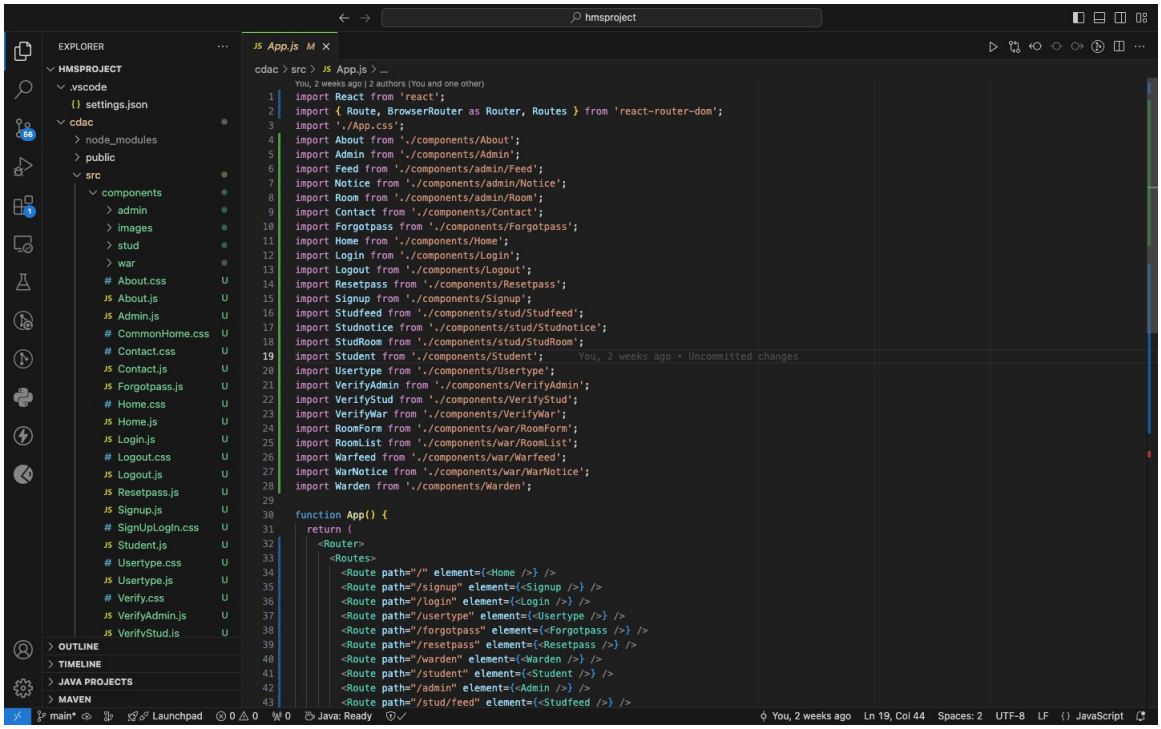
NoticeService.java

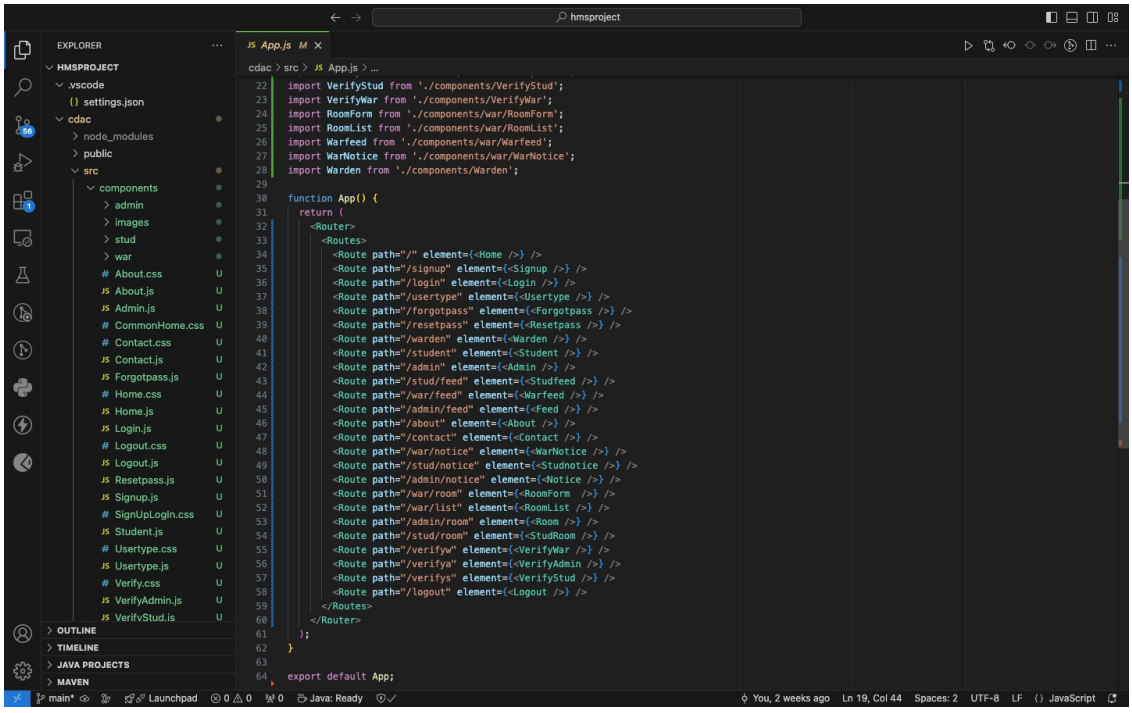
4.1.5 ProjectApplication.java



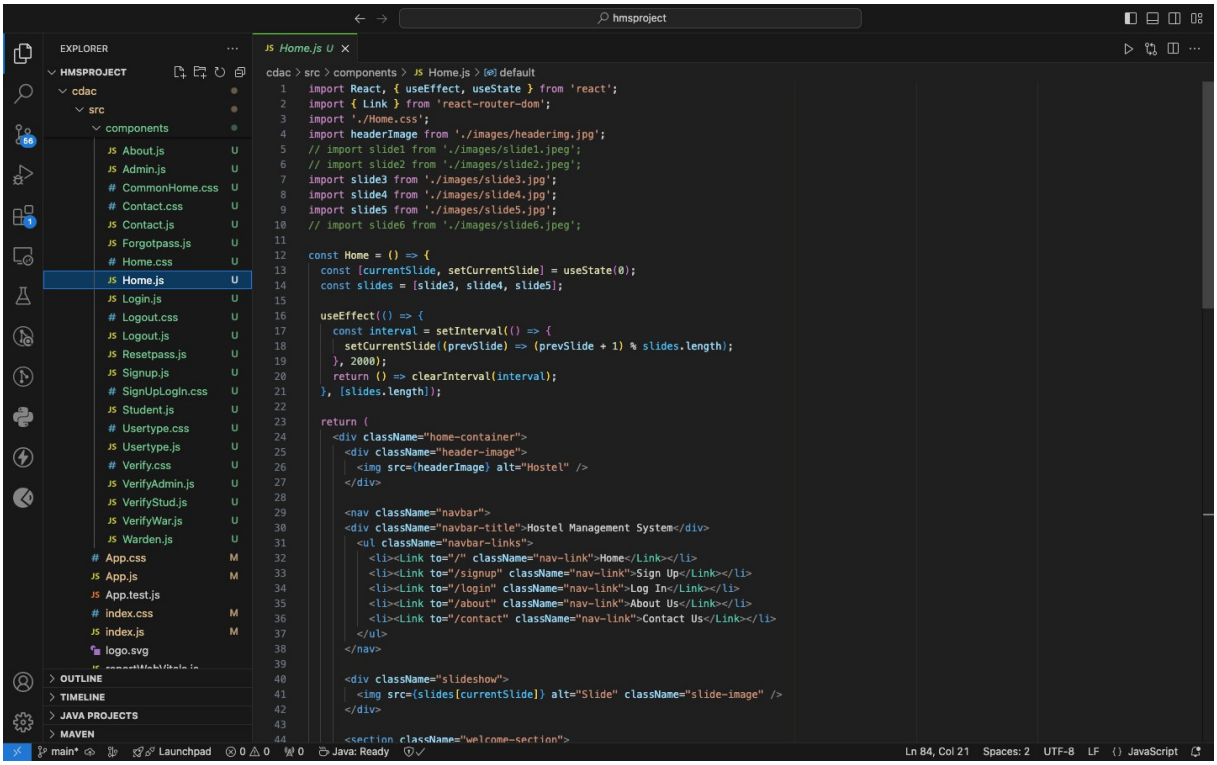
4.2 Frontend

4.2.1 App.js



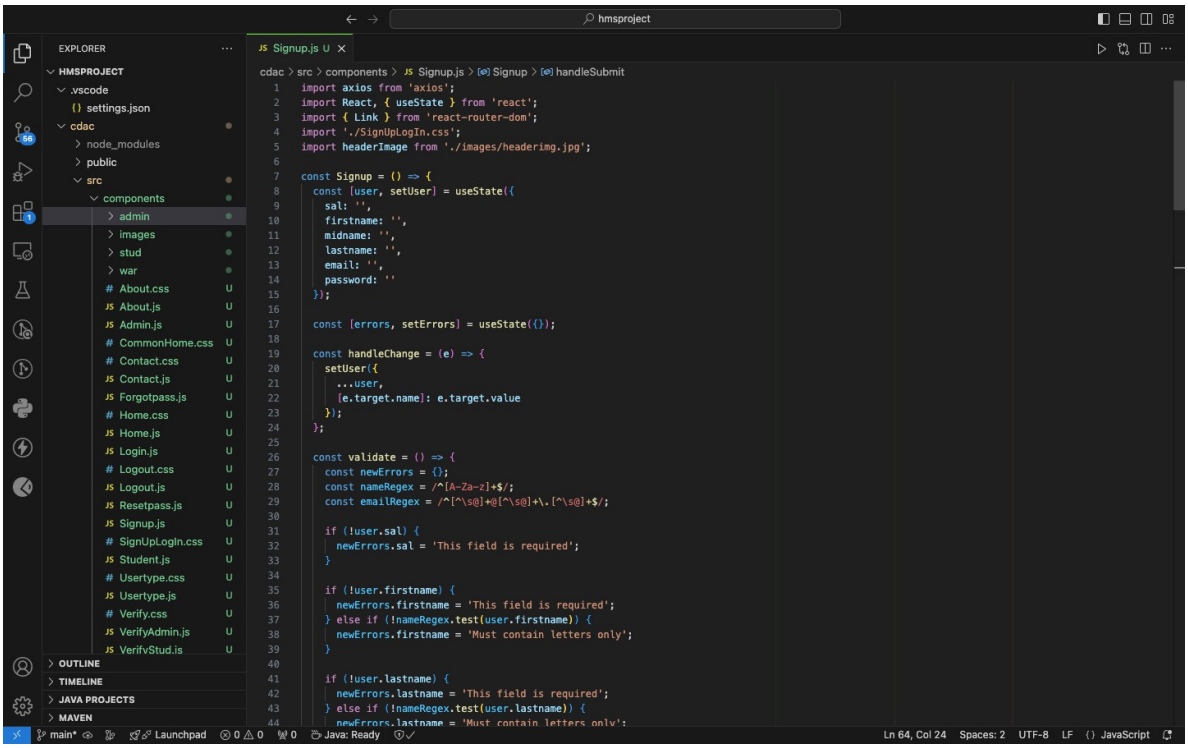


4.2.2 Home.js

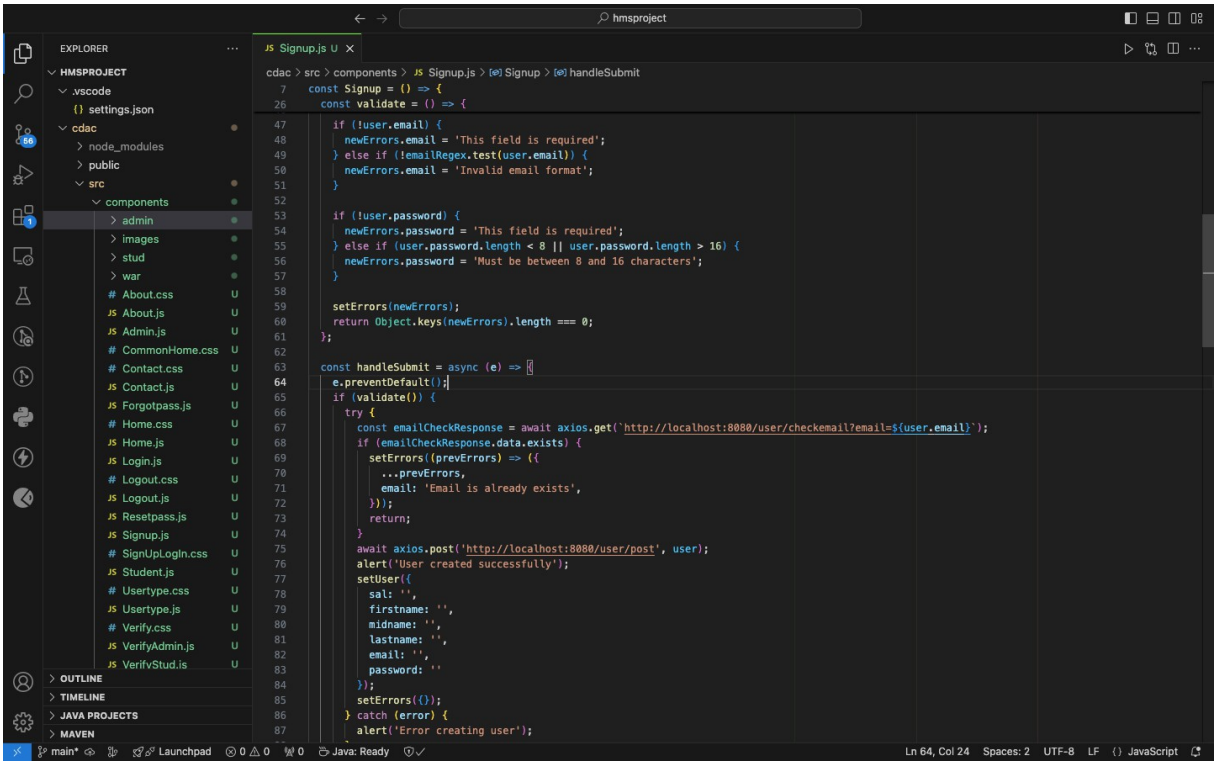


```
import React, { useEffect, useState } from 'react';
import { Link } from 'react-router-dom';
import './Home.css';
import headerImage from './images/headerimg.jpg';
// import slide1 from './images/slide1.jpeg';
// import slide2 from './images/slide2.jpeg';
import slide3 from './images/slide3.jpg';
import slide4 from './images/slide4.jpg';
import slide5 from './images/slide5.jpg';
// import slide6 from './images/slide6.jpeg';

const Home = () => {
  const [currentSlide, setCurrentSlide] = useState(0);
  const slides = [slide3, slide4, slide5];

  useEffect(() => {
    const interval = setInterval(() => {
      setCurrentSlide((prevSlide) => (prevSlide + 1) % slides.length);
    }, 2000);
    return () => clearInterval(interval);
  }, [slides.length]);

  return (
    <div className="home-container">
      <div className="header-image">
        <img src={headerImage} alt="Hostel" />
      </div>

      <nav className="navbar">
      <div className="navbar-title">Hostel Management System</div>
        <ul className="navbar-links">
          <li><Link to="/" className="nav-link">Home</Link></li>
          <li><Link to="/signup" className="nav-link">Sign Up</Link></li>
          <li><Link to="/login" className="nav-link">Log In</Link></li>
          <li><Link to="/about" className="nav-link">About Us</Link></li>
          <li><Link to="/contact" className="nav-link">Contact Us</Link></li>
        </ul>
      </nav>

      <div className="slideshow">
        <img src={slides[currentSlide]} alt="Slide" className="slide-image" />
      </div>
```

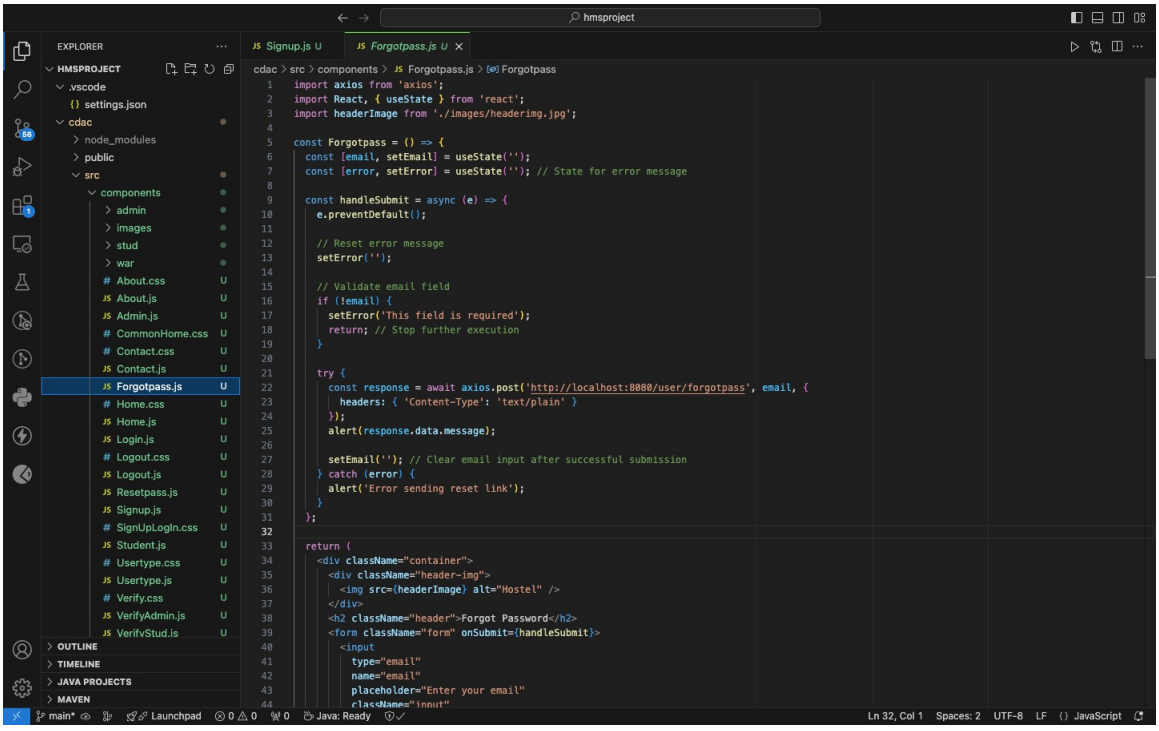
4.2.3 Signup.js



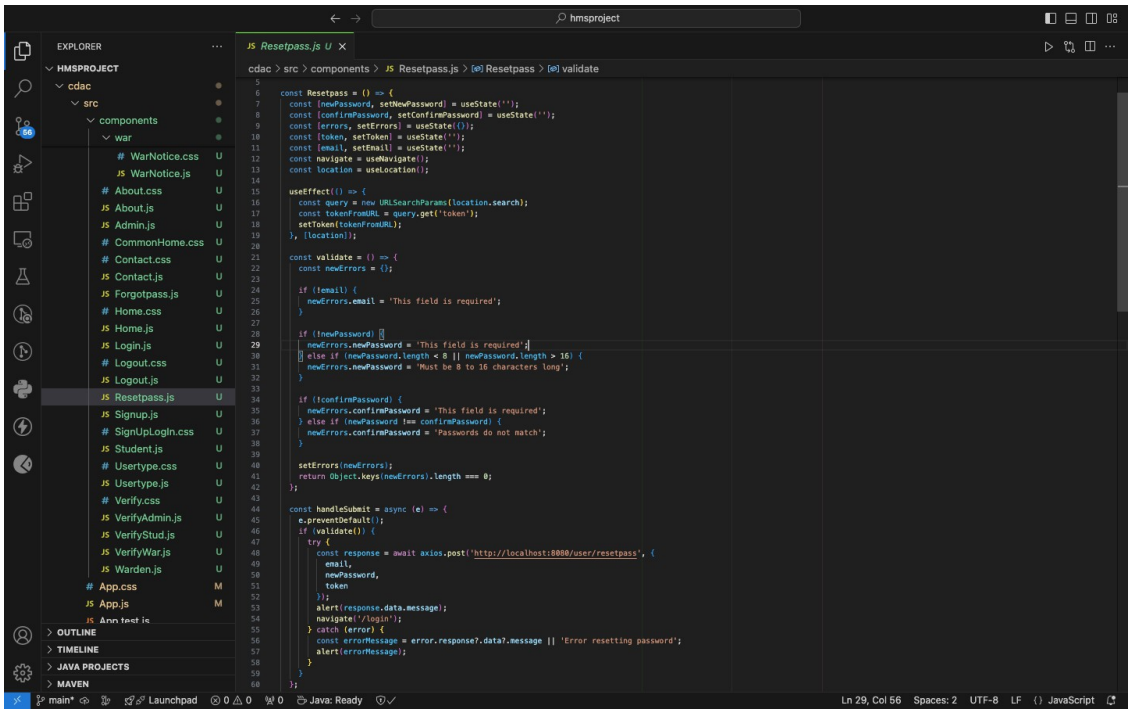
```
import axios from 'axios';
import React, { useState } from 'react';
import { Link } from 'react-router-dom';
import './SignUpLogIn.css';
import headerImage from './images/headerimg.jpg';

const Signup = () => {
  const [user, setUser] = useState({
    sal: '',
    firstname: '',
    midname: '',
    lastname: '',
    email: '',
    password: ''
  });

  const [errors, setErrors] = useState({});

  const handleChange = (e) => {
    setUser({
      ...user,
      [e.target.name]: e.target.value
    });
  };

  const validate = () => {
    const newErrors = {};
    const nameRegex = /^[A-Za-z]+$/;
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;

    if (!user.sal) {
      newErrors.sal = 'This field is required';
    }

    if (!user.firstname) {
      newErrors.firstname = 'This field is required';
    } else if (!nameRegex.test(user.firstname)) {
      newErrors.firstname = 'Must contain letters only';
    }

    if (!user.lastname) {
      newErrors.lastname = 'This field is required';
    } else if (!nameRegex.test(user.lastname)) {
      newErrors.lastname = 'Must contain letters only';
```



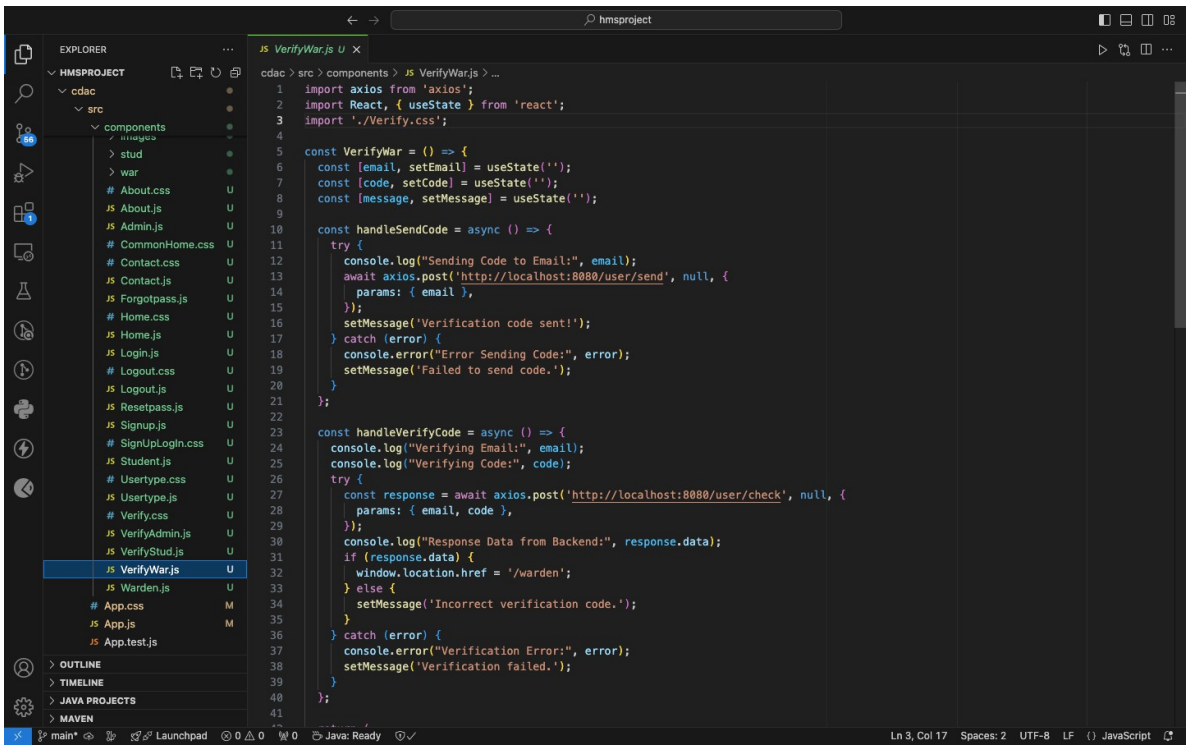
4.2.4 ForgotPass.js



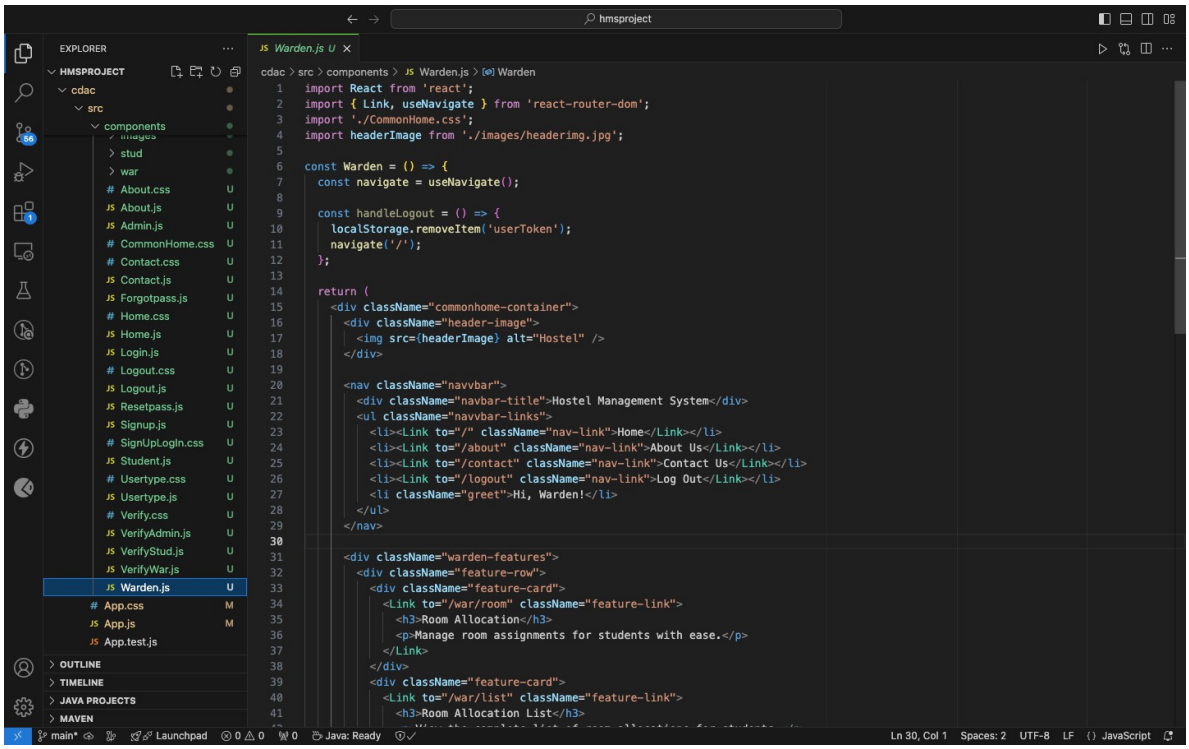
4.2.5 ResetPass.js



4.2.6 VerifyWar.js

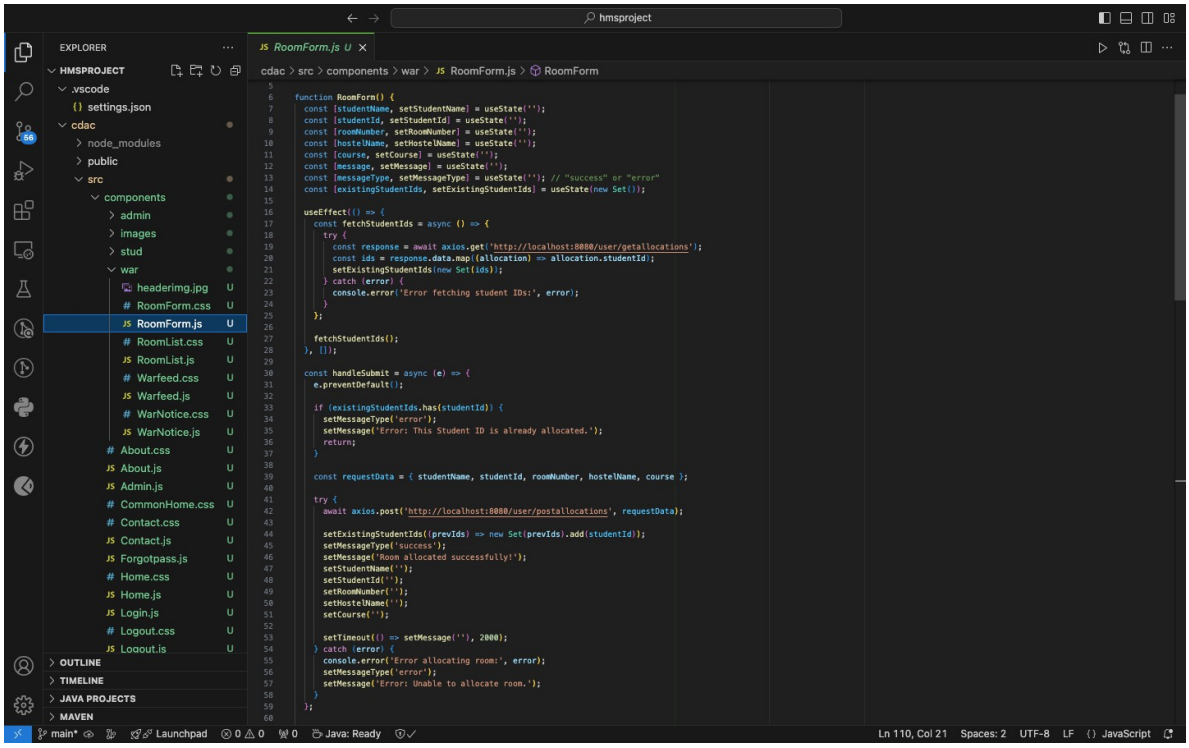


4.2.7 Warden.js



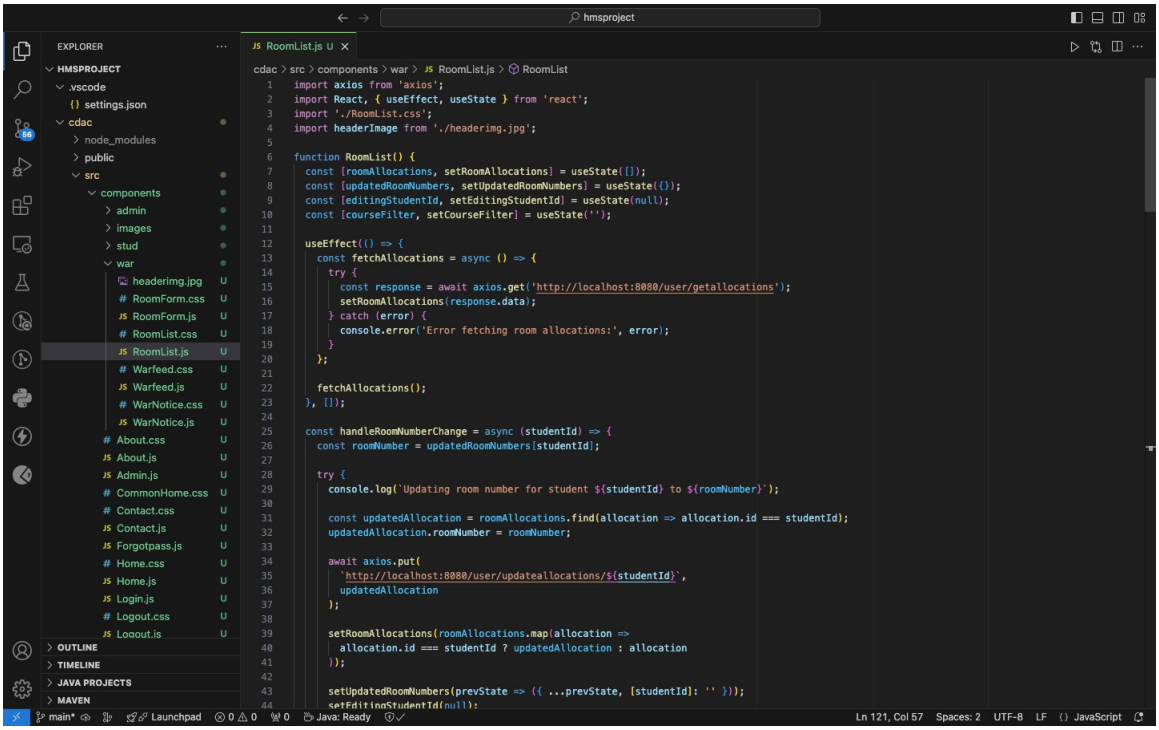
```
import React from 'react';
import { Link, useNavigate } from 'react-router-dom';
import './CommonHome.css';
import headerImage from './images/headerimg.jpg';

const Warden = () => {
  const navigate = useNavigate();

  const handleLogout = () => {
    localStorage.removeItem('userToken');
    navigate('/');
  };

  return (
    <div className="commonhome-container">
      <div className="header-image">
        <img src={headerImage} alt="Hostel" />
      </div>

      <nav className="navvbar">
        <div className="navbar-title">Hostel Management System</div>
        <ul className="navvbar-links">
          <li><Link to="/" className="nav-link">Home</Link></li>
          <li><Link to="/about" className="nav-link">About Us</Link></li>
          <li><Link to="/contact" className="nav-link">Contact Us</Link></li>
          <li><Link to="/logout" className="nav-link">Log Out</Link></li>
          <li className="greet">Hi, Warden!</li>
        </ul>
      </nav>

      <div className="warden-features">
        <div className="feature-row">
          <div className="feature-card">
            <Link to="/war/room" className="feature-link">
              <h3>Room Allocation</h3>
              <p>Manage room assignments for students with ease.</p>
            </Link>
          </div>
          <div className="feature-card">
            <Link to="/war/list" className="feature-link">
              <h3>Room Allocation List</h3>
```

4.2.8 RoomForm.js

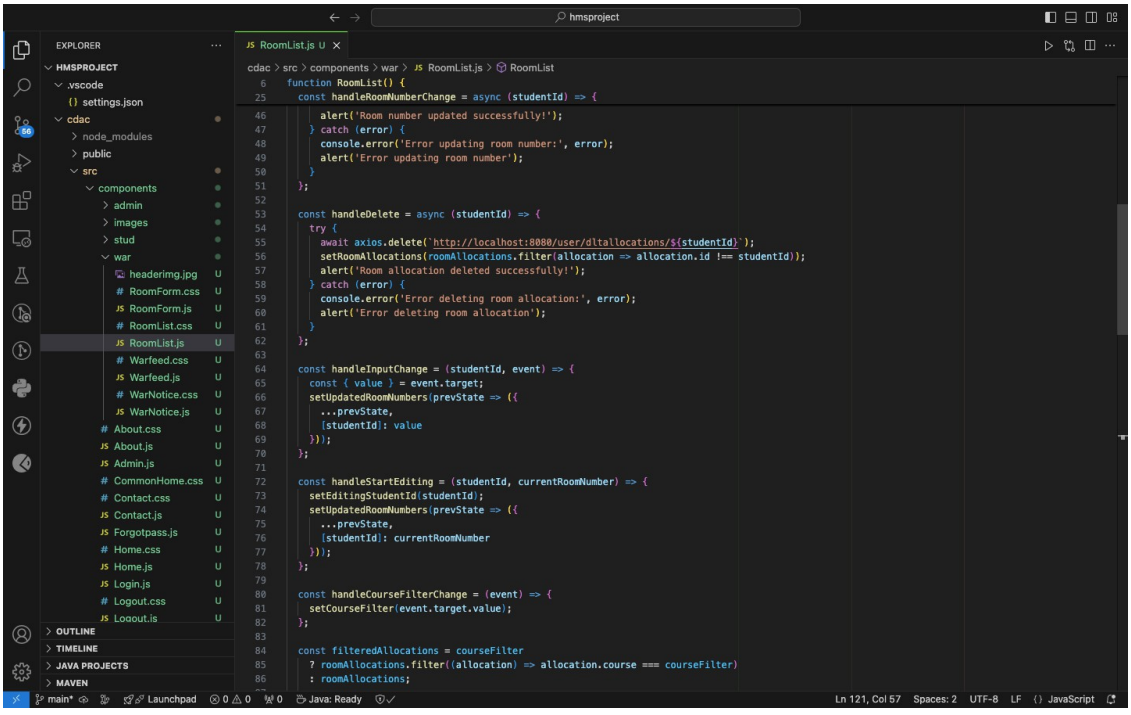


```
function RoomForm() {
  const [studentName, setStudentName] = useState('');
  const [studentId, setStudentId] = useState('');
  const [roomNumber, setRoomNumber] = useState('');
  const [hostelName, setHostelName] = useState('');
  const [course, setCourse] = useState('');
  const [message, setMessage] = useState('');
  const [messageType, setMessageType] = useState(''); // "success" or "error"
  const [existingStudentIds, setExistingStudentIds] = useState(new Set());

  useEffect(() => {
    const fetchStudentIds = async () => {
      try {
        const response = await axios.get('http://localhost:8080/user/getallocations');
        const ids = response.data.map((allocation) => allocation.studentId);
        setExistingStudentIds(new Set(ids));
      } catch (error) {
        console.error('Error fetching student IDs:', error);
      }
    };

    fetchStudentIds();
  }, []);

  const handleSubmit = async (e) => {
    e.preventDefault();

    if (existingStudentIds.has(studentId)) {
      setMessageType('error');
      setMessage('Error: This Student ID is already allocated.');
      return;
    }

    const requestData = { studentName, studentId, roomNumber, hostelName, course };

    try {
      await axios.post('http://localhost:8080/user/postallocations', requestData);

      setExistingStudentIds((prevIds) => new Set(prevIds).add(studentId));
      setMessageType('success');
      setMessage('Room allocated successfully!');
      setStudentName('');
      setStudentId('');
      setRoomNumber('');
      setHostelName('');
      setCourse('');

      setTimeout(() => setMessage(''), 2000);
    } catch (error) {
      console.error('Error allocating room:', error);
      setMessageType('error');
      setMessage('Error: Unable to allocate room.');
    }
  };
```

4.2.9 RoomList.js

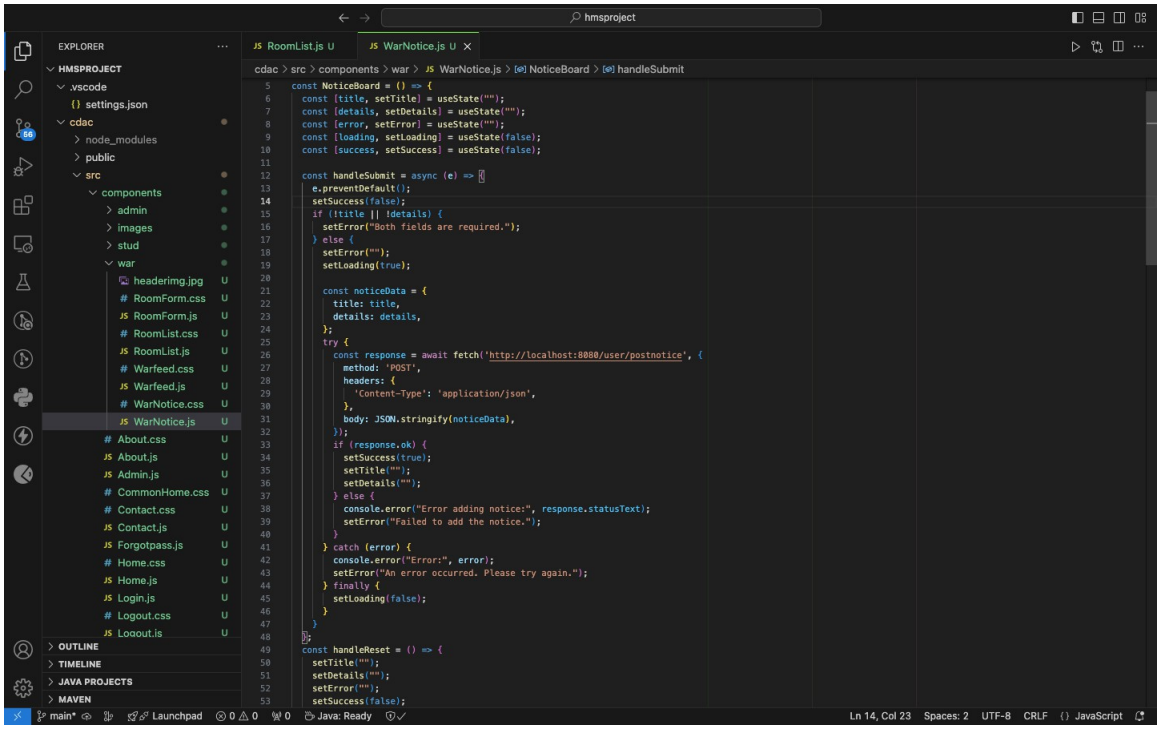


```
import axios from 'axios';
import React, { useEffect, useState } from 'react';
import './RoomList.css';
import headerImage from './headerimg.jpg';

function RoomList() {
  const [roomAllocations, setRoomAllocations] = useState([]);
  const [updatedRoomNumbers, setUpdatedRoomNumbers] = useState({});
  const [editingStudentId, setEditingStudentId] = useState(null);
  const [courseFilter, setCourseFilter] = useState('');

  useEffect(() => {
    const fetchAllocations = async () => {
      try {
        const response = await axios.get('http://localhost:8080/user/getallocations');
        setRoomAllocations(response.data);
      } catch (error) {
        console.error('Error fetching room allocations:', error);
      }
    };

    fetchAllocations();
  }, []);

  const handleRoomNumberChange = async (studentId) => {
    const roomNumber = updatedRoomNumbers[studentId];

    try {
      console.log(`Updating room number for student ${studentId} to ${roomNumber}`);

      const updatedAllocation = roomAllocations.find(allocation => allocation.id === studentId);
      updatedAllocation.roomNumber = roomNumber;

      await axios.put(
        `http://localhost:8080/user/updateallocations/${studentId}`,
        updatedAllocation
      );

      setRoomAllocations(roomAllocations.map(allocation =>
        allocation.id === studentId ? updatedAllocation : allocation
      ));

      setUpdatedRoomNumbers(prevState => ({ ...prevState, [studentId]: '' }));
      setEditingStudentId(null);
```



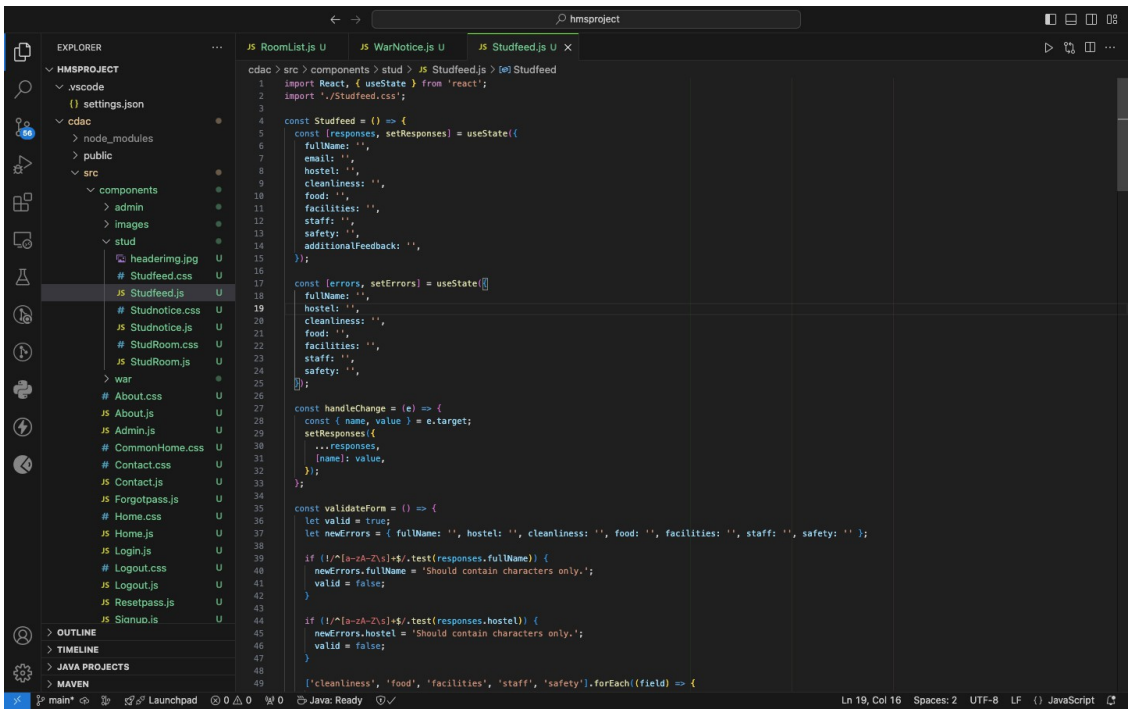
```
function RoomList() {
  const handleRoomNumberChange = async (studentId) => {
      alert('Room number updated successfully!');
    } catch (error) {
      console.error('Error updating room number:', error);
      alert('Error updating room number');
    }
  };

  const handleDelete = async (studentId) => {
    try {
      await axios.delete(`http://localhost:8080/user/dltallocations/${studentId}`);
      setRoomAllocations(roomAllocations.filter(allocation => allocation.id !== studentId));
      alert('Room allocation deleted successfully!');
    } catch (error) {
      console.error('Error deleting room allocation:', error);
      alert('Error deleting room allocation');
    }
  };

  const handleInputChange = (studentId, event) => {
    const { value } = event.target;
    setUpdatedRoomNumbers(prevState => ({
      ...prevState,
      [studentId]: value
    }));
  };

  const handleStartEditing = (studentId, currentRoomNumber) => {
    setEditingStudentId(studentId);
    setUpdatedRoomNumbers(prevState => ({
      ...prevState,
      [studentId]: currentRoomNumber
    }));
  };

  const handleCourseFilterChange = (event) => {
    setCourseFilter(event.target.value);
  };

  const filteredAllocations = courseFilter
    ? roomAllocations.filter((allocation) => allocation.course === courseFilter)
    : roomAllocations;
```

4.2.10 WarNotice.js

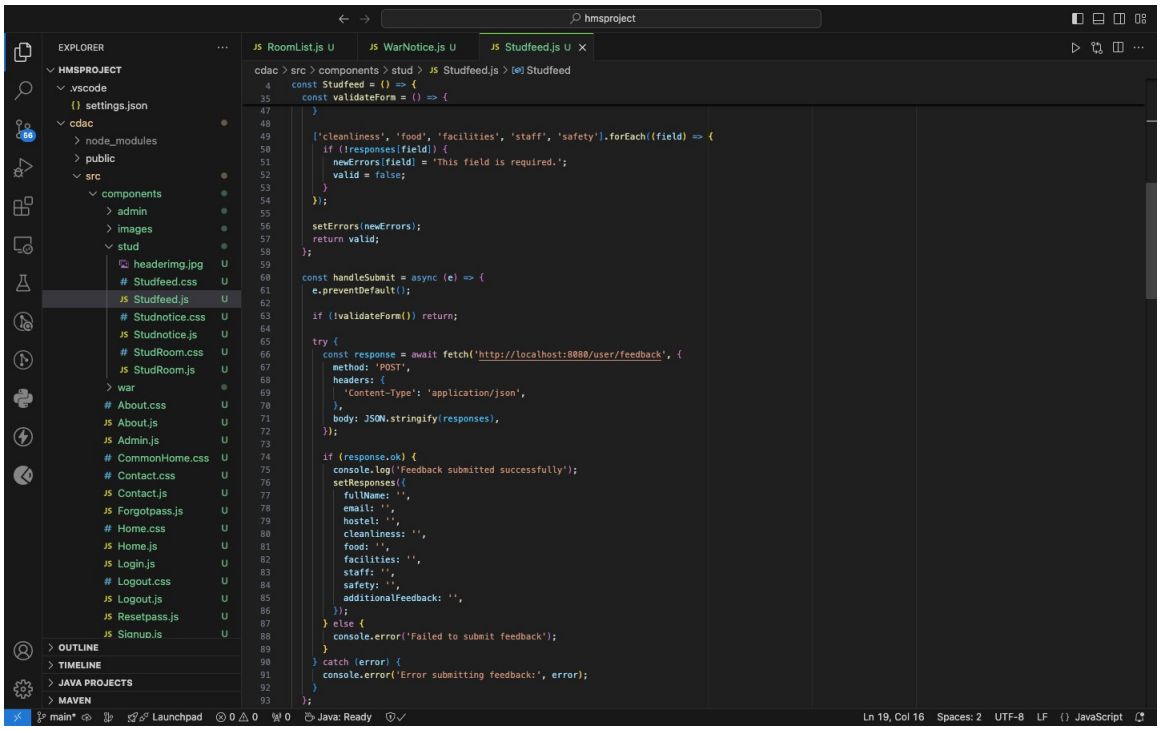


```
const NoticeBoard = () => {
  const [title, setTitle] = useState("");
  const [details, setDetails] = useState("");
  const [error, setError] = useState("");
  const [loading, setLoading] = useState(false);
  const [success, setSuccess] = useState(false);

  const handleSubmit = async (e) => {
    e.preventDefault();
    setSuccess(false);
    if (!title || !details) {
      setError("Both fields are required.");
    } else {
      setError("");
      setLoading(true);

      const noticeData = {
        title: title,
        details: details,
      };
      try {
        const response = await fetch('http://localhost:8080/user/postnotice', {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
          },
          body: JSON.stringify(noticeData),
        });
        if (response.ok) {
          setSuccess(true);
          setTitle("");
          setDetails("");
        } else {
          console.error("Error adding notice:", response.statusText);
          setError("Failed to add the notice.");
        }
      } catch (error) {
        console.error("Error:", error);
        setError("An error occurred. Please try again.");
      } finally {
        setLoading(false);
      }
    }
  };
  const handleReset = () => {
    setTitle("");
    setDetails("");
    setError("");
    setSuccess(false);
```

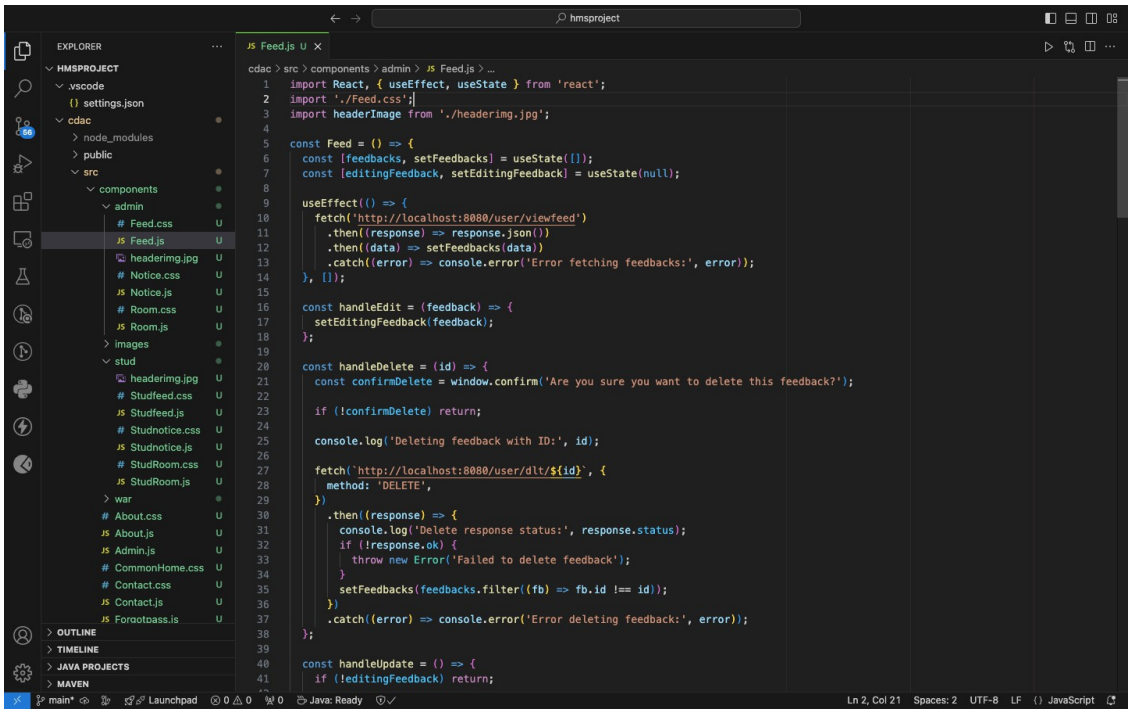
4.2.11 Studfeed.js

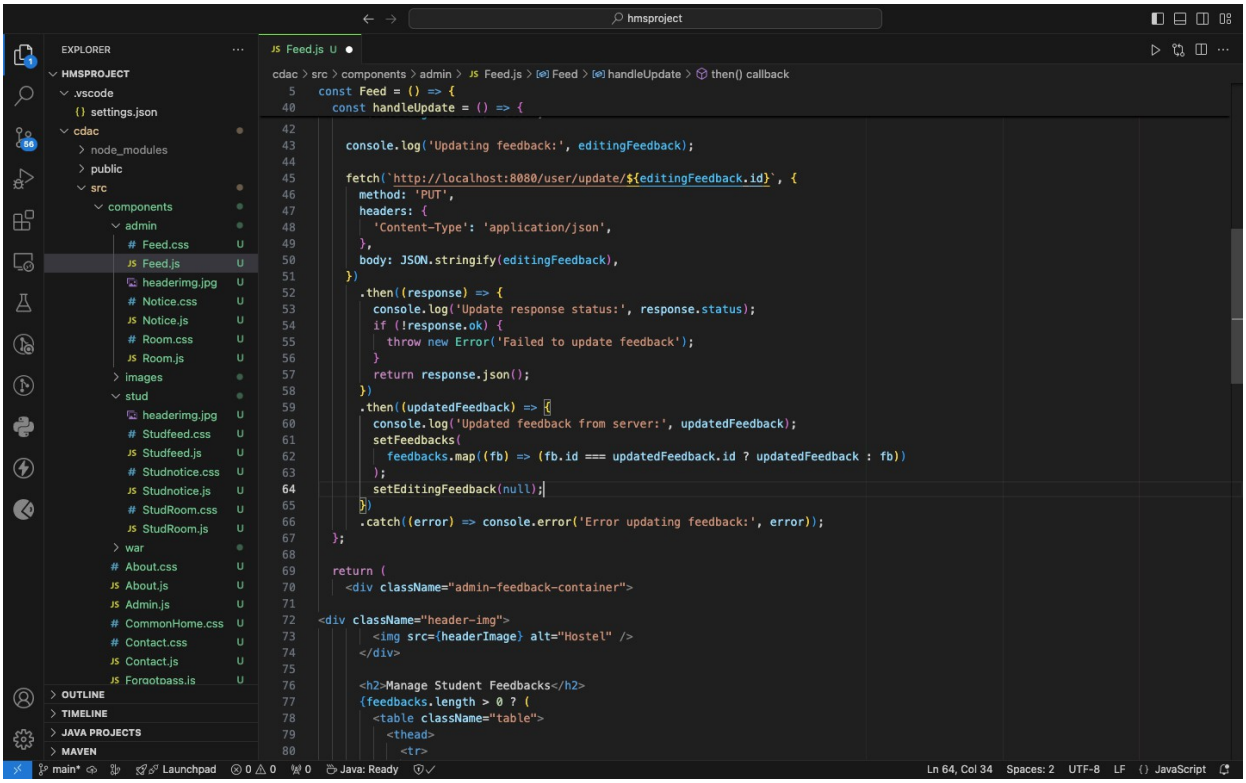


```
import React, { useState } from 'react';
import './Studfeed.css';

const Studfeed = () => {
  const [responses, setResponses] = useState({
    fullName: '',
    email: '',
    hostel: '',
    cleanliness: '',
    food: '',
    facilities: '',
    staff: '',
    safety: '',
    additionalFeedback: '',
  });

  const [errors, setErrors] = useState({
    fullName: '',
    hostel: '',
    cleanliness: '',
    food: '',
    facilities: '',
    staff: '',
    safety: '',
  });

  const handleChange = (e) => {
    const { name, value } = e.target;
    setResponses({
      ...responses,
      [name]: value,
    });
  };

  const validateForm = () => {
    let valid = true;
    let newErrors = { fullName: '', hostel: '', cleanliness: '', food: '', facilities: '', staff: '', safety: '' };

    if (!/^[a-zA-Z\s]+$/.test(responses.fullName)) {
      newErrors.fullName = 'Should contain characters only.';
      valid = false;
    }

    if (!/^[a-zA-Z\s]+$/.test(responses.hostel)) {
      newErrors.hostel = 'Should contain characters only.';
      valid = false;
    }

    ['cleanliness', 'food', 'facilities', 'staff', 'safety'].forEach((field) => {
```

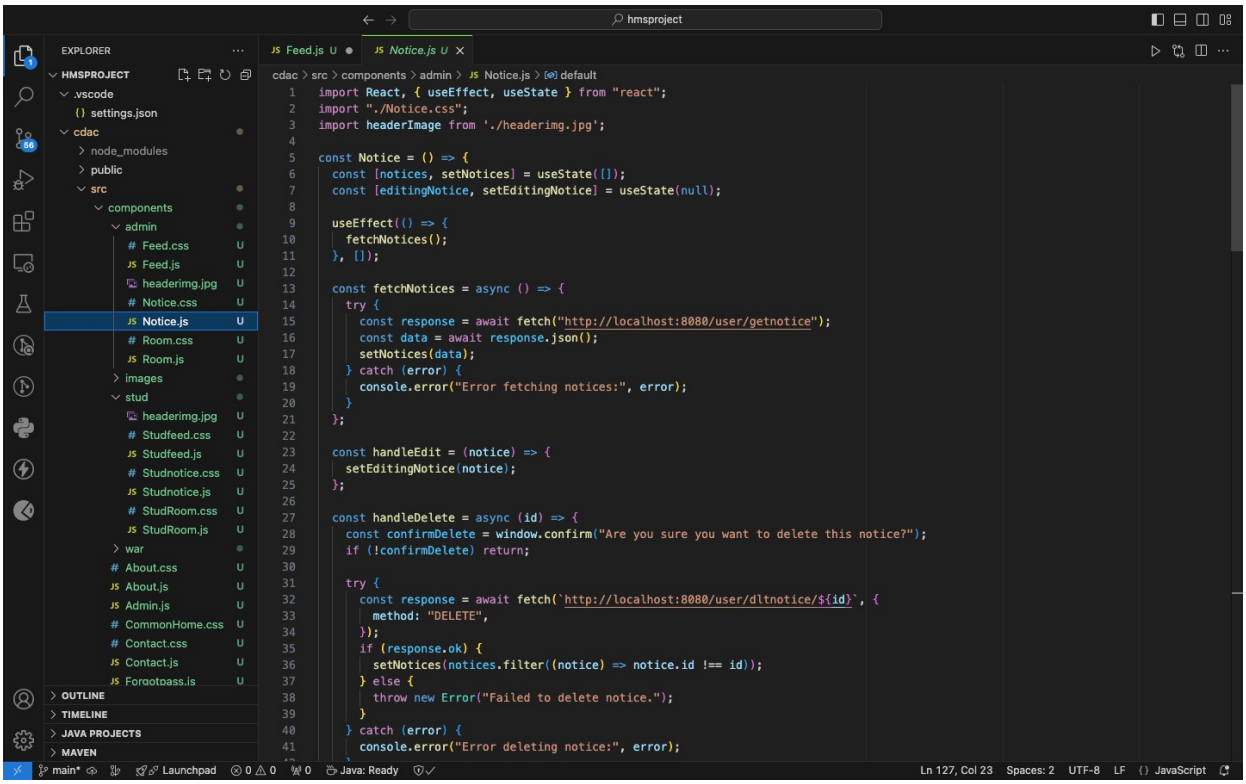

hmsproject
EXPLORER
HMSPROJECT
.vscode
settings.json
cdac
node_modules
public
src
components
admin
images
stud
headerimg.jpg
Studfeed.css
Studfeed.js
Studnotice.css
Studnotice.js
StudRoom.css
StudRoom.js
war
About.css
About.js
Admin.js
CommonHome.css
Contact.css
Contact.js
Forgotpass.js
Home.css
Home.js
Login.js
Logout.css
Logout.js
Resetpass.js
Signup.js
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
RoomList.js U
WarNotice.js U
Studfeed.js U
cdac > src > components > stud > Studfeed.js > Studfeed
const Studfeed = () => {
const validateForm = () => {
}
['cleanliness', 'food', 'facilities', 'staff', 'safety'].forEach((field) => {
if (!responses[field]) {
newErrors[field] = 'This field is required.';
valid = false;
}
});
setErrors(newErrors);
return valid;
};
const handleSubmit = async (e) => {
e.preventDefault();
if (!validateForm()) return;
try {
const response = await fetch('http://localhost:8080/user/feedback', {
method: 'POST',
headers: {
'Content-Type': 'application/json',
},
body: JSON.stringify(responses),
});
if (response.ok) {
console.log('Feedback submitted successfully');
setResponses({
fullName: '',
email: '',
hostel: '',
cleanliness: '',
food: '',
facilities: '',
staff: '',
safety: '',
additionalFeedback: '',
});
} else {
console.error('Failed to submit feedback');
}
} catch (error) {
console.error('Error submitting feedback:', error);
}
};
main*
Launchpad
Java: Ready
Ln 19, Col 16
Spaces: 2
UTF-8
LF
JavaScript

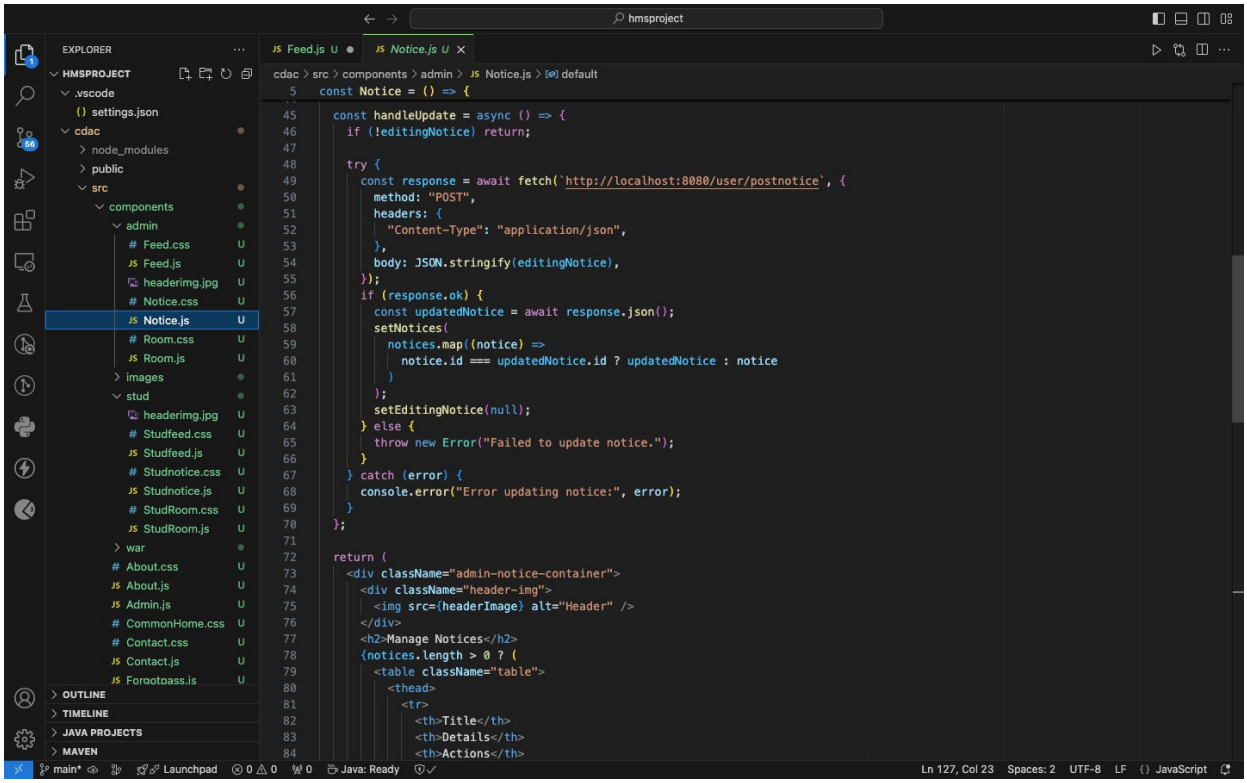
4.2.12 Feed.js


hmsproject
EXPLORER
HMSPROJECT
.vscode
settings.json
cdac
node_modules
public
src
components
admin
Feed.css
Feed.js
headerimg.jpg
Notice.css
Notice.js
Room.css
Room.js
images
stud
headerimg.jpg
Studfeed.css
Studfeed.js
Studnotice.css
Studnotice.js
StudRoom.css
StudRoom.js
war
About.css
About.js
Admin.js
CommonHome.css
Contact.css
Contact.js
Forgotpass.js
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Feed.js U
cdac > src > components > admin > Feed.js > ...
import React, { useEffect, useState } from 'react';
import './Feed.css';
import headerImage from './headerimg.jpg';
const Feed = () => {
const [feedbacks, setFeedbacks] = useState([]);
const [editingFeedback, setEditingFeedback] = useState(null);
useEffect(() => {
fetch('http://localhost:8080/user/viewfeed')
.then((response) => response.json())
.then((data) => setFeedbacks(data))
.catch((error) => console.error('Error fetching feedbacks:', error));
}, []);
const handleEdit = (feedback) => {
setEditingFeedback(feedback);
};
const handleDelete = (id) => {
const confirmDelete = window.confirm('Are you sure you want to delete this feedback?');
if (!confirmDelete) return;
console.log('Deleting feedback with ID:', id);
fetch(`http://localhost:8080/user/dlt/${id}`, {
method: 'DELETE',
})
.then((response) => {
console.log('Delete response status:', response.status);
if (!response.ok) {
throw new Error('Failed to delete feedback');
}
setFeedbacks(feedbacks.filter((fb) => fb.id !== id));
})
.catch((error) => console.error('Error deleting feedback:', error));
};
const handleUpdate = () => {
if (!editingFeedback) return;
main*
Launchpad
Java: Ready
Ln 2, Col 21
Spaces: 2
UTF-8
LF
JavaScript



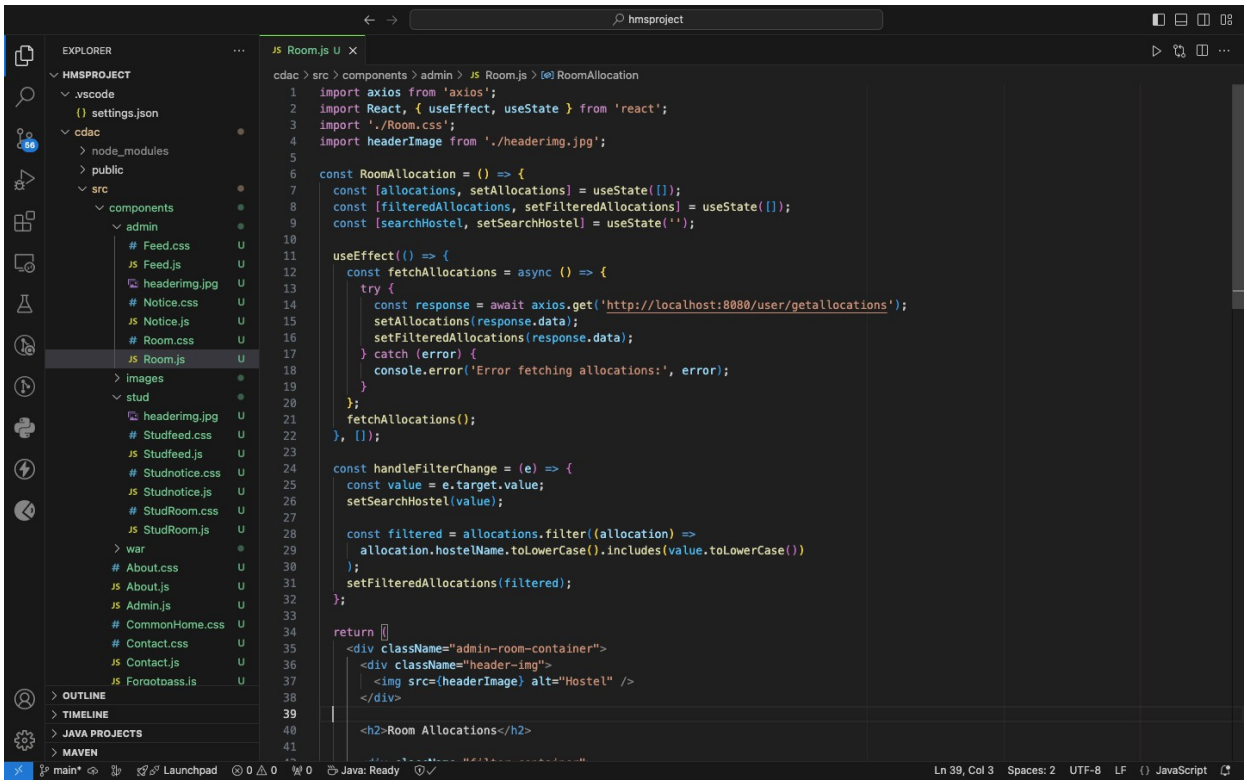
4.2.13 Notice.js




hmsproject
EXPLORER
HMSPROJECT
.vscode
settings.json
cdac
node_modules
public
src
components
admin
Feed.css
Feed.js
headerimg.jpg
Notice.css
Notice.js
Room.css
Room.js
images
stud
headerimg.jpg
Studfeed.css
Studfeed.js
Studnotice.css
Studnotice.js
StudRoom.css
StudRoom.js
war
About.css
About.js
Admin.js
CommonHome.css
Contact.css
Contact.js
Forgotpass.js
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Feed.js U
Notice.js U
cdac > src > components > admin > Notice.js > default
const Notice = () => {
const handleUpdate = async () => {
if (!editingNotice) return;
try {
const response = await fetch(`http://localhost:8080/user/postnotice`, {
method: "POST",
headers: {
"Content-Type": "application/json",
},
body: JSON.stringify(editingNotice),
});
if (response.ok) {
const updatedNotice = await response.json();
setNotices(
notices.map((notice) =>
notice.id === updatedNotice.id ? updatedNotice : notice
)
);
setEditingNotice(null);
} else {
throw new Error("Failed to update notice.");
}
} catch (error) {
console.error("Error updating notice:", error);
}
};
return (

<h2>Manage Notices</h2>
{notices.length > 0 ? (
<table className="table">
<thead>
<tr>
<th>Title</th>
<th>Details</th>
<th>Actions</th>
main*
Launchpad
Java: Ready
Ln 127, Col 23
Spaces: 2
UTF-8
LF
JavaScript

4.2.14 Room.js


hmsproject
EXPLORER
HMSPROJECT
.vscode
settings.json
cdac
node_modules
public
src
components
admin
Feed.css
Feed.js
headerimg.jpg
Notice.css
Notice.js
Room.css
Room.js
images
stud
headerimg.jpg
Studfeed.css
Studfeed.js
Studnotice.css
Studnotice.js
StudRoom.css
StudRoom.js
war
About.css
About.js
Admin.js
CommonHome.css
Contact.css
Contact.js
Forgotpass.js
OUTLINE
TIMELINE
JAVA PROJECTS
MAVEN
Room.js U
cdac > src > components > admin > Room.js > RoomAllocation
import axios from 'axios';
import React, { useEffect, useState } from 'react';
import './Room.css';
import headerImage from './headerimg.jpg';
const RoomAllocation = () => {
const [allocations, setAllocations] = useState([]);
const [filteredAllocations, setFilteredAllocations] = useState([]);
const [searchHostel, setSearchHostel] = useState('');
useEffect(() => {
const fetchAllocations = async () => {
try {
const response = await axios.get('http://localhost:8080/user/getallocations');
setAllocations(response.data);
setFilteredAllocations(response.data);
} catch (error) {
console.error('Error fetching allocations:', error);
}
};
fetchAllocations();
}, []);
const handleFilterChange = (e) => {
const value = e.target.value;
setSearchHostel(value);
const filtered = allocations.filter((allocation) =>
allocation.hostelName.toLowerCase().includes(value.toLowerCase())
);
setFilteredAllocations(filtered);
};
return (

<h2>Room Allocations</h2>
main*
Launchpad
Java: Ready
Ln 39, Col 3
Spaces: 2
UTF-8
LF
JavaScript

Testing

5.1 User registration:

S.No	Test Case	Objective	Input	Expected Outcome	Result
1.1	Valid User Registration	Test the registration functionality with valid user details.	Full name, email, password etc.	User should be registered successfully, and the system should redirect to the login page.	Pass
1.2	Invalid Email Format	Test the registration functionality with an invalid email format.	Email: invalid-email.com, valid full name, password	Error message: "Please enter a valid email address."	Pass
1.3	Missing Mandatory Fields	Test the registration functionality with missing fields (e.g., missing full name).	Leave the full name field blank.	Error message: "Full name is required." •	Pass

5.2 User login:

S.No	Test Case	Objective	Input	Expected Output	Result
2.1	Valid Login	Test the login functionality with valid credentials.	Registered email and password.	User should be successfully logged in and redirected to the dashboard.	Pass
2.2	Invalid Password	Test the login functionality with an incorrect password.	Registered email and incorrect password.	Error message: "Invalid credentials."	Pass

5.3 Room allocation:

S.No	Test Case	Objective	Input	Expected Outcome	Result
3.1	Valid Room Allocation	Test the room allocation functionality by assigning a room to a student.	Student registration details with valid room availability.	The student should be assigned a room, and the details should be updated in the system.	Pass

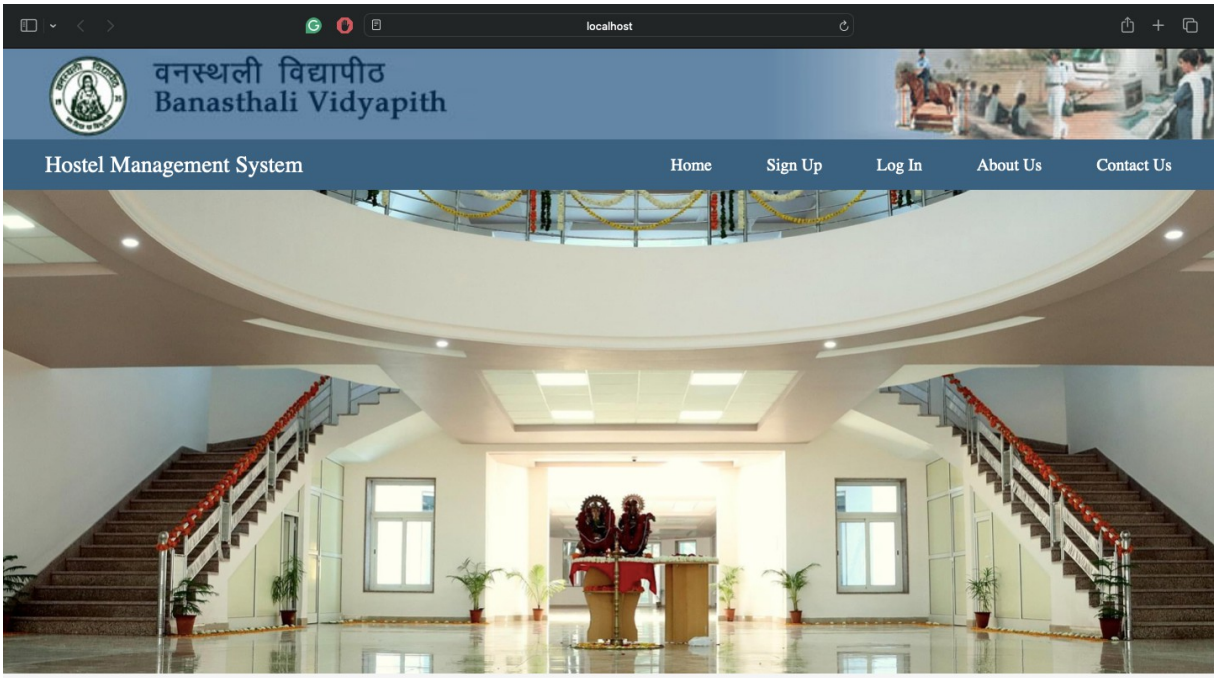
5.4 Feedback submission:

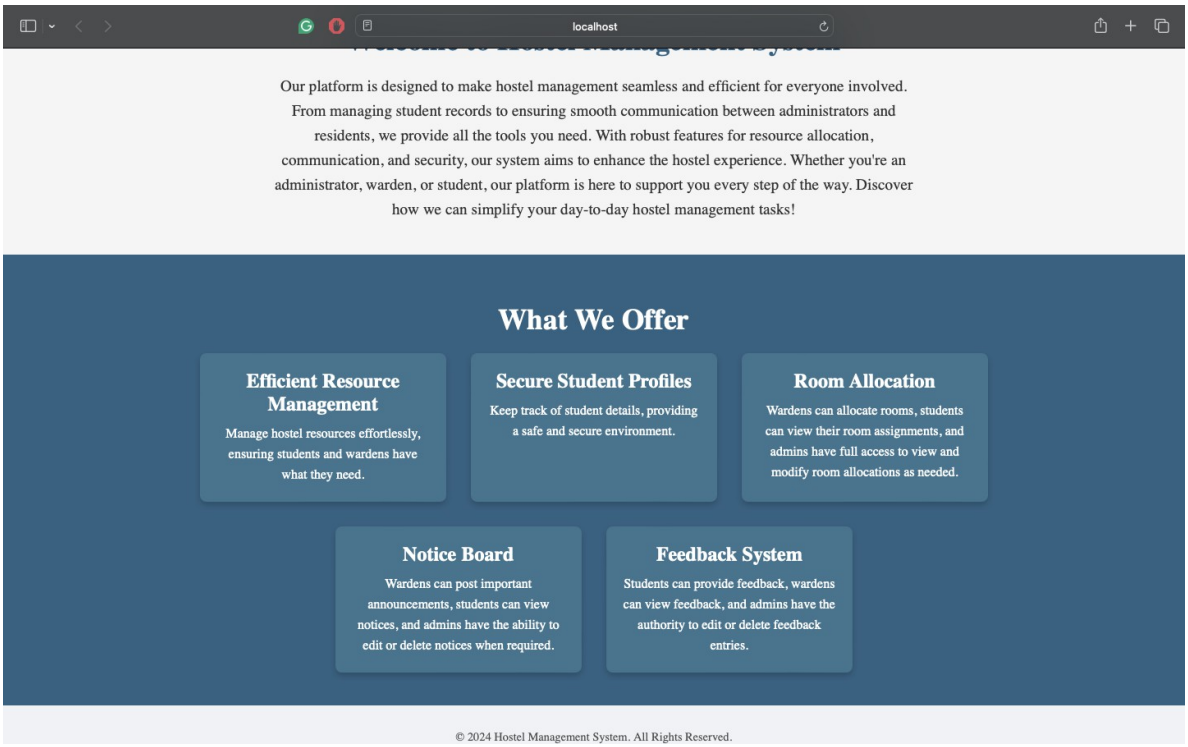
S.No	Test Case	Objective	Input	Expected Outcome	Result
4.1	Valid Feedback Submission	Test the feedback submission functionality for valid input.	Feedback ratings for cleanliness, food, facilities, staff, and safety.	Feedback should be submitted successfully and stored in the database.	Pass
4.2	Incomplete Feedback Submission	Test the feedback submission functionality with missing fields.	Leave the "Cleanliness" rating field blank.	Error message: "This field is required."	Pass

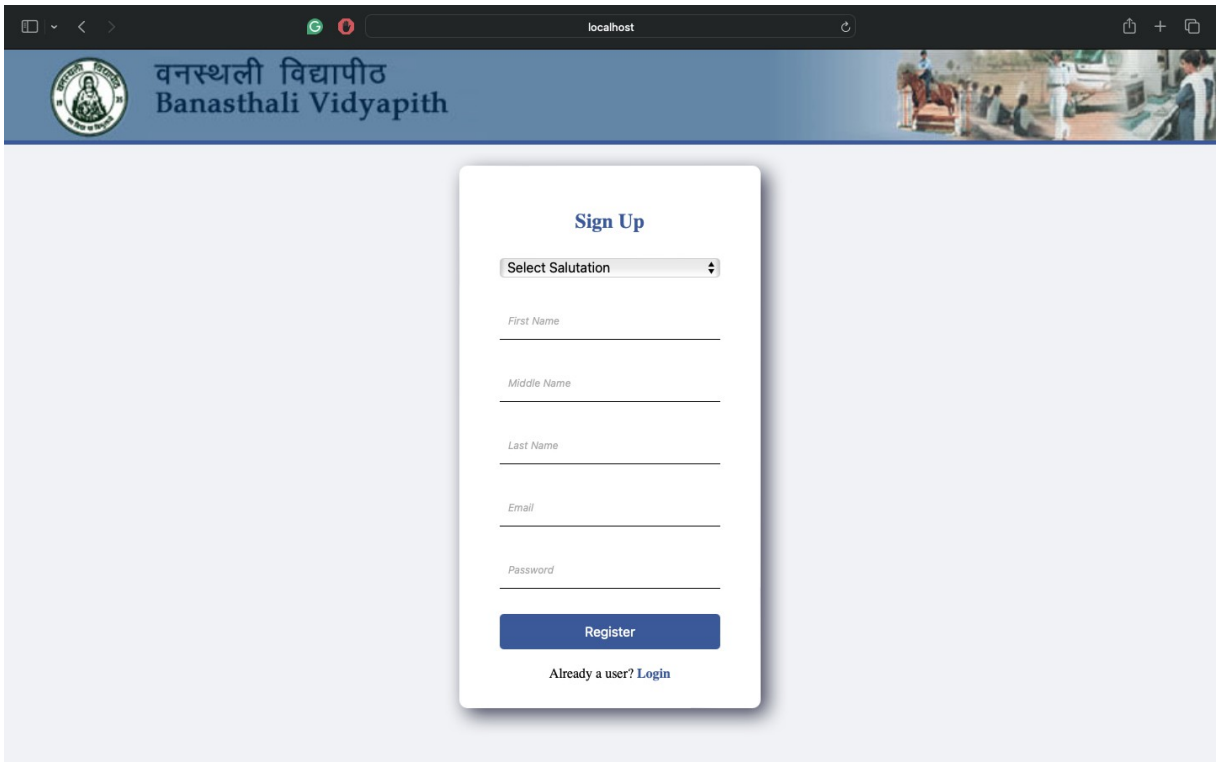
5.5 Notice board:

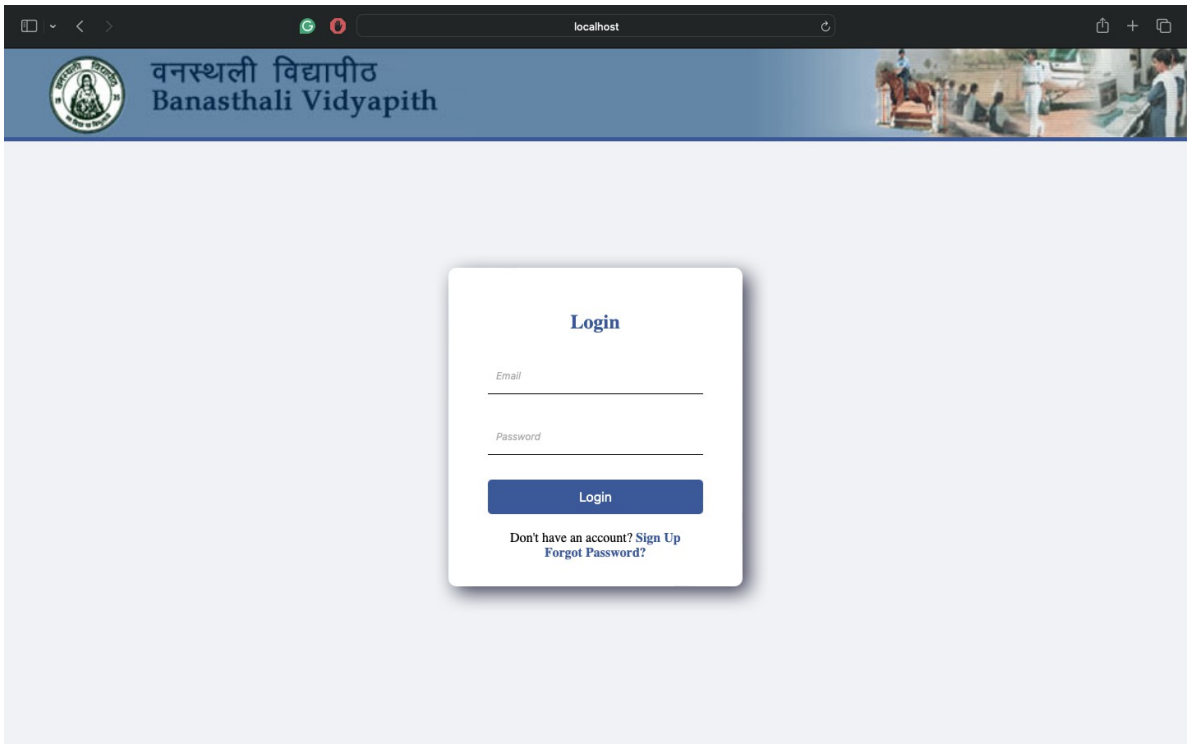
S.No	Test Case	Objective	Input	Expected Outcome	Result
5.1	View Notice Board	Test the ability to view notices posted on the notice board.	Click on the "Notice Board" section.	The system should display the list of all active notices.	Pass
5.2	Post a New Notice	Test the ability to post a new notice by the warden.	Warden enters a new notice and clicks "Add Notice".	The notice should appear on the notice board for all users.	Pass

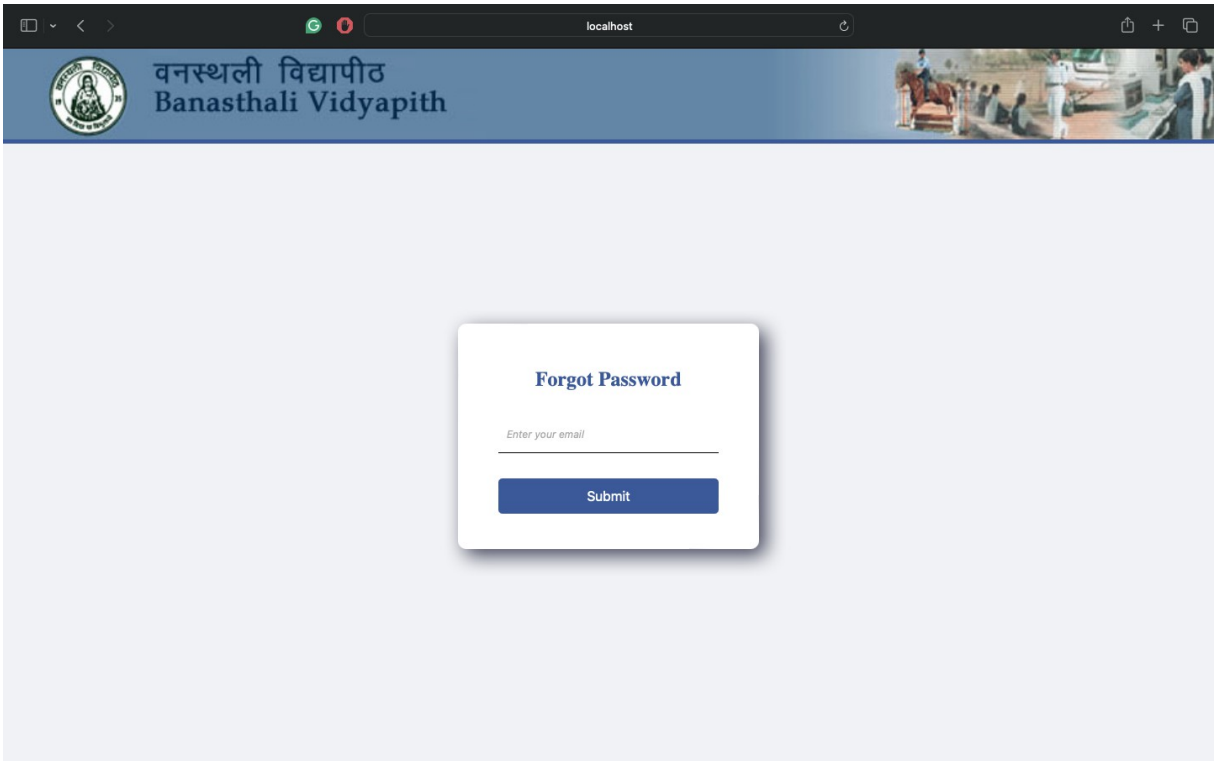
User Interface

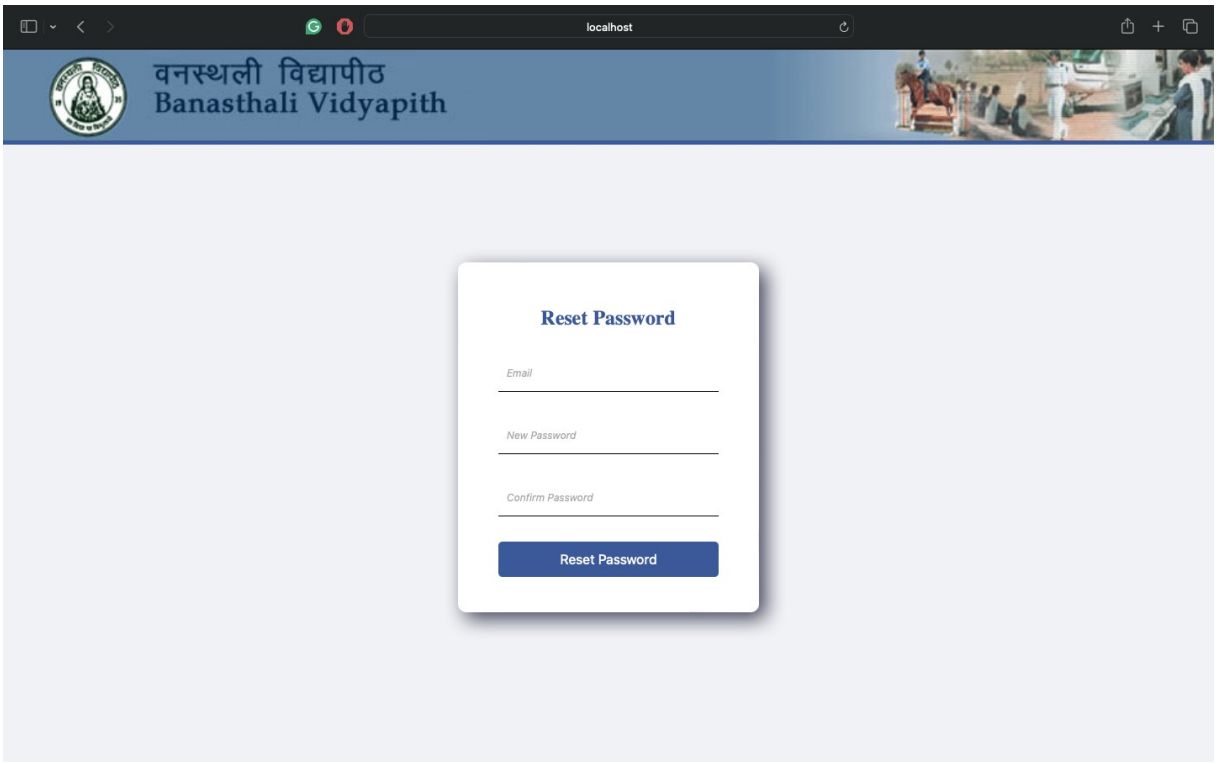


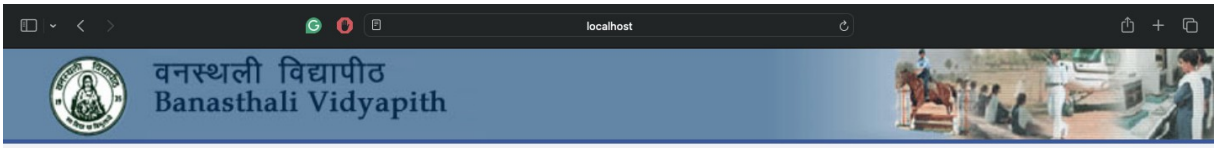



localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Sign Up
Select Salutation
First Name
Middle Name
Last Name
Email
Password
Register
Already a user? Login


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Login
Email
Password
Login
Don't have an account? Sign Up
Forgot Password?


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Forgot Password
Enter your email
Submit


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Reset Password
Email
New Password
Confirm Password
Reset Password



About Us

Banasthali Vidyapith

Banasthali Vidyapith is a premier residential women's higher education institute in India, offering an integrated education system from primary to Ph.D. level. Founded on October 6, 1935, by Smt. Ratan Shastri and Pandit Hiralal Shastri, it arose from the desire to honor their late daughter, Shantabai, who had the potential to champion women's causes. Over the past 75 years, Banasthali has become a National Centre for women's education, emphasizing holistic personality development through its Five-fold Educational Programme (Panchmukhi Shiksha), which focuses on Physical, Practical, Aesthetic, Moral, and Intellectual growth. The rural setting of Banasthali promotes simple living, self-reliance, and tolerance, fostering an environment conducive to integrated personality development. The institute, which has been awarded 'A' Grade by the National Assessment and Accreditation Council (NAAC), creatively restructures its courses to cover traditional subjects and emerging fields up to the doctoral level. In recognition of its contributions to women's education, Banasthali received the SANTBAL Award in 2000, reflecting its commitment to empowering women for leadership roles.

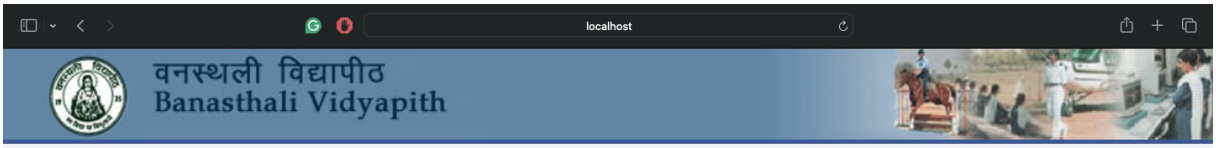
Our Mission

The Hostel Management System for Banasthali Vidyapith is designed to streamline and automate the various administrative tasks involved in managing a hostel. This comprehensive, user-friendly platform simplifies the process of hostel and room allotment, ensuring efficient allocation based on availability and student preferences. It features an integrated notice board, enabling administrators to communicate important updates and announcements effectively, ensuring that students are always informed about critical information regarding their accommodation. The system also facilitates the collection of student feedback, allowing for continuous improvement based on their experiences and suggestions, thus fostering a culture of open communication and responsiveness. Additionally, it includes modules for managing student and staff records, making hostel management more efficient and transparent. By automating routine tasks such as attendance tracking, maintenance requests, and fee payments, the system alleviates administrative burdens, allowing staff to focus on providing quality service to residents.

We aim to create a conducive living environment for students, ensuring their comfort, safety, and well-being while residing in the hostel. Our focus is on facilitating seamless communication and feedback between students and hostel management, enhancing the overall hostel experience. We are committed to providing a supportive community where students can thrive academically and personally, equipping them with the necessary resources and assistance to succeed. Through this innovative system, we strive to promote a sense of belonging and empowerment among students, encouraging them to take an active role in their hostel life while fostering a spirit of collaboration and mutual respect. Ultimately, our goal is to enhance the quality of life for all residents, making Banasthali Vidyapith a home away from home.

Contact Us

For any inquiries or feedback, please reach out to us via the contact page. We appreciate your input and are committed to enhancing your hostel experience.



Contact Us

If you have any inquiries or feedback, please feel free to reach out using the form below. We are here to help and improve your experience.

Name

Enter your name

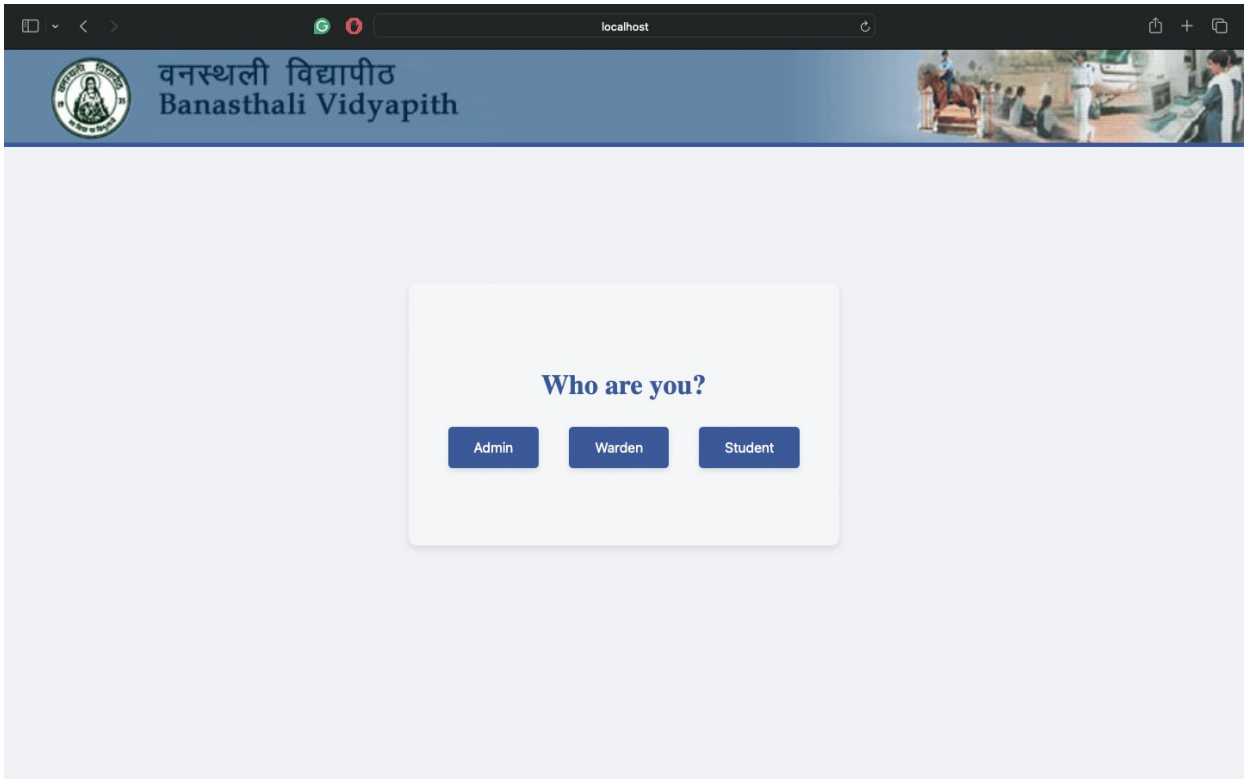
Email

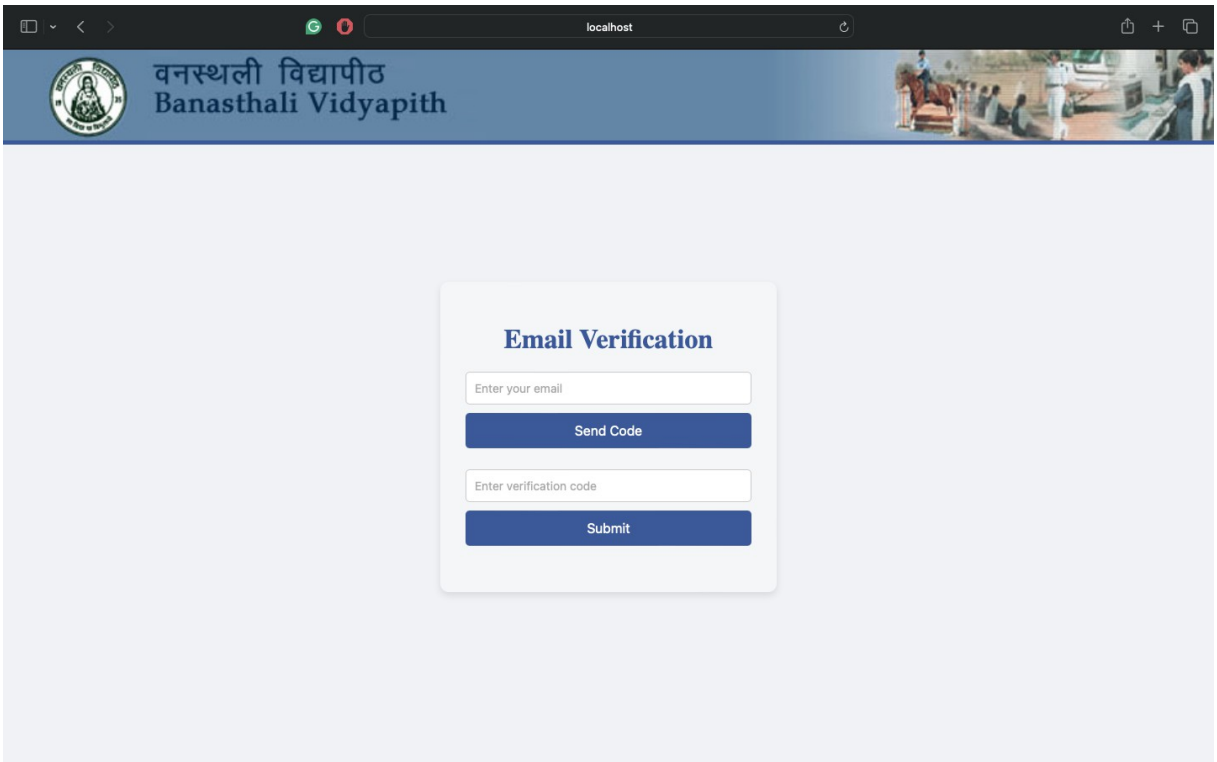
Enter your email

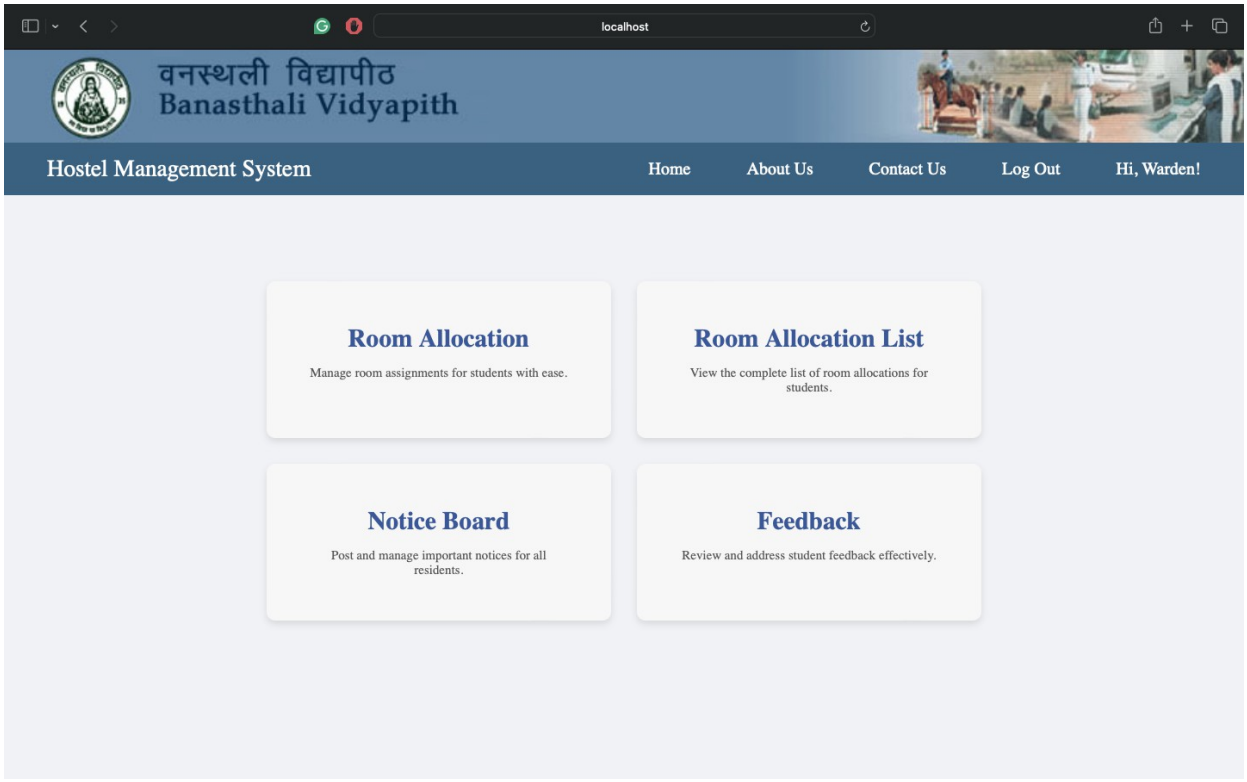
Message

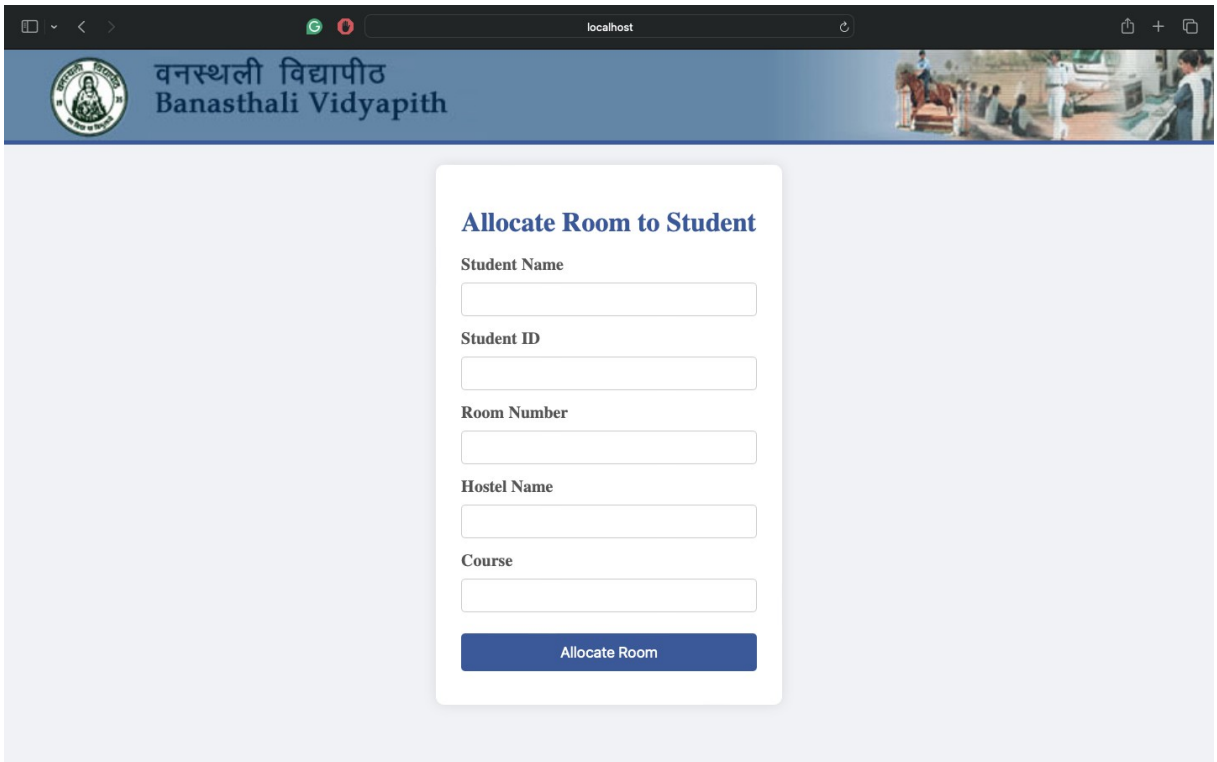
Enter your message

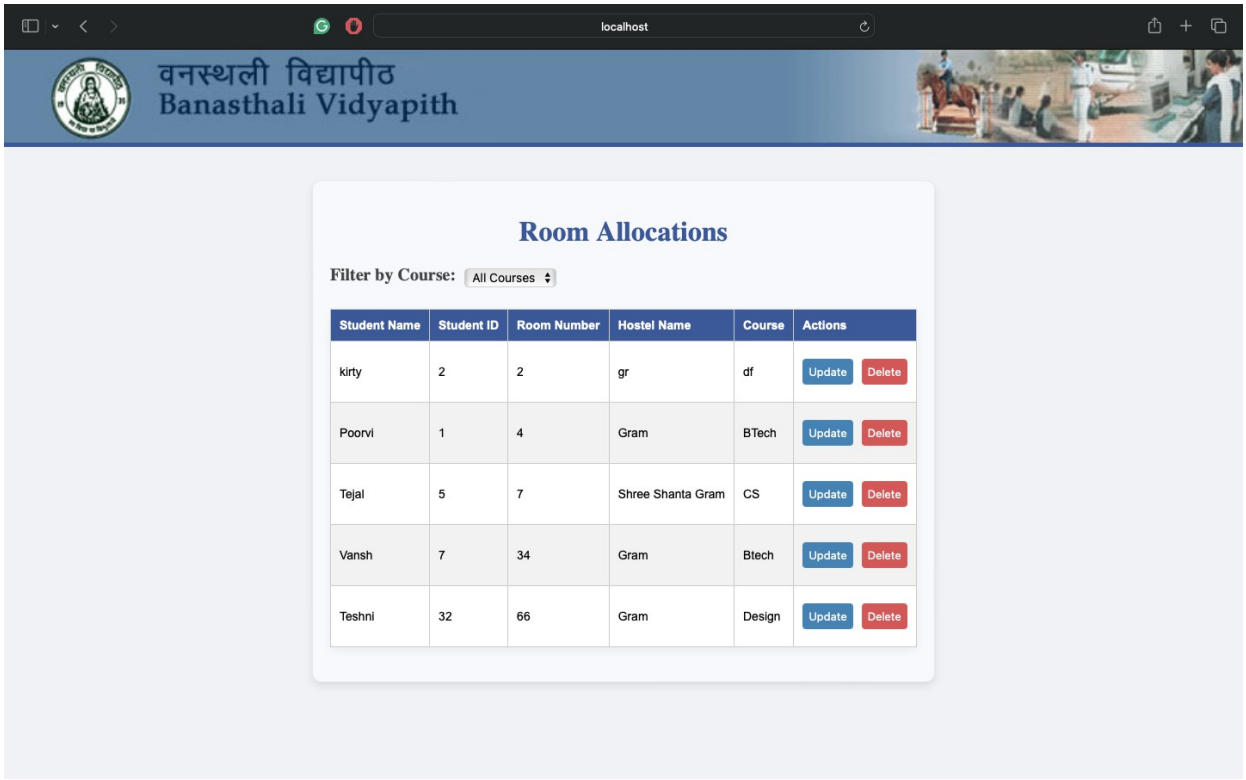
Send Message

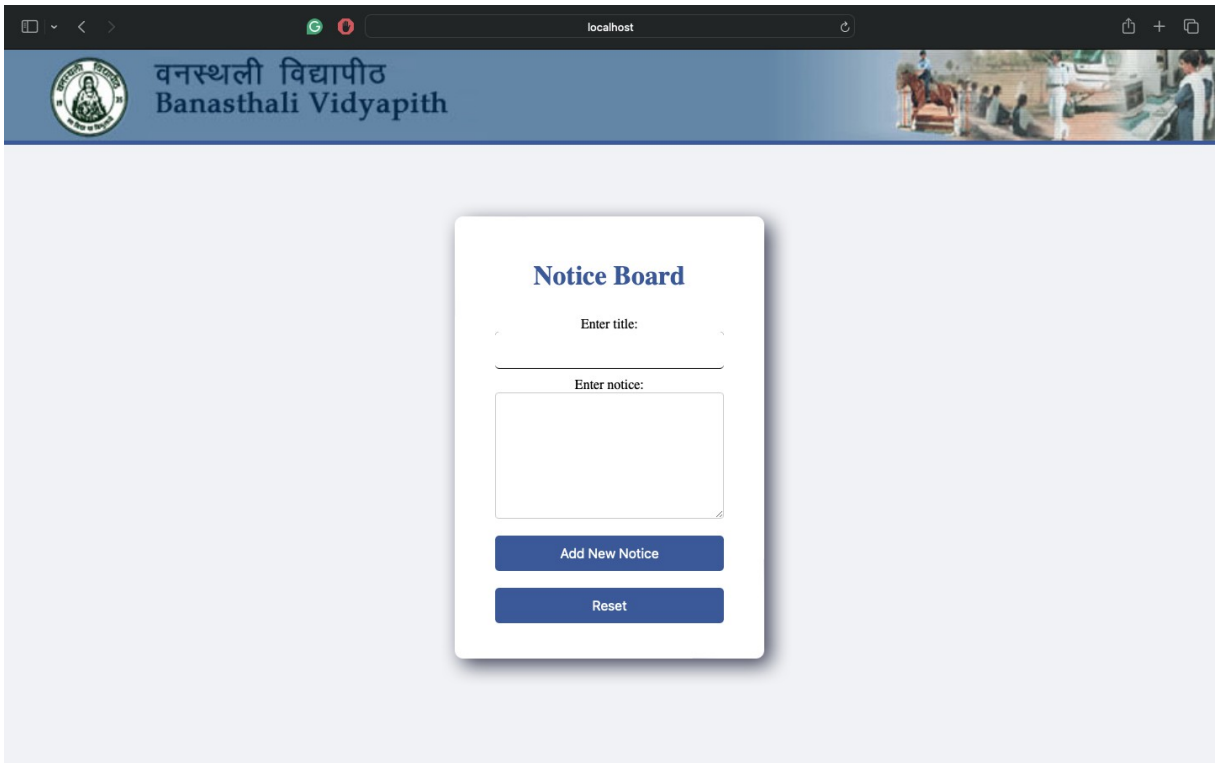

localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Who are you?
Admin
Warden
Student

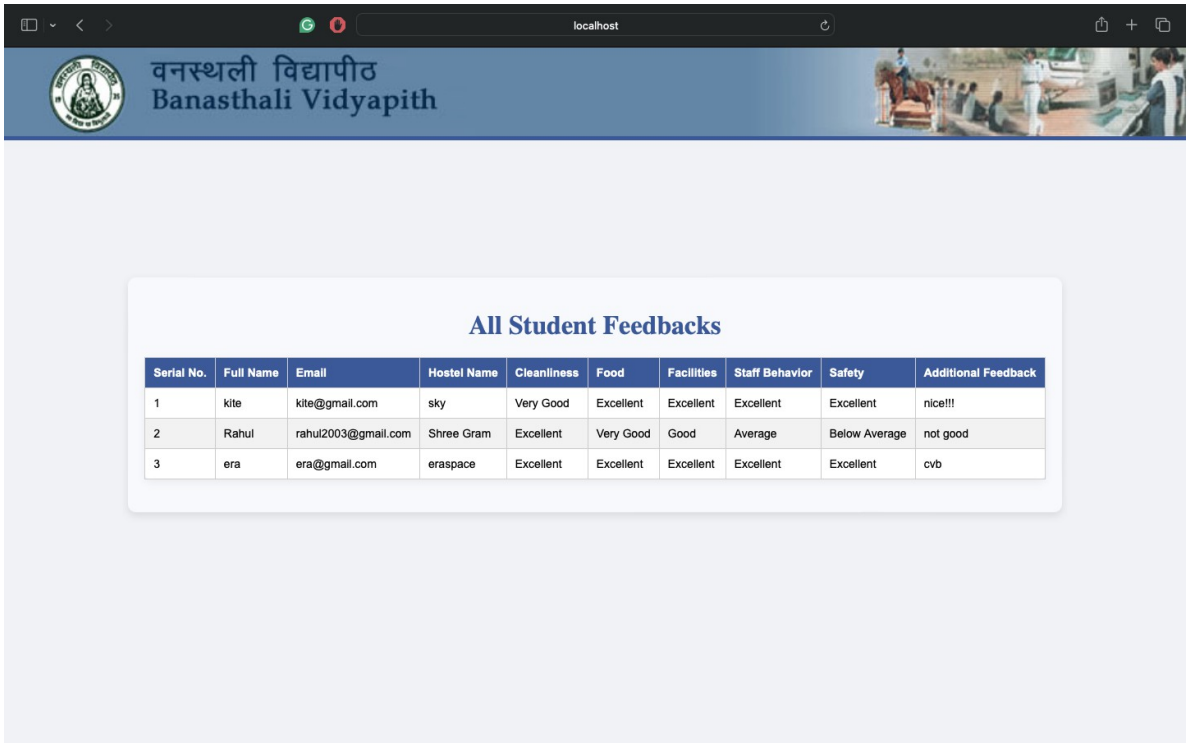

localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Email Verification
Enter your email
Send Code
Enter verification code
Submit

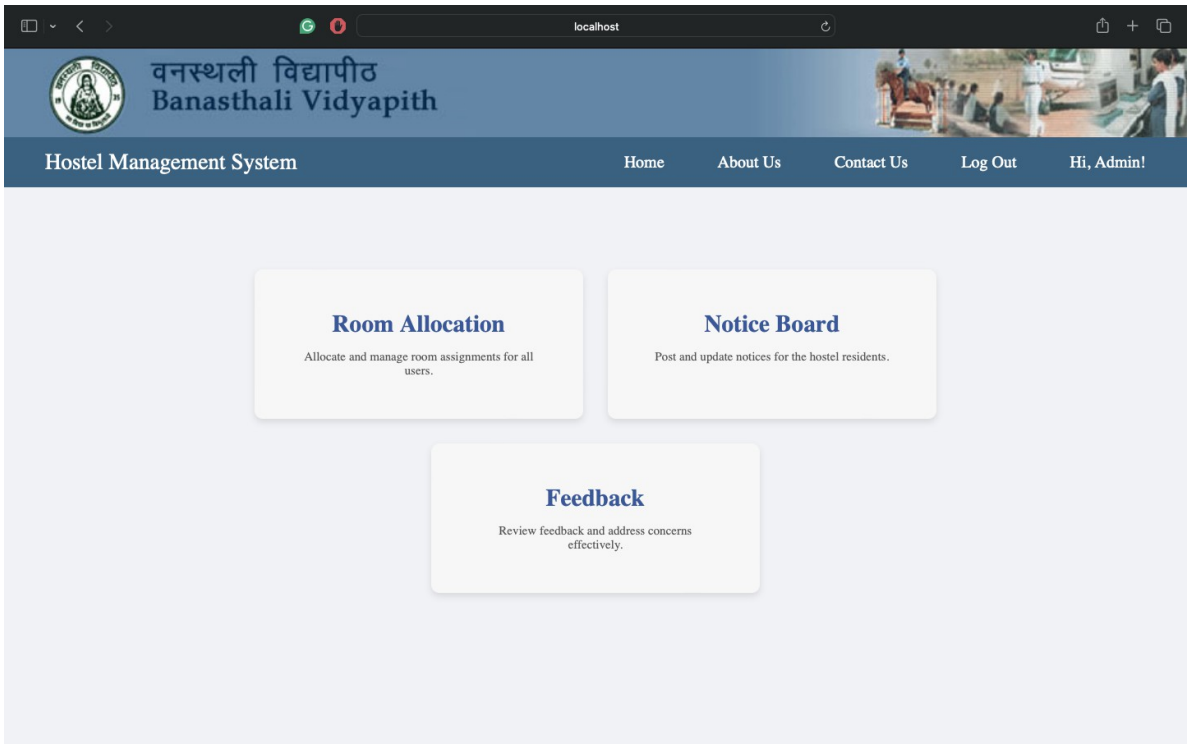

localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Hostel Management System
Home
About Us
Contact Us
Log Out
Hi, Warden!
Room Allocation
Manage room assignments for students with ease.
Room Allocation List
View the complete list of room allocations for students.
Notice Board
Post and manage important notices for all residents.
Feedback
Review and address student feedback effectively.

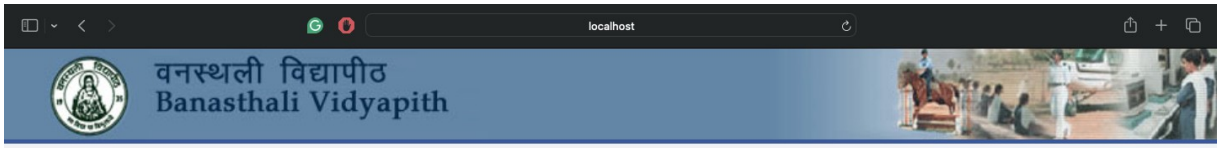

localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Allocate Room to Student
Student Name
Student ID
Room Number
Hostel Name
Course
Allocate Room


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Room Allocations
Filter by Course: All Courses
Student Name | Student ID | Room Number | Hostel Name | Course | Actions
kirty | 2 | 2 | gr | df | Update Delete
Poorvi | 1 | 4 | Gram | BTech | Update Delete
Tejal | 5 | 7 | Shree Shanta Gram | CS | Update Delete
Vansh | 7 | 34 | Gram | Btech | Update Delete
Teshni | 32 | 66 | Gram | Design | Update Delete


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Notice Board
Enter title:
Enter notice:
Add New Notice
Reset


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
All Student Feedbacks
Serial No. | Full Name | Email | Hostel Name | Cleanliness | Food | Facilities | Staff Behavior | Safety | Additional Feedback
1 | kite | kite@gmail.com | sky | Very Good | Excellent | Excellent | Excellent | Excellent | nice!!!
2 | Rahul | rahul2003@gmail.com | Shree Gram | Excellent | Very Good | Good | Average | Below Average | not good
3 | era | era@gmail.com | eraspace | Excellent | Excellent | Excellent | Excellent | Excellent | cvb

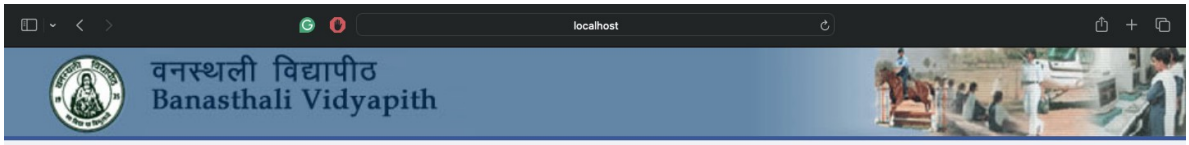

localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Hostel Management System
Home
About Us
Contact Us
Log Out
Hi, Admin!
Room Allocation
Allocate and manage room assignments for all users.
Notice Board
Post and update notices for the hostel residents.
Feedback
Review feedback and address concerns effectively.



Room Allocations

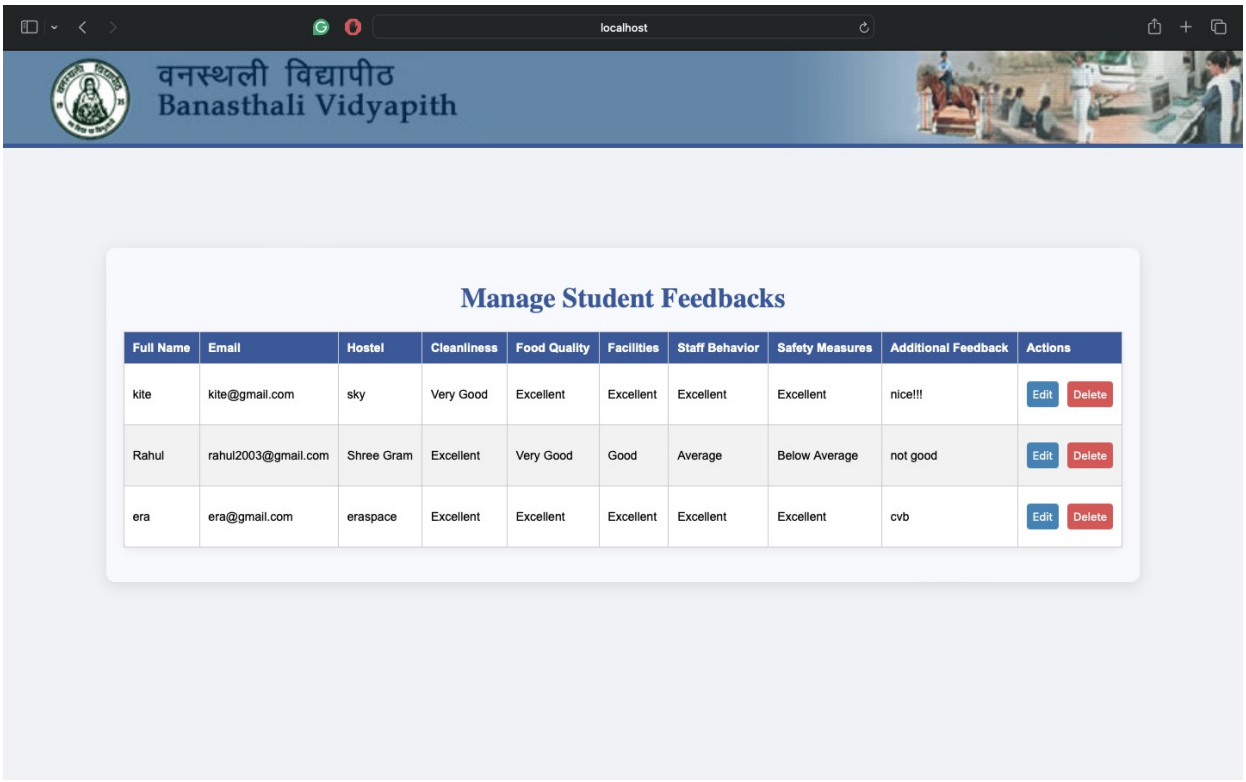
Filter by Hostel Name: Enter hostel name...

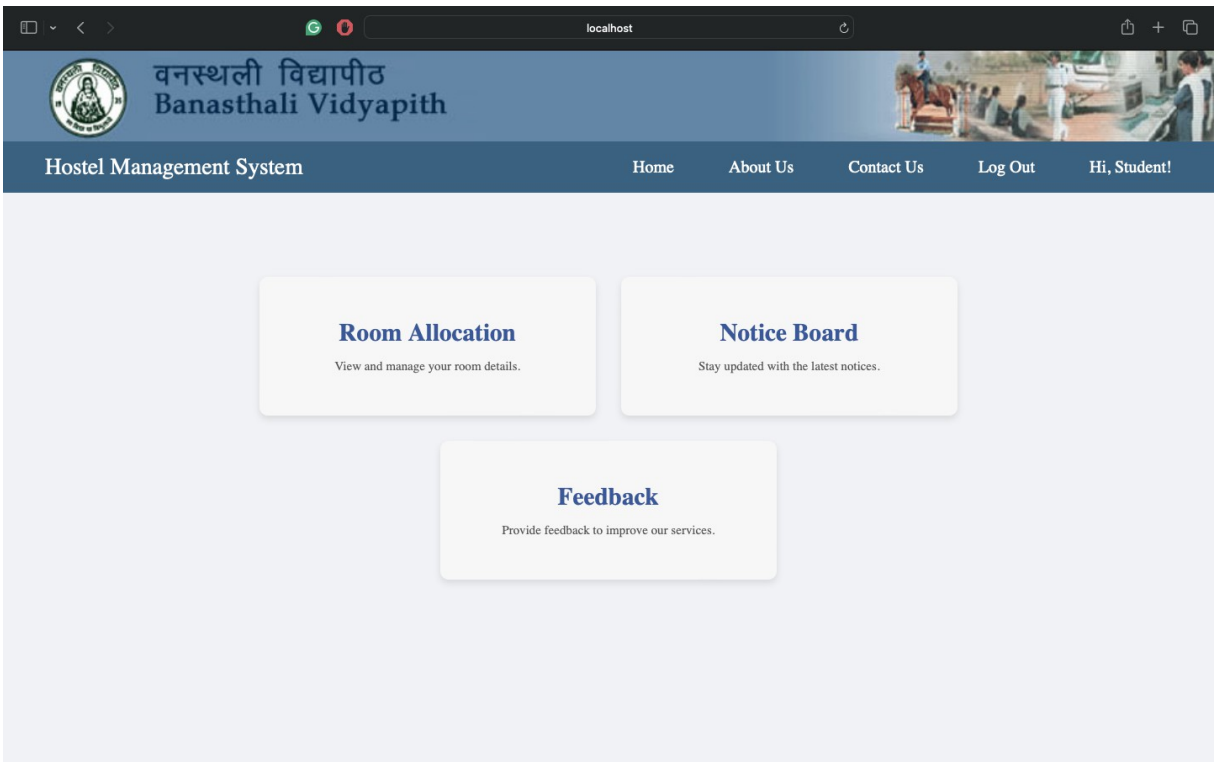
Student Name	Student ID	Room Number	Hostel Name	Course
Poorvi	1	4	Gram	BTech
Tejal	5	7	Shree Shanta Gram	CS
Teshni	32	66	Gram	Design
kirty	2	2	gr	df
Vansh	7	34	Gram	Btech

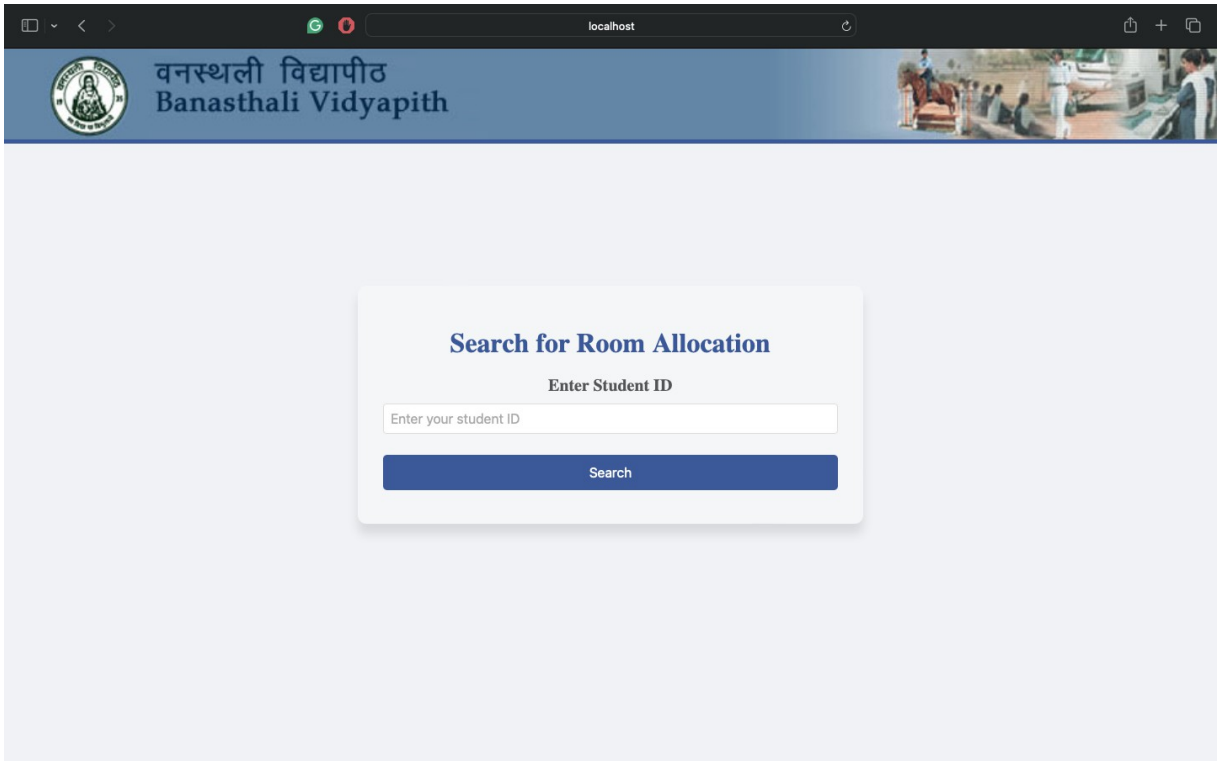


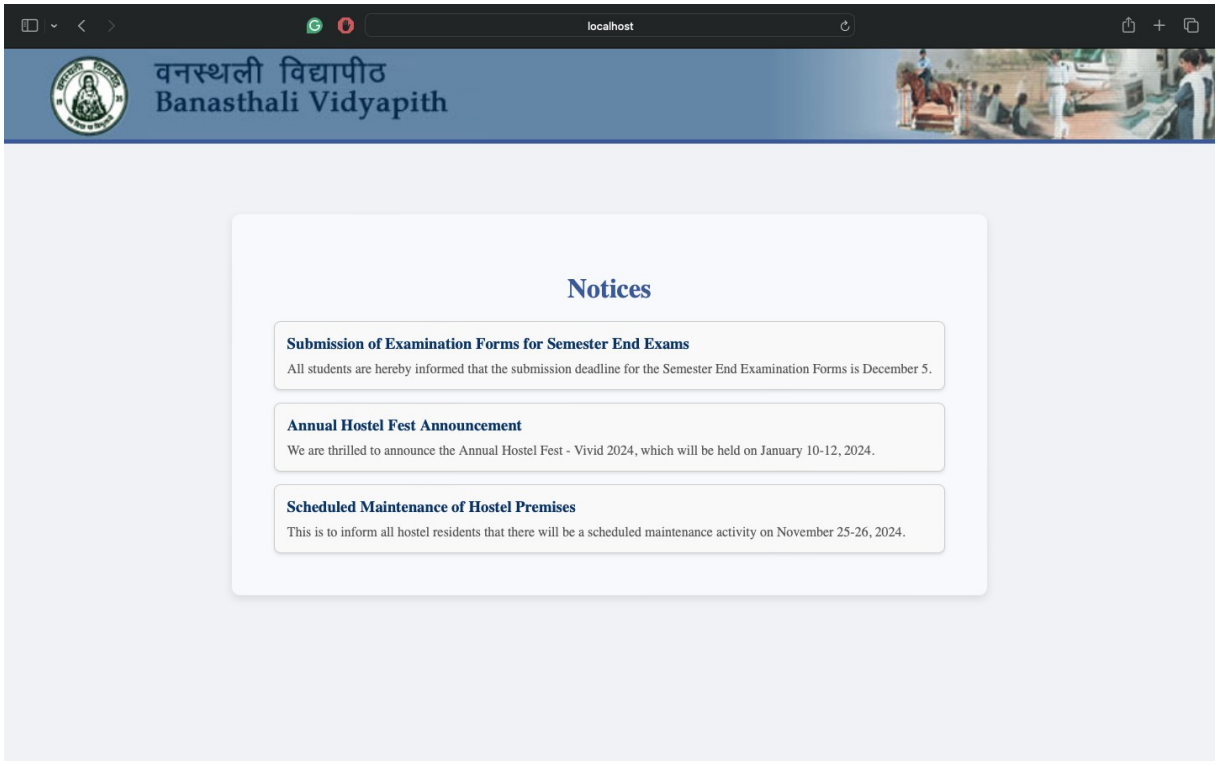
Manage Notices

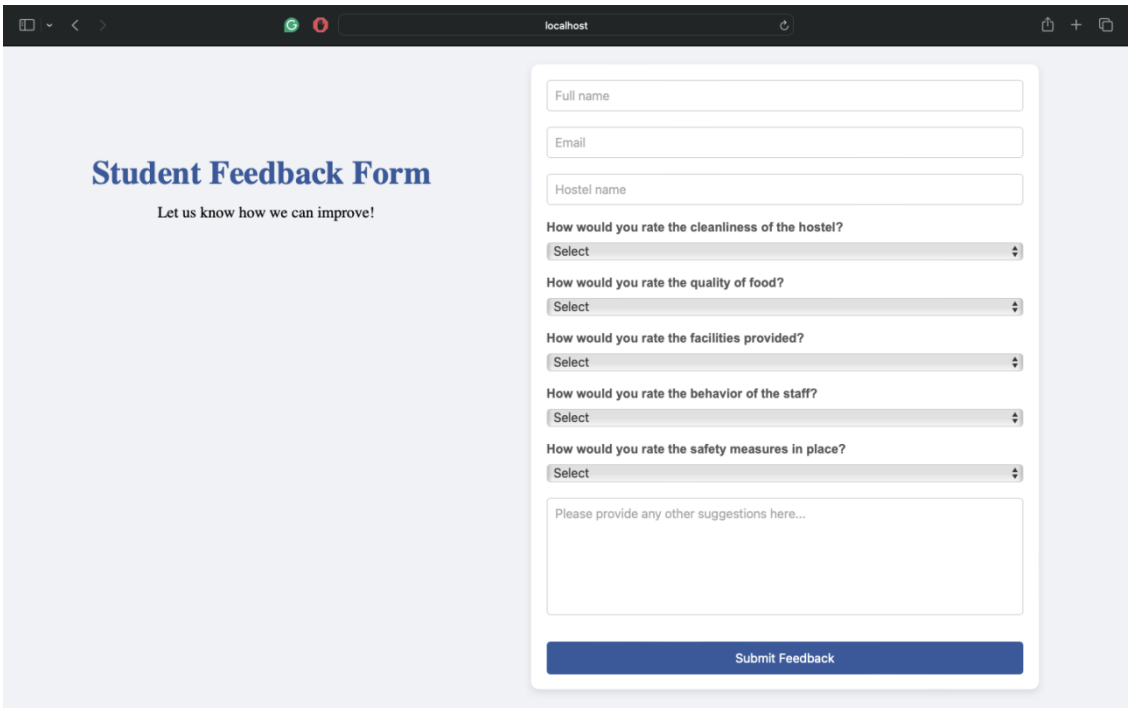
Title	Details	Actions
Submission of Examination Forms for Semester End Exams	All students are hereby informed that the submission deadline for the Semester End Examination Forms is December 5.	Edit Delete
Annual Hostel Fest Announcement	We are thrilled to announce the Annual Hostel Fest - Vivid 2024, which will be held on January 10-12, 2024.	Edit Delete
Scheduled Maintenance of Hostel Premises	This is to inform all hostel residents that there will be a scheduled maintenance activity on November 25-26, 2024.	Edit Delete


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Manage Student Feedbacks
Full Name | Email | Hostel | Cleanliness | Food Quality | Facilities | Staff Behavior | Safety Measures | Additional Feedback | Actions
kite | kite@gmail.com | sky | Very Good | Excellent | Excellent | Excellent | Excellent | nice!!! | Edit Delete
Rahul | rahul2003@gmail.com | Shree Gram | Excellent | Very Good | Good | Average | Below Average | not good | Edit Delete
era | era@gmail.com | eraspace | Excellent | Excellent | Excellent | Excellent | Excellent | cvb | Edit Delete


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Hostel Management System
Home
About Us
Contact Us
Log Out
Hi, Student!
Room Allocation
View and manage your room details.
Notice Board
Stay updated with the latest notices.
Feedback
Provide feedback to improve our services.


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Search for Room Allocation
Enter Student ID
Enter your student ID
Search


localhost
वनस्थली विद्यापीठ
Banasthali Vidyapith
Notices
Submission of Examination Forms for Semester End Exams
All students are hereby informed that the submission deadline for the Semester End Examination Forms is December 5.
Annual Hostel Fest Announcement
We are thrilled to announce the Annual Hostel Fest - Vivid 2024, which will be held on January 10-12, 2024.
Scheduled Maintenance of Hostel Premises
This is to inform all hostel residents that there will be a scheduled maintenance activity on November 25-26, 2024.


localhost
Student Feedback Form
Let us know how we can improve!
Full name
Email
Hostel name
How would you rate the cleanliness of the hostel?
Select
How would you rate the quality of food?
Select
How would you rate the facilities provided?
Select
How would you rate the behavior of the staff?
Select
How would you rate the safety measures in place?
Select
Please provide any other suggestions here...
Submit Feedback

Appendices

Appendix A: Glossary

Admin: The administrator responsible for managing the hostel system.

API (Application Programming Interface): A set of tools and protocols for building software applications.

Authentication: The process of verifying the identity of a user.

Authorization: The process of determining the access level or permissions a user has.

GDPR (General Data Protection Regulation): A regulation in EU law on data protection and privacy.

GUI (Graphical User Interface): The visual part of a software application through which users interact with the program.

HMS (Hostel Management System): The software system designed to manage hostel operations.

HTTP (Hypertext Transfer Protocol): The protocol used for transmitting web pages over the Internet.

RBAC (Role-Based Access Control): A method of restricting system access based on roles.

Appendix B: Tools and Technologies

This appendix provides an overview of the tools and technologies used in the development of the **Hostel Management System**. Each component of the project has been implemented using specialised tools to ensure efficiency, scalability, and maintainability.

- **Frontend**

Technology Used: **React**

- React is a JavaScript library used for building dynamic and interactive user interfaces.
- **Features**:
  - Component-based architecture for reusability.

        - Virtual DOM for efficient updates and rendering.
        - Easy integration with APIs to fetch and display data.
    - **Usage in the Project**:
        - Design of the admin dashboard for managing hostel operations.
        - User interfaces for room allocation, student management, and report generation.
        - Real-time form validation and interactive input fields.

- **Backend**

Technology Used: **Spring Boot with Java**

    - Spring Boot is a powerful Java-based framework for building robust backend services.
    - **Features**:
        - Embedded server for easy deployment.
        - Extensive support for RESTful APIs.
        - Integration with MongoDB for data persistence.
        - Security features for user authentication and authorisation.
    - **Usage in the Project**:
        - Implementation of REST APIs for CRUD operations (e.g., adding, updating, and deleting student and room data).
        - Business logic for room allocation functionality.
        - Integration with the frontend for seamless data exchange.

- **Database**

Technology Used: **MongoDB**

    - MongoDB is a NoSQL database known for its flexibility and scalability.
    - **Features**:
        - Schema-less design, allowing rapid development and changes.

- High performance for handling large datasets.
- Built-in support for horizontal scaling and replication.

- **Usage in the Project**:
  - Storing student details, room allocations, and warden information.
  - Querying data efficiently to support real-time operations.
  - Use of collections to organize data into logical groups.

- **Integrated Development Environment (IDE)**

Tool Used: **Visual Studio Code (VS Code)**

- VS Code is a lightweight and powerful source code editor with support for multiple programming languages and extensions.
- **Features**:
  - IntelliSense for intelligent code completion.
  - Integrated terminal for executing commands and scripts.
  - Support for version control systems like Git.
  - Extensions for Java, Spring Boot, React, and MongoDB.
- **Usage in the Project**:
  - Writing and debugging frontend and backend code.
  - Managing project files and dependencies.
  - Collaboration using Git for version control.

References

1. **Books & Academic Resources**:
    - Sommerville, I. (2015). *Software Engineering* (10th Edition). Pearson Education.
        - Referenced for understanding software development lifecycle models and best practices for system design.
    - Elmasri, R., & Navathe, S. B. (2016). *Fundamentals of Database Systems* (7th Edition). Pearson Education.
        - Used for designing and normalizing the database schema.
2. **Online Documentation**:
    - Spring Boot Documentation: https://spring.io/projects/spring-boot
        - Referenced for implementing the backend framework and understanding dependency injection and RESTful APIs.
    - MongoDB Documentation: https://www.mongodb.com/docs/
        - Used for database management and CRUD operations.
3. **APIs and Libraries**:
    - Postman: https://www.postman.com/
        - Used for testing API endpoints during the development process.
    - Bootstrap: https://getbootstrap.com/
        - Used for styling and creating responsive user interfaces.

4. **Technologies and Tools**:
    - Java Development Kit (JDK): https://www.oracle.com/java/technologies/javase-jdk15-downloads.html
        - Used for programming the application backend.
    - IntelliJ IDEA: https://www.jetbrains.com/idea/
        - Integrated Development Environment (IDE) used for coding the application.
    - GitHub: https://github.com/
        - Version control and collaboration tool used to manage the project source code.
5. **Tutorials and Online Courses**:
    - Colt Steele, *The Web Developer Bootcamp 2023* (Udemy).
        - Provided foundational knowledge for building the frontend and backend of the project.
    - Spring Framework Masterclass: https://www.udemy.com/course/spring-framework-tutorial/
        - Used for learning advanced concepts in Spring Boot.
6. **Articles and Blogs**:
    - GeeksforGeeks: *Hostel Management System in Java*. https://www.geeksforgeeks.org/
        - Provided inspiration for database schema design and application architecture.
    - Baeldung: *Building a REST API with Spring Boot*. https://www.baeldung.com/
        - Referenced for implementing RESTful API endpoints.

7. **Research Papers**:

- Wang, Y., & Zhao, J. (2018). *Web-Based Hostel Management Systems: A Case Study Approach*. International Journal of Computer Science and Network Security, 18(6), 45-52.
  - Insights into system features and database design.
- Agrawal, S., & Gupta, P. (2019). *Enhancing User Experience in Management Systems Using Agile Development Methods*. Journal of Software Engineering and Applications, 12(3), 67-75.
  - Referenced for adopting Agile methodologies in project development.