SILIGURI INSTITUTE OF TECHNOLOGY

Minor Project: MCAN381

EXAM HALL MANAGEMENT

By

Debratan Mondal
33671023022

Subhra Sarkar
33671023038

Pallab Dutta
33671023026

Gayetri Sarkar
33671023025

Under the guidance of Prof. S.K Bhattacharya

Submitted to the Department of **Master of Computer Application** in partial fulfillment of the requirements for the award of the degree MCA.

Year of Submission: 2025

Siliguri Institute of Technology

P.O. SUKNA, SILIGURI, DIST. DARJEELING, PIN: 734009

Tel: (0353)2778002/04, Fax: (0353) 2778003

DECLARATION

-

This is to certify that Report entitled "EXAM HALL MANAGEMENT" which is submitted by me in partial fulfillment of the requirement for the award of degree MCA at Siliguri Institute of Technology under Maulana Abul Kalam Azad University of Technology, West Bengal. We took the help of other materials in our dissertation which have been properly acknowledged. This report has not been submitted to any other Institute for the award of any other degree.

Date: 22/01/25

Student Name: Subhra Sarkar

Roll No: 33671023038

Signature:

Student Name: Pallab Dutta

Roll No: 33671023026

Signature:

Student Name: Debratan Mondal

Roll No: 33671023022

Signature:

Student Name: Gayetri Sarkar

Roll No: 33671023025

Signature:

CERTIFICATE

This is to certify that the project report entitled Exam Hall Management submitted to Department of MCA of Siliguri Institute of Technology in partial fulfilment of the requirement for the award of the degree of MCA during the academic year 2022-23, is a bonafide record of the project work carried out by them under my guidance and supervision.

Student Name: Subhra Sarkar
Student's Registration No: 233360510037
Student’s Roll No: 33671023038

Student Name: Debratan Mondal
Student's Registration No: 233360510021
Student’s Roll No: 33671023022

Student Name: Pallab Dutta
Student's Registration No: 233360510025
Student’s Roll No: 33671023026

Student Name: Gayetri Sarkar
Student's Registration No: 233360510024
Student’s Roll No: 33671023025

....
Signature of Project Guide
<Name of the Guide>

....
Signature of the HOD
Department of MCA

Acknowledgment

We express our heartfelt gratitude to Mr. S.K. Bhattacharya, our guide, for his invaluable support and guidance throughout this project. His expertise and encouragement were key to its successful completion.

We also thank the faculty members and staff for their assistance, and our family and friends for their constant support and motivation.

This project would not have been possible without their contributions, and we are sincerely grateful to all.

Signature of all the group members with date

-

1.

2.

3.

4.

Table of Contents

1. Abstract

This project focuses on developing a user-friendly Exam Hall Management System, a website designed to streamline the process of seat allotment and arrangement in educational institutions. The system enables users, primarily students and administrators, to view the seating arrangement for exams in a specific building using student details such as college code, branch, semester, subject, date, time, roll number start, and roll number end. This information is displayed on a dedicated page called the "Exam Schedule."

The website comprises several interconnected modules, including the Homepage, Login/Register Page, Room Management Page, and Seat Allocation Page. The Homepage serves as a gateway with a welcome message and navigation bar. The Login/Register Page ensures secure access to the system by providing login and registration features.

In the Room Management Page, users can add building names and manage room details by specifying room numbers, floor levels, and seating capacity, including the number of columns. The Seat Allocation Page allows users to select branch details, exam dates, and times to view the seat allocation. This page provides critical insights such as seat statistics, student statistics, and a graphical visualization of the room layout.

The project aims to simplify exam hall management by providing an efficient, centralized platform for students and administrators. It enhances accuracy, reduces manual workload, and ensures smooth coordination of exam arrangements. The system is built with scalability and flexibility in mind, offering scope for future improvements, such as integrating advanced analytics or multi-institution support.

2. Introduction

Organizing exams is an essential task for educational institutions, and managing seating arrangements is a key part of this process. The **"Exam Hall Seat Management System"** is developed to simplify and streamline this task by automating seat allocation, ensuring fairness, and reducing manual errors. This system provides an efficient way to handle seating arrangements, making the examination process smoother for both staff and students.

2.1. Problem statement:

In many institutions, assigning seats for exams is done manually, which takes a lot of time and can lead to mistakes. Sometimes, students are given the same seat by error, which causes confusion. The arrangements may also not be fair, making it easier for students to cheat. If changes need to be made at the last minute, it becomes difficult and stressful to adjust the seating plan. Often, the available space in exam halls is not used properly, which wastes resources. These problems make the process harder for staff and create confusion for students during exams.

2.2. Goal:

The goal of the **Exam Hall Seat Management System** is to provide an automated and user-friendly solution for managing exam seating arrangements. It aims to allocate seats accurately, optimize the use of available resources, and enable quick adjustments to seating plans when required. By addressing these challenges, the system ensures a more organized and efficient examination process, benefiting both administrators and students.

3. System analysis

3.1. Identification of Need :

The **Exam Hall Seat Management System** is necessary to address several challenges faced by educational institutions in managing exam seating arrangements. Below are the key reasons highlighting the need for this project:

1. **Error-Free Seat Allocation**

   Manual processes often lead to mistakes such as duplicate seat assignments or overlooked constraints. An automated system ensures accurate and conflict-free seat allocation.

2. **Fair Seating Arrangements**

   It minimizes the chances of cheating by ensuring that students are seated according to predefined rules, such as avoiding grouping of students from the same course.

3. **Time and Effort Saving**

   Manual seat management is time-consuming and labor-intensive. Automating the process reduces the workload on staff and saves valuable time.

4. **Efficient Use of Resources**

   The system helps optimize the use of available exam halls and seating capacity, preventing wastage and ensuring maximum efficiency.

5. **Ease of Managing Changes**

   Last-minute changes, such as reassigning seats or accommodating additional students, can be handled quickly and efficiently with an automated system.

6. **Improved Student Experience**

   Clear and organized seating plans reduce confusion for students, ensuring a smoother examination process.

3.2. Preliminary Investigation:

The **Exam Hall Seat Management System** was conceptualized after identifying the challenges faced by educational institutions in managing exam seating arrangements. A detailed investigation was carried out to understand the current process, its limitations, and the requirements for an improved system. Below are the key findings of the preliminary investigation:

1. **Current Process Analysis**

   The existing process of manually assigning seats involves significant administrative effort. Staff members need to prepare seating plans by considering factors such as student roll numbers, available exam halls, and anti-cheating measures. This process is time-consuming, prone to human errors, and becomes increasingly complex as the number of students and halls increases.

2. **Identified Challenges**

   Manual seat allocation often results in errors such as duplicate assignments, inefficient use of exam hall space, and improper seating arrangements. Handling last-minute changes, such as adding students or reallocating seats, is particularly challenging and stressful for staff.

3. **Stakeholder Input**

   Feedback from administrative staff and students highlighted the need for a system that simplifies the process and minimizes errors. Staff emphasized the need for automation to save time and reduce their workload, while students expressed the importance of clear and accurate seating plans to avoid confusion.

4. **Technological Feasibility**

   The investigation found that implementing an automated system using available technologies is feasible. Tools such as database management systems, programming languages, and user-friendly interfaces can be leveraged to develop a robust solution.

5. **Expected Benefits**

   An automated system will ensure accurate seat allocation, efficient resource utilization, and reduced workload for staff. It will also enhance the overall examination experience for students by providing clear and conflict-free seating arrangements.

3.3. Feasibility Study:

The feasibility study for the **Exam Hall Seat Management System** integrates specific software and tools that ensure the project is achievable, efficient, and practical. Below is the evaluation based on the provided information:

❖ **Technical Feasibility**

The project uses a well-structured combination of technologies to ensure functionality and maintainability:

- **Web Framework**:
  - **Flask (Version 2.0.3)**: A lightweight Python web framework used for building web applications, ensuring simplicity and flexibility.
- **Python Libraries**:
  - **Werkzeug (Version 2.0.3)**: A WSGI utility library that provides essential tools for Flask.
  - **MySQL-Connector-Python (Version 8.0.26)**: Facilitates seamless integration of Python applications with MySQL databases for efficient data management.
  - **PDFKit (Version 1.0.0)**: Enables dynamic generation of PDF reports, useful for creating detailed seating plans and reports.
- **Development Environment (IDE)**:
  - **Visual Studio Code**: A powerful and user-friendly code editor, suitable for working with multiple languages like Python, HTML, CSS, and JavaScript.
- **Languages Used**:
  - **HTML and CSS**: For designing the user interface and ensuring responsiveness.
  - **JavaScript**: For adding interactivity to the web application.

These technologies provide a solid foundation for developing the system, making it technically feasible.

❖ **Operational Feasibility**

The use of widely adopted frameworks and libraries ensures ease of use for developers and administrators. Flask and the accompanying libraries offer flexibility, and the interface can be designed to be intuitive, reducing the learning curve for users.

❖ **Conclusion**

The feasibility study confirms that the use of Flask, Python libraries, and supporting tools makes the project technically sound and practical for implementation. The integration of these technologies ensures the system will function effectively while being cost-efficient and user-friendly.

3.4. Project Planning:

The success of the **Exam Hall Seat Management System** depends on a well-structured project plan. Below is a detailed breakdown of the planning process, covering phases, activities, and timelines.

1. Project Objectives

- To develop an automated system for managing exam seating arrangements.
- To ensure error-free seat allocation, resource optimization, and user-friendly interaction.
- To generate clear and detailed seating plans and reports.

2. Project Phases and Timelines

Phase 1: Requirement Analysis (Week 1-2)

- Understand the needs of stakeholders, including administrators and students.
- Define functional and non-functional requirements.
- Prepare a Software Requirement Specification (SRS) document.

Phase 2: System Design (Week 3-4)

- Design the system architecture using Flask as the web framework.
- Create database schemas for storing student and exam hall data.
- Design the user interface using HTML, CSS, and JavaScript.

Phase 3: Development (Week 5-8)

- Implement the backend functionality using Python and Flask.
- Integrate MySQL database for managing data.
- Use PDFKit for generating PDF reports of seating plans.

Phase 4: Testing (Week 9-10)

- Perform unit testing for individual components.
- Conduct system testing to ensure the application meets all requirements.
- Perform user acceptance testing (UAT) with sample data.

Phase 5: Deployment (Week 11)

- Deploy the system on a local or cloud server.
- Provide user training for administrators to familiarize them with the system.

Phase 6: Maintenance and Support (Ongoing)

- Monitor system performance.
- Fix bugs and provide updates based on user feedback.

3.5. Project scheduling :

The following is a detailed project schedule for the development of the **Exam Hall Seat Management System**, divided into specific tasks and their estimated timelines. The project is planned to be completed over a period of 12 weeks, ensuring that each phase is carried out efficiently.

3.5.1. Gantt Chart Overview

Phase	**Task Description**	**Duration**	**Timeline**
Requirement Analysis	Gather requirements from stakeholders	1 week	Week 1
	Prepare Software Requirement Specification (SRS)	1 week	Week 2
System Design	Design system architecture	1 week	Week 3
	Create database schema and UI wireframes	1 week	Week 4
Development	Implement backend functionality with Flask	2 weeks	Week 5-6
	Integrate MySQL database for data management	1 week	Week 7
	Develop front-end interface (HTML, CSS, JS)	1 week	Week 8
	Implement PDF generation with PDFKit	1 week	Week 8
Testing	Conduct unit testing for individual modules	1 week	Week 9
	Perform system and integration testing	1 week	Week 10
	Conduct User Acceptance Testing (UAT)	1 week	Week 10
Deployment	Deploy the system on a server	1 week	Week 11
	Provide training to users	1 week	Week 11
Maintenance	Monitor performance and fix issues	Ongoing	Week 12 onwards

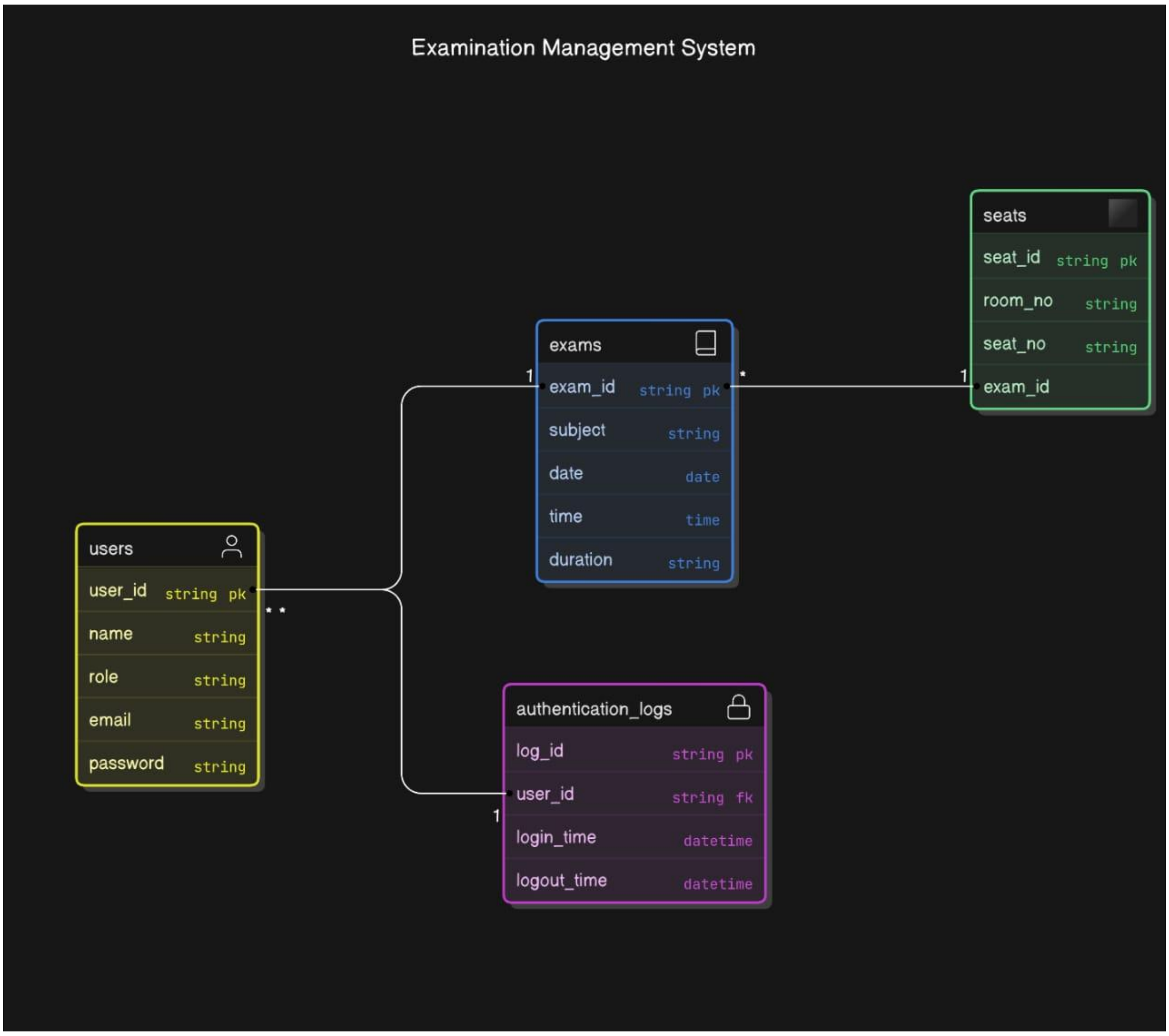
3.5.2. Critical Path Analysis

The critical tasks, such as requirement analysis, backend implementation, and testing, have no slack and must be completed on time to avoid delays. Proper planning and resource allocation ensure timely completion.

3.5.3. Dependencies

- Requirement analysis must be completed before starting the design phase.
- Design must be finalized before development begins.
- Testing depends on the completion of development.
- Deployment requires successful testing and user acceptance.

3.6. ER diagram:



4 . System Design

4.1 Modularization Details:

In this project, modularization involves breaking the entire system into smaller, self-contained modules that handle specific tasks. This approach enhances code readability, maintainability, and reusability. Below is a breakdown of the key modules in the "Exam hall management" project:

4.1.1 For Room management page:

Navigation Bar Module

- **Purpose:**
  - Provides navigation links to different sections of the website.
  - Enables seamless switching between key pages: Room Management, Exam Schedule, Seat Allocation, and Logout.
- **Key Features:**
  - Uses a <nav> tag for structured navigation.
  - Links to various endpoints (/room, /exam, /seat-allocation, /logout).

Room Management Module

- **Purpose:**
  - Enables users to add and manage buildings and rooms for exam hall arrangements.
- **Key Features:**
  - **Add Building:**
    - A form to input and submit building names.
    - Uses POST requests with form validation (e.g., required attribute for input fields).
  - **Add Room:**
    - Form with input fields for room number, floor, column count, and rows for each column.
    - Dropdown for selecting the building name from existing options.
    - Dynamic row input generation based on the column count (generateRowInputs JavaScript function).
  - **Edit Room:**
    - Prepopulates the form for editing existing room details using the populateEditForm JavaScript function.
    - Ensures that users can update room details easily without creating duplicates.

Building and Room Overview Module

- **Purpose:**
  - Displays a comprehensive overview of all buildings and their associated rooms.
- **Key Features:**
  - Each building is shown in a separate container (building-box), making it visually distinct.
  - Each room displays:
    - Room number, floor, column count, row count, and total seats.
  - Action buttons for each room:
    - **View:** Redirects to the room details page using a GET request.
    - **Edit:** Populates the form for updating room details dynamically.
    - **Delete:** Sends a POST request to delete the room with a confirmation prompt.

JavaScript Functions Module

- **Purpose:**
  - Provides interactivity and dynamic behavior for the Room Management page.
- **Key Features:**
  - **generateRowInputs:**
    - Dynamically generates input fields for row counts based on the column count.
    - Ensures users can specify rows for each column in a flexible manner.
  - **populateEditForm:**
    - Pre-fills the form with room details for editing.
    - Adjusts the form dynamically, including generating row inputs and changing the submit button text to "Edit Room."
    - Smooth scrolls to the form for a better user experience.

Responsive Design Module

- **Purpose:**
  - Ensures compatibility across devices with different screen sizes.
- **Key Features:**
  - Utilizes a <meta> tag for setting the viewport.
  - Links external CSS files (room.css and nav.css) for maintaining a consistent layout and style.

Form Validation and Action Module

- **Purpose:**
  - Ensures the correctness and submission of data entered in forms.
- **Key Features:**
  - Each form (Add Building and Add Room) includes required fields (required attribute).
  - The "Add Room" form validates column count before generating row inputs.
  - Confirmation prompt for deleting rooms to prevent accidental deletions.

4.1.2 For Homepage:

Welcome Header Module

- Displays the welcome message: "Welcome to Exam Hall Management."
- Provides a simple and user-friendly introduction to the system.

Navigation Bar Module

- Offers links to key sections:
  - Room Management (/room)
  - Exam Schedule (/exam)
  - Seat Allocation (/seat-allocation)
  - Logout (/logout)
- Ensures consistent navigation across all pages.

Styling and Layout Module

- Uses two CSS files:
  - styles.css: General styling for all pages.
  - home.css: Specific styling for the home page.
- Enhances the visual appearance and maintains a clean layout.

4.1.3 For Exam schedule page:

Exam Creation Module

- **Purpose:**
  - Provides a form for administrators to schedule new exams.
- **Key Features:**
  - Input fields for:
    - College Code
    - Branch
    - Semester
    - Subject
    - Date
    - Time
    - Roll Number Start and Roll Number End
  - Validation enforced using the required attribute to ensure data integrity.
  - Data submitted via a POST request to add new exam details.

Exam Schedule Overview Module

- **Purpose:**
  - Displays a table summarizing all scheduled exams.
- **Key Features:**
  - **Table Columns:**
    - ID
    - College Code
    - Branch
    - Semester
    - Subject
    - Date
    - Time
    - Total Students
    - Action
  - Each row represents an individual exam schedule.
  - Action column includes a delete button for removing specific schedules.

Delete Exam Module

- **Purpose:**
  - Provides functionality to delete an existing exam schedule.
- **Key Features:**
  - Each exam row includes a form with a hidden field (exam_id) to identify the specific schedule.
  - Delete button sends a POST request to delete the exam.
  - Confirmation prompt (confirm) prevents accidental deletions.

Table Styling and Icon Integration Module

- **Purpose:**
  - Enhances the visual appearance of the exam schedule table and adds context through icons.
- **Key Features:**
  - Icons from Font Awesome used for better visual representation (e.g., ID, college code, branch).
  - Table bordered for clear separation of rows and columns.

Responsive Design and Styling Module

- **Purpose:**
  - Ensures the page is user-friendly across different devices and screen sizes.
- **Key Features:**
  - Includes <meta> tag for viewport settings.
  - External CSS file (exam.css) handles the styling of the page.
  - External Font Awesome stylesheet provides scalable icons.

4.1.4 For Seat allocation page:

Allocated Seats Table Module

- **Purpose:**
    - Displays all allocated seat arrangements for a particular branch, date, and time.
- **Key Features:**
    - Table with columns:
        - Select
        - Branch
        - Date
        - Time
        - Action (e.g., edit or delete allocation).
    - Supports interaction for managing existing seat allocations.

Seat Arrangement Pattern Module

- **Purpose:**
    - Provides an option to enable and configure seat arrangement patterns.
- **Key Features:**
    - Checkbox to toggle the "Mix Arrangement Pattern."
    - Dropdown to select the type of arrangement:
        - One by One
        - Side by Side

Seat Statistics Module

- **Purpose:**
    - Displays a summary of seat-related statistics.
- **Key Features:**
    - Displays:
        - Total Seats
        - Allocated Seats
        - Available Seats
    - Dynamically updates values based on allocation status.

Student Statistics Module

- **Purpose:**
    - Provides insights into student allocation.
- **Key Features:**
    - Displays:
        - Total Students
        - Allocated Students
        - Remaining Students
    - Dynamically updates values as students are allocated seats.

Room Visualization Module

- **Purpose:**
    - Offers a graphical representation of the seating arrangement.
- **Key Features:**
    - Displays room layout in the roomContainer section.
    - Button to arrange seats dynamically using the "Arrange Seats" function.
    - Visualization of the final seating arrangement in the visualizationContainer section.

4.2 Data Integrity and Constraints

To ensure the accuracy, consistency, and reliability of data in the **Exam Hall Management System**, the following data integrity and constraints should be applied:

4.2.1. Data Integrity Types

a. Entity Integrity

- Ensures that each record in the database is uniquely identifiable.
- **Implementation:**
  - Primary keys for all tables (e.g., room_id, exam_id, building_id, student_id).
  - Avoids duplicate entries for unique records like exams or rooms.

b. Referential Integrity

- Maintains consistency between related tables through foreign keys.
- **Implementation:**
  - Foreign key relationships:
    - building_id in the **Rooms** table references the **Buildings** table.
    - exam_id in the **Seat Allocation** table references the **Exams** table.
    - student_id in the **Seat Allocation** table references the **Students** table.

c. Domain Integrity

- Ensures that the data entered in each column falls within the predefined range or format.
- **Implementation:**
  - Data type constraints (e.g., integers for room_number, valid dates for exam_date).
  - Use of enums or specific value constraints (e.g., branch values like Computer_Science, Electronics, etc.).

d. User-Defined Integrity

- Enforces business rules specific to the system.
- **Implementation:**
  - Students cannot be assigned to more than one seat for the same exam.
  - A room cannot have more students allocated than its total seating capacity.

4.2.2. Data Constraints

a. Primary Key Constraints

- Ensure uniqueness for key columns.
- Examples:
  - building_id in the **Buildings** table.
  - exam_id in the **Exams** table.
  - room_id in the **Rooms** table.

b. Foreign Key Constraints

- Maintain relationships between tables and prevent invalid references.
- Examples:
  - building_id in the **Rooms** table references **Buildings**.
  - exam_id in the **Seat Allocation** table references **Exams**.

c. NOT NULL Constraints

- Ensure that critical fields are never left empty.
- Examples:
    - room_number in the **Rooms** table.
    - exam_date and exam_time in the **Exams** table.
    - student_id in the **Seat Allocation** table.

d. UNIQUE Constraints

- Ensure that certain columns have unique values.
- Examples:
    - room_number should be unique within the same building.
    - A combination of branch, exam_date, and exam_time should be unique in the **Exams** table.

e. CHECK Constraints

- Enforce conditions on column values.
- Examples:
    - total_seats > 0 in the **Rooms** table.
    - allocated_seats <= total_seats in the **Seat Allocation** table.
    - semester should be within a valid range (e.g., 1–8).

f. Default Constraints

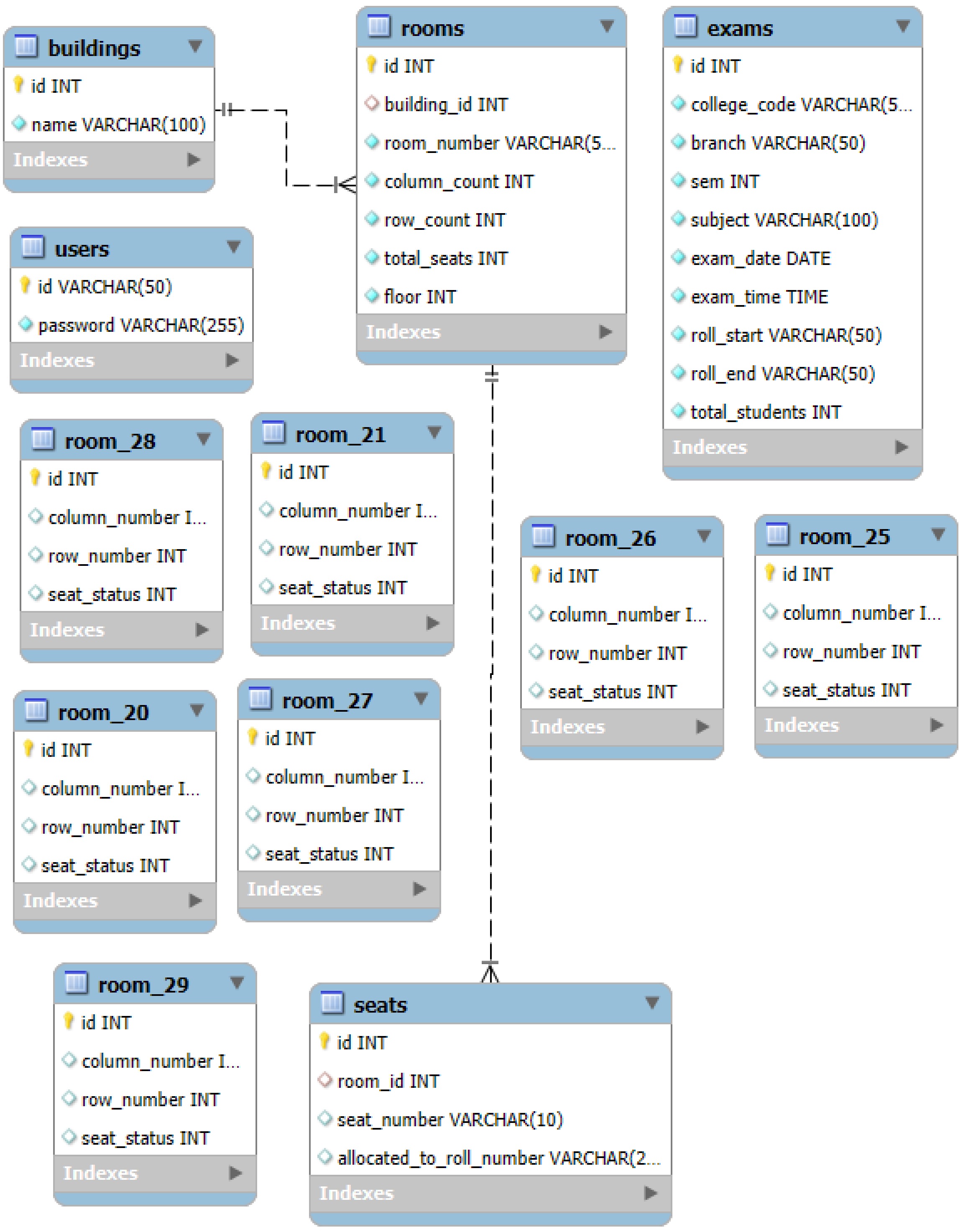
- Provide default values for certain columns.
- Examples:
    - Default value of allocated_seats is 0 in the **Rooms** table.
    - Default status for a student is "Not Allocated" in the **Seat Allocation** table.

4.2.3. Business Rules and Validation

- **Exam Schedule Validation:**
    - The exam_date cannot be in the past.
    - Overlapping exam times for the same branch should not be allowed.
- **Seat Allocation Validation:**
    - The total_students allocated to a room must not exceed its seating capacity.
    - Students must belong to the branch specified in the exam.
- **Room Management Validation:**
    - Each room must belong to a valid building.
    - Columns and rows in a room must have valid positive values.

4.3 Design Approach

The system design for the Exam Hall Management System incorporates efficient and scalable approaches across database, procedural, and object-oriented aspects. This section outlines the methods used to ensure robust functionality, maintainability, and usability.



4.3.1 Database Design
The database is structured to provide efficient data storage and retrieval while ensuring integrity and eliminating redundancy through normalization. Key design elements include:

- Core Tables:
  - *Buildings*: Stores information about buildings (e.g., building_id, building_name).
  - *Rooms*: Contains room details, including room_id, room_number, building_id, floor, and capacity.
  - *Exams*: Manages exam schedules with attributes such as exam_id, branch, exam_date, and exam_time.
  - *Seat Allocations*: Handles seat assignments, including allocation patterns and references to room and student data.
  - *Students*: Captures student-specific details like student_id, name, and roll_number.
- Relationships:
  - *building_id* in the Rooms table references the Buildings table.
  - *exam_id* in the Seat Allocations table references the Exams table.
- Constraints:
  - *NOT NULL* ensures critical fields like room_number and exam_date are always populated.
  - *UNIQUE* enforces uniqueness, e.g., no duplicate room_number within the same building.
  - *CHECK* validates conditions like total_seats > 0 in Rooms.
  - Indexes on frequently queried columns (e.g., exam_date) enhance performance.

4.3.2 Procedural Design
The system is divided into well-defined procedures that handle specific functionalities. Each procedure integrates validation, data processing, and results communication to ensure smooth operation. Key procedures include:

- Room Management:
  - Handles adding, editing, and deleting buildings and rooms.
  - Includes validation for room details, such as ensuring column and row counts are positive integers.
- Exam Schedule Management:
  - Manages the creation, editing, and deletion of exams.
  - Ensures no overlapping schedules for the same branch.
- Seat Allocation:
  - Dynamically assigns seats based on predefined patterns (*One by One* or *Side by Side*).
  - Ensures no room exceeds its capacity.
  - Validates against duplicate or conflicting assignments.
- Visualization and Reporting:
  - Generates room layouts for visual verification of seating arrangements.
  - Provides real-time statistics on seat allocation and student distribution.

4.3.3 Object-Oriented Design
The system models real-world entities as objects to ensure modularity, scalability, and reusability. The following key classes encapsulate attributes and methods:

- Building:
  - Attributes: building_id, building_name.
  - Methods: add_building(), delete_building().
- Room:
  - Attributes: room_id, room_number, building_id, capacity.
  - Methods: add_room(), edit_room(), delete_room().
- Exam:
  - Attributes: exam_id, branch, date, time.
  - Methods: schedule_exam(), edit_exam(), delete_exam().
- SeatAllocation:
  - Attributes: allocation_id, room_id, student_id, pattern.
  - Methods: allocate_seat(), visualize_allocation().
- Student:
  - Attributes: student_id, name, roll_number.
  - Methods: register_student(), update_details().

Key design principles applied:

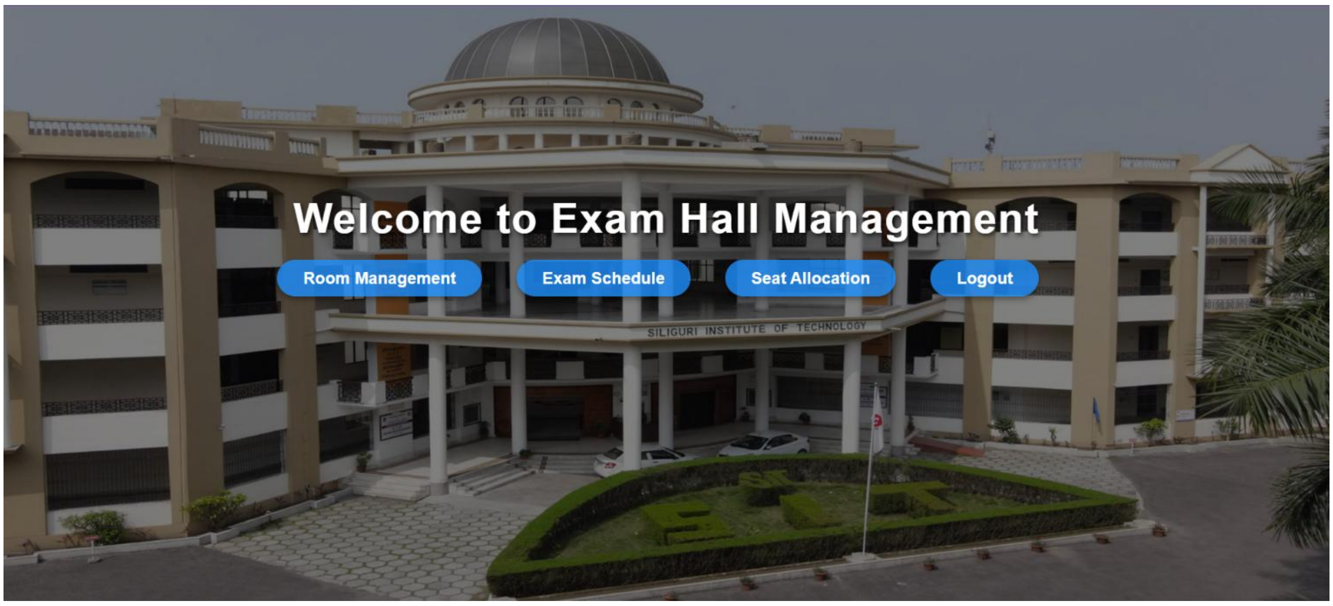
- Encapsulation: Each class encapsulates its data and methods to ensure modularity.
- Abstraction: Focuses on relevant attributes and methods, hiding unnecessary details.
- Polymorphism: Provides flexibility, e.g., allocate_seat() supports multiple seating patterns.
- Relationships:
    - *Association*: A building contains multiple rooms.
    - *Aggregation*: Seat allocation combines room and student data.

By integrating these approaches, the Exam Hall Management System ensures reliability, scalability, and efficient handling of exam-related operations.

4.4 User Interface and design

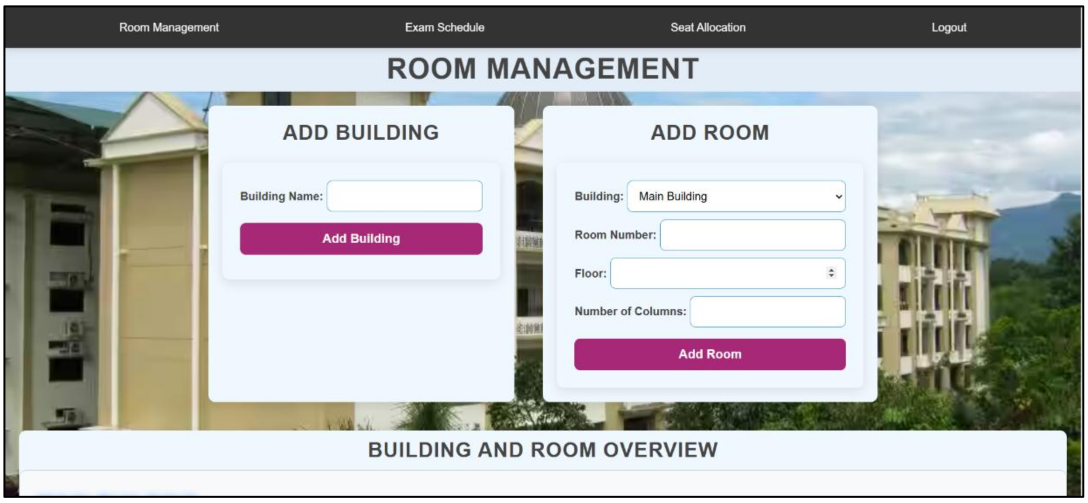
4.4.1. Homepage:

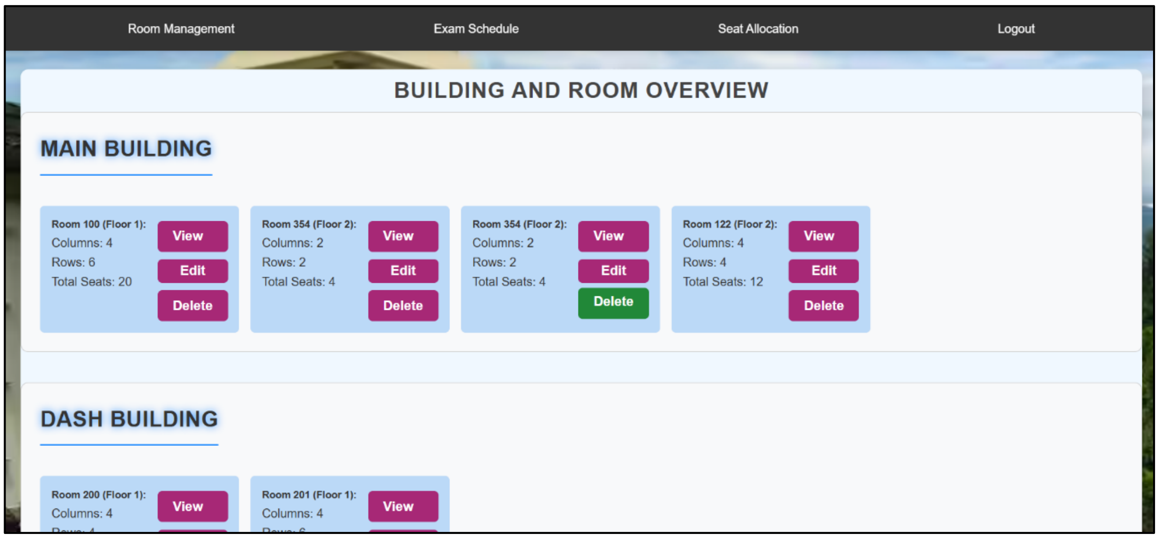
- **Purpose:** Serves as the introduction to the system.
- **Key Elements:**
    - **Header:** A large "Welcome to Exam Hall Management" message to immediately inform users about the system's purpose.
    - **Navigation Bar:** Provides quick links to **Room Management**, **Exam Schedule**, **Seat Allocation**, and **Logout**.
    - **Design:** Clean layout with a clear call-to-action (CTA) for users to start navigating to the core features.


Welcome to Exam Hall Management
Room Management
Exam Schedule
Seat Allocation
Logout
SILIGURI INSTITUTE OF TECHNOLOGY

4.4.2. Room Management Page:

- **Purpose:** Allows administrators to manage room details such as adding buildings, rooms, and displaying room details.
- **Key Elements:**
  - **Room Addition Form:**
    - Input fields for adding room details like Building Name, Room Number, Floor, Column Count, and Row Count.
    - **Add Building** section to input and store building names.
    - Submit button for adding a room.
  - **Room Overview Table:**
    - Displays a list of rooms, showing Room Number, Floor, Total Seats, and a **View**, **Edit**, and **Delete** action for each room.
  - **Design:** Use forms and tables for input and display. Buttons are visually distinct and user-friendly.





Room 100 (Floor 1):
Columns: 4
Rows: 6
Total Seats: 20

Room 354 (Floor 2):
Columns: 2
Rows: 2
Total Seats: 4

Room 354 (Floor 2):
Columns: 2
Rows: 2
Total Seats: 4

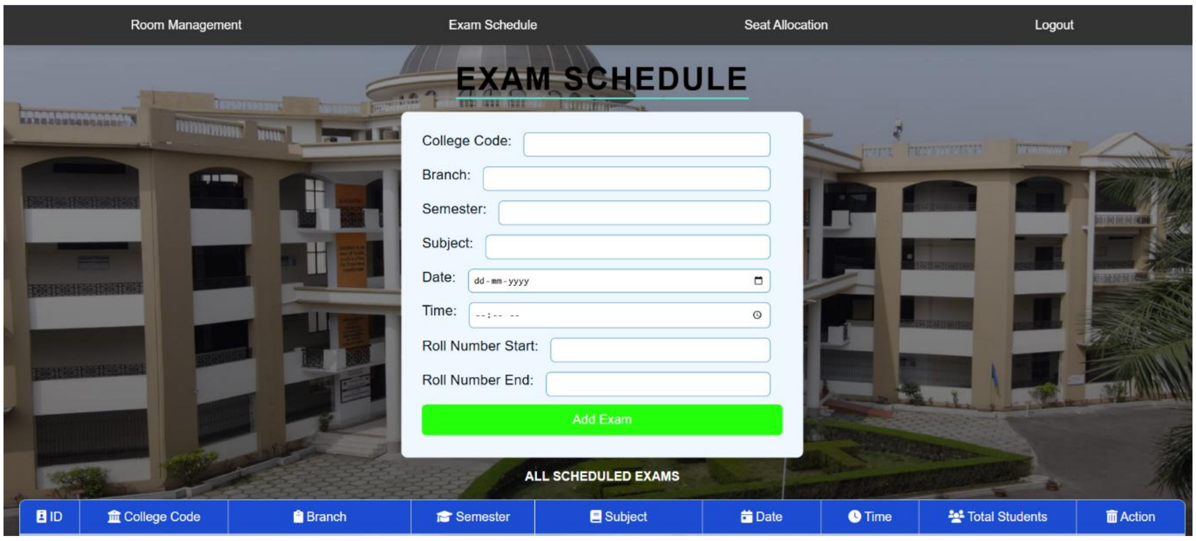
Room 122 (Floor 2):
Columns: 4
Rows: 4
Total Seats: 12

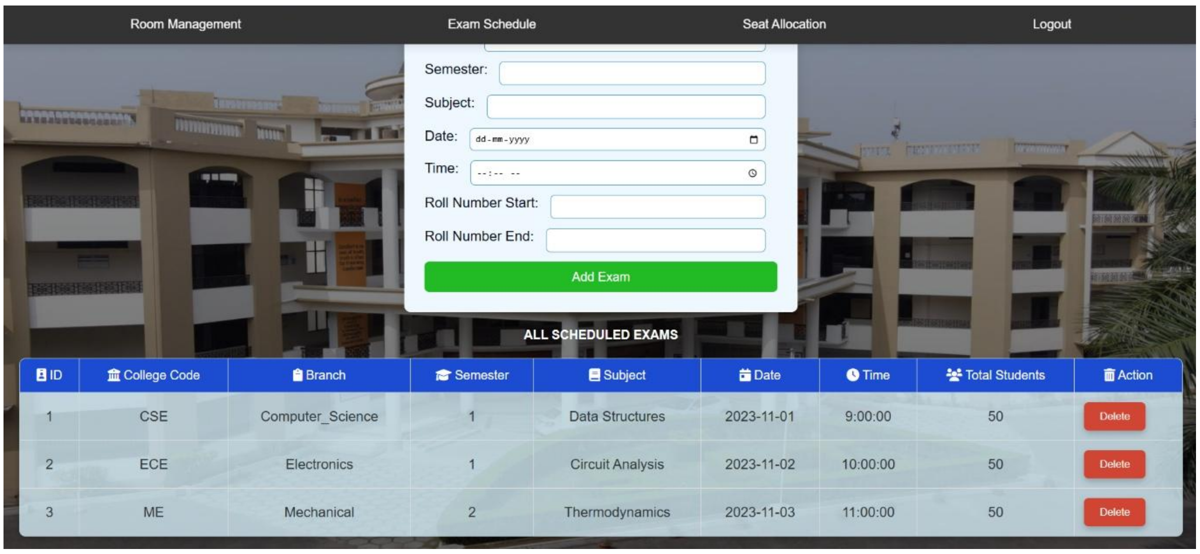
Room 200 (Floor 1):
Columns: 4

Room 201 (Floor 1):
Columns: 4

4.4.3. Exam Schedule Page

- **Purpose:** Displays the scheduling and management of exams.
- **Key Elements:**
  - **Exam Schedule Form:**
    - Input fields for College Code, Branch, Semester, Subject, Date, Time, and Roll Numbers.
    - A **Submit** button to add exams to the schedule.
  - **Scheduled Exams Table:**
    - Displays the list of scheduled exams with columns for ID, College Code, Branch, Semester, Subject, Date, Time, and **Delete** action for each exam.
  - **Design:** Easy-to-use form layout and a responsive table for exam details. Use icons (e.g., trash for delete) to make actions clear.

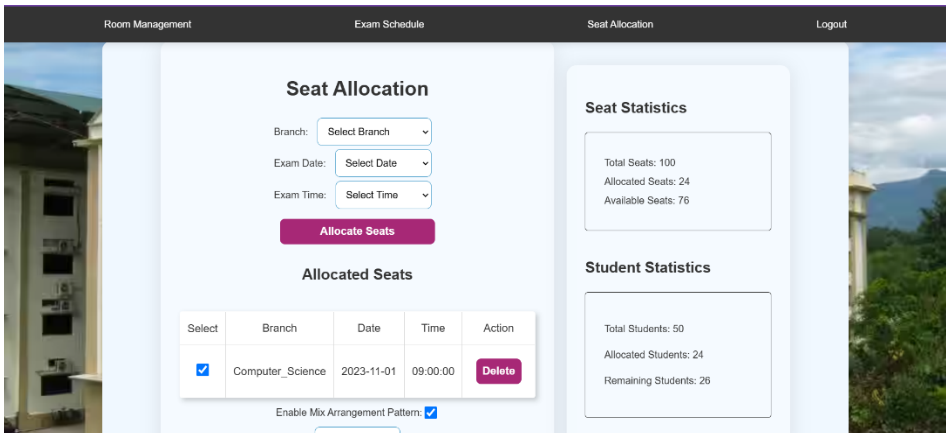




ID	College Code	Branch	Semester	Subject	Date	Time	Total Students	Action
1	CSE	Computer_Science	1	Data Structures	2023-11-01	9:00:00	50	Delete
2	ECE	Electronics	1	Circuit Analysis	2023-11-02	10:00:00	50	Delete
3	ME	Mechanical	2	Thermodynamics	2023-11-03	11:00:00	50	Delete

4.4.4. Seat Allocation Page

- **Purpose:** Manages the allocation of seats based on the exam, room, and student details.
- **Key Elements:**
  - **Seat Allocation Form:**
    - Dropdowns for selecting Branch, Exam Date, and Time.
    - **Allocate Seats** button to trigger seat assignments.
  - **Allocated Seats Table:**
    - Displays the list of seat allocations for students with Branch, Date, Time, and **Action** for managing allocations.
  - **Seat Statistics Section:**
    - Shows total seats, allocated seats, available seats, and remaining students.
  - **Room Visualization Section:**
    - Visual representation of the room seating arrangement (e.g., grid layout for rows and columns).
  - **Seat Arrangement Options:**
    - A toggle for different seating patterns, such as "One by One" or "Side by Side."



Seat Allocation

Branch: Select Branch
Exam Date: Select Date
Exam Time: Select Time

Allocated Seats

Select	Branch	Date	Time	Action
✓	Computer_Science	2023-11-01	09:00:00	Delete

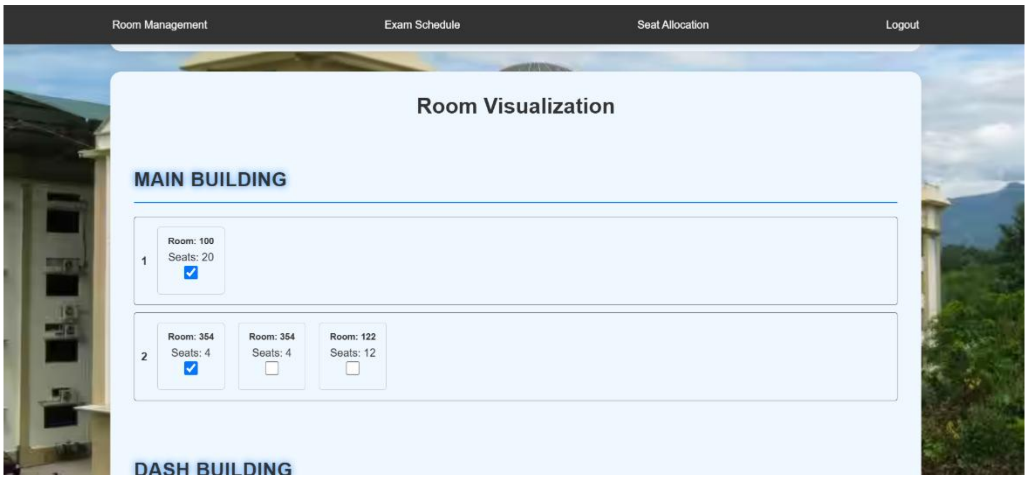
Enable Mix Arrangement Pattern: ✓

Seat Statistics

Total Seats: 100
Allocated Seats: 24
Available Seats: 76

Student Statistics

Total Students: 50
Allocated Students: 24
Remaining Students: 26



Room Visualization

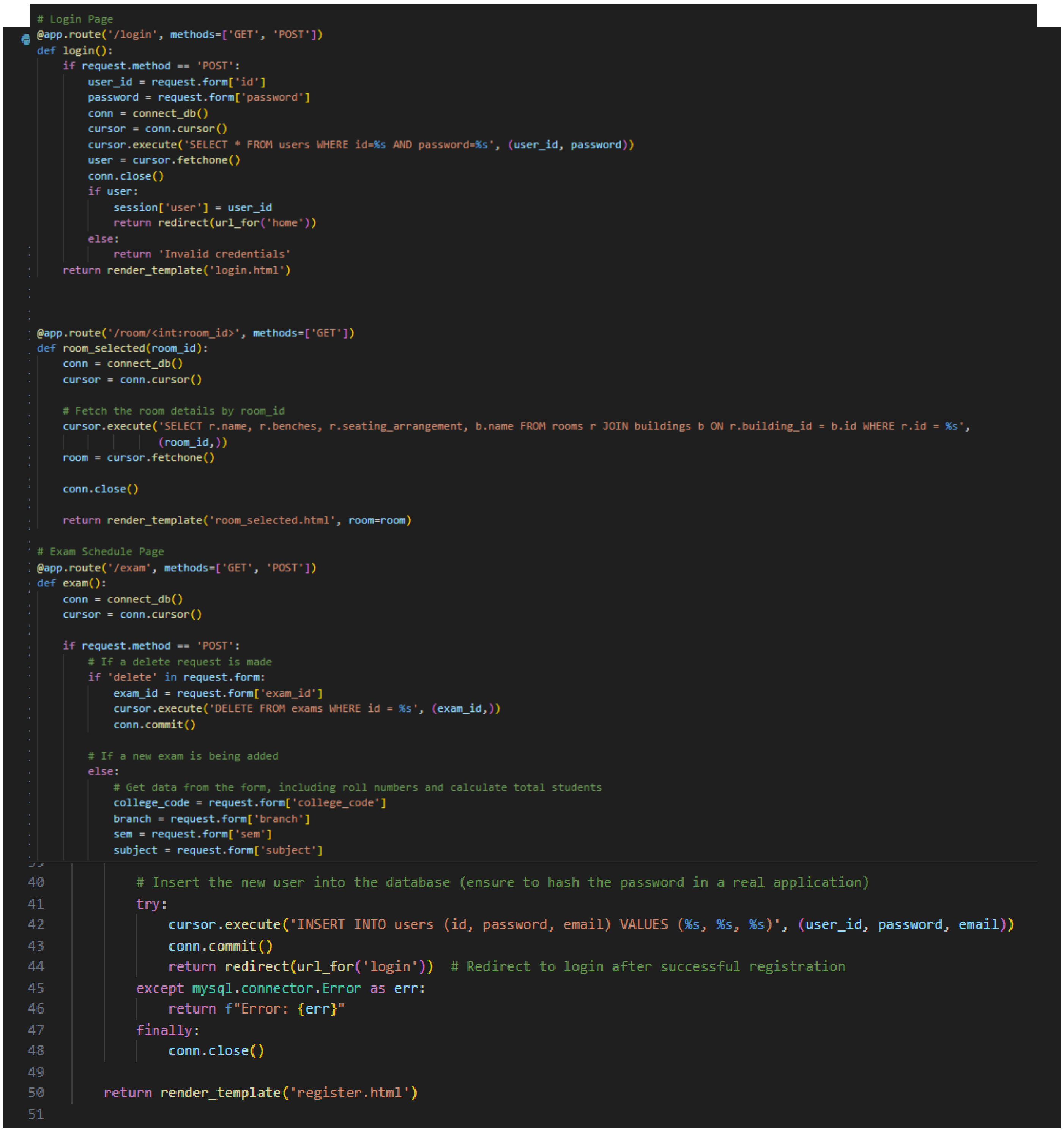
MAIN BUILDING

1	Room: 100 Seats: 20 ✓		
2	Room: 354 Seats: 4 ✓	Room: 354 Seats: 4 ☐	Room: 122 Seats: 12 ☐

DASH BUILDING

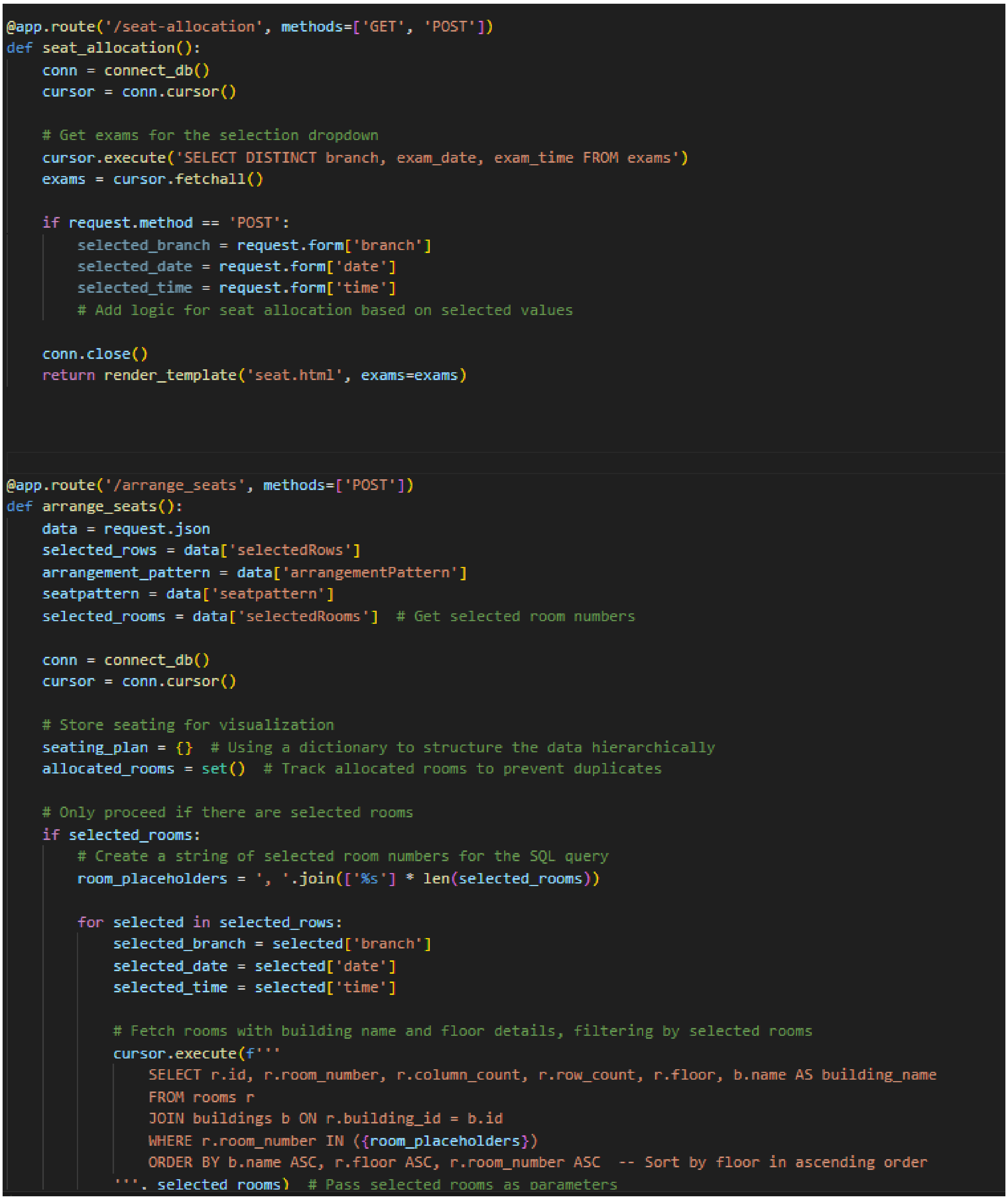
5. Coding

5.1 Sample Code -

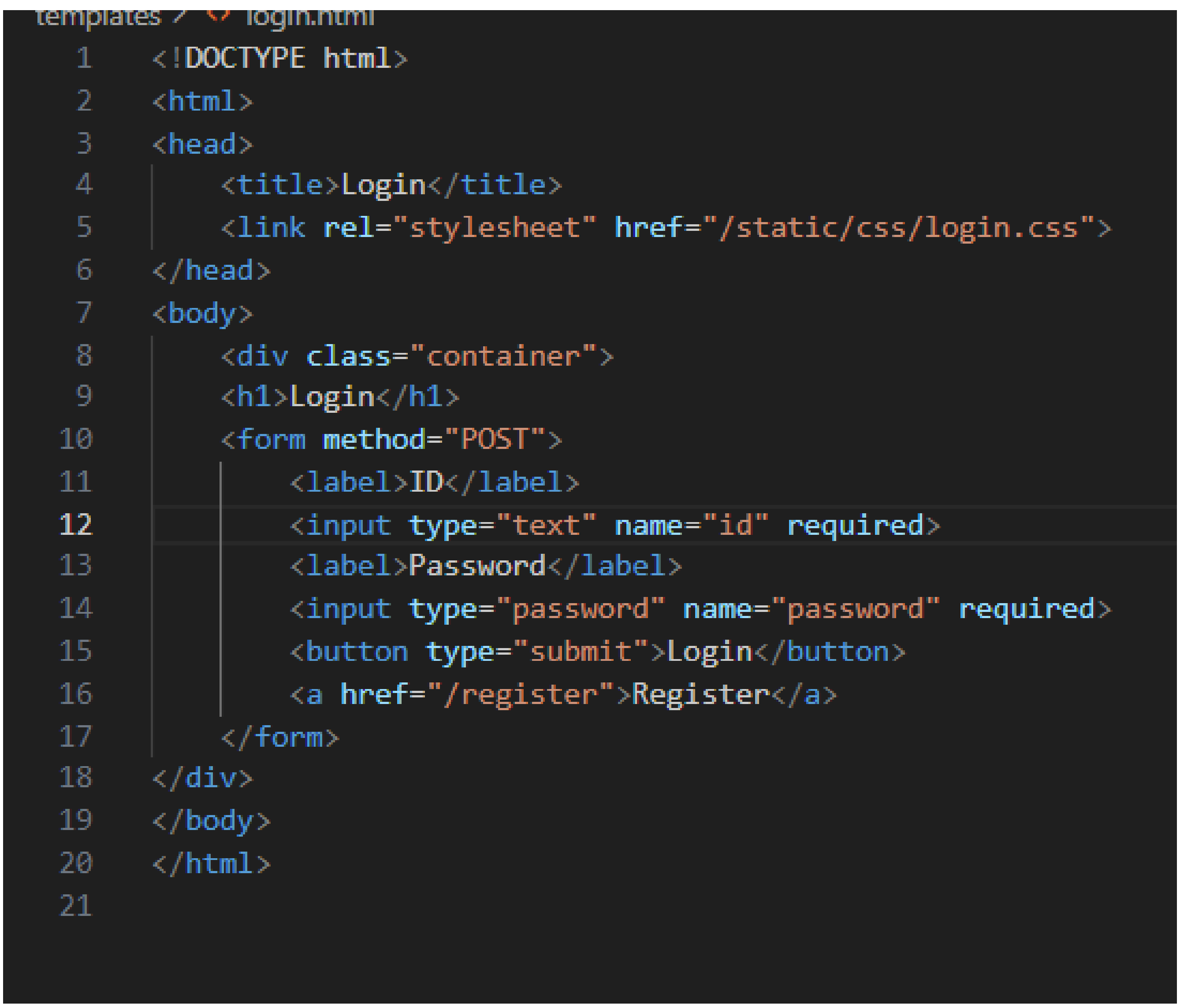


```python
# Login Page
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        user_id = request.form['id']
        password = request.form['password']
        conn = connect_db()
        cursor = conn.cursor()
        cursor.execute('SELECT * FROM users WHERE id=%s AND password=%s', (user_id, password))
        user = cursor.fetchone()
        conn.close()
        if user:
            session['user'] = user_id
            return redirect(url_for('home'))
        else:
            return 'Invalid credentials'
    return render_template('login.html')


@app.route('/room/<int:room_id>', methods=['GET'])
def room_selected(room_id):
    conn = connect_db()
    cursor = conn.cursor()

    # Fetch the room details by room_id
    cursor.execute('SELECT r.name, r.benches, r.seating_arrangement, b.name FROM rooms r JOIN buildings b ON r.building_id = b.id WHERE r.id = %s',
                   (room_id,))
    room = cursor.fetchone()

    conn.close()

    return render_template('room_selected.html', room=room)

# Exam Schedule Page
@app.route('/exam', methods=['GET', 'POST'])
def exam():
    conn = connect_db()
    cursor = conn.cursor()

    if request.method == 'POST':
        # If a delete request is made
        if 'delete' in request.form:
            exam_id = request.form['exam_id']
            cursor.execute('DELETE FROM exams WHERE id = %s', (exam_id,))
            conn.commit()

        # If a new exam is being added
        else:
            # Get data from the form, including roll numbers and calculate total students
            college_code = request.form['college_code']
            branch = request.form['branch']
            sem = request.form['sem']
            subject = request.form['subject']
```

```python
        # Insert the new user into the database (ensure to hash the password in a real application)
        try:
            cursor.execute('INSERT INTO users (id, password, email) VALUES (%s, %s, %s)', (user_id, password, email))
            conn.commit()
            return redirect(url_for('login'))  # Redirect to login after successful registration
        except mysql.connector.Error as err:
            return f"Error: {err}"
        finally:
            conn.close()

    return render_template('register.html')

```

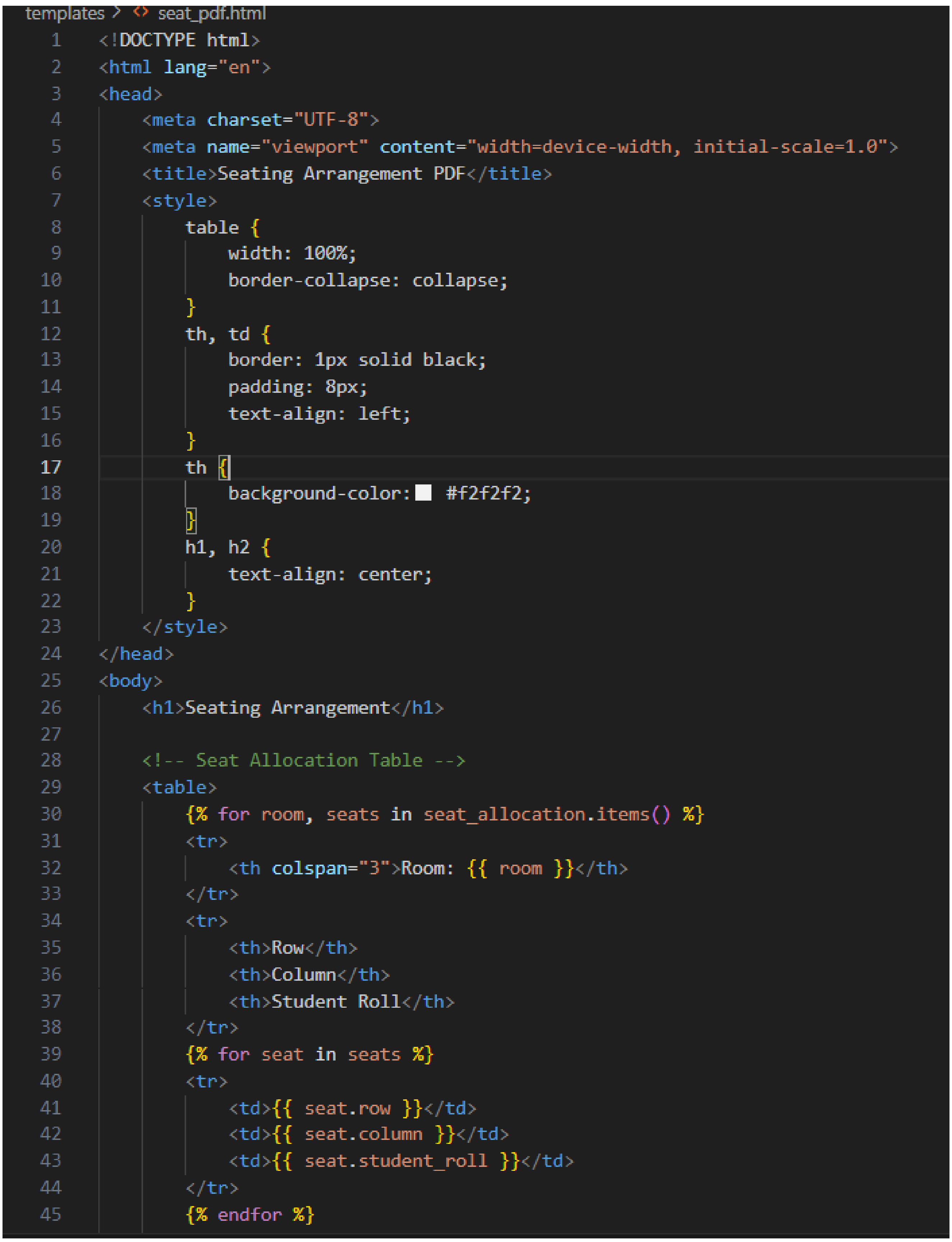


```python
@app.route('/seat-allocation', methods=['GET', 'POST'])
def seat_allocation():
    conn = connect_db()
    cursor = conn.cursor()

    # Get exams for the selection dropdown
    cursor.execute('SELECT DISTINCT branch, exam_date, exam_time FROM exams')
    exams = cursor.fetchall()

    if request.method == 'POST':
        selected_branch = request.form['branch']
        selected_date = request.form['date']
        selected_time = request.form['time']
        # Add logic for seat allocation based on selected values

    conn.close()
    return render_template('seat.html', exams=exams)



@app.route('/arrange_seats', methods=['POST'])
def arrange_seats():
    data = request.json
    selected_rows = data['selectedRows']
    arrangement_pattern = data['arrangementPattern']
    seatpattern = data['seatpattern']
    selected_rooms = data['selectedRooms']  # Get selected room numbers

    conn = connect_db()
    cursor = conn.cursor()

    # Store seating for visualization
    seating_plan = {}  # Using a dictionary to structure the data hierarchically
    allocated_rooms = set()  # Track allocated rooms to prevent duplicates

    # Only proceed if there are selected rooms
    if selected_rooms:
        # Create a string of selected room numbers for the SQL query
        room_placeholders = ', '.join(['%s'] * len(selected_rooms))

        for selected in selected_rows:
            selected_branch = selected['branch']
            selected_date = selected['date']
            selected_time = selected['time']

            # Fetch rooms with building name and floor details, filtering by selected rooms
            cursor.execute(f'''
                SELECT r.id, r.room_number, r.column_count, r.row_count, r.floor, b.name AS building_name
                FROM rooms r
                JOIN buildings b ON r.building_id = b.id
                WHERE r.room_number IN ({room_placeholders})
                ORDER BY b.name ASC, r.floor ASC, r.room_number ASC  -- Sort by floor in ascending order
            ''', selected_rooms)  # Pass selected rooms as parameters
```

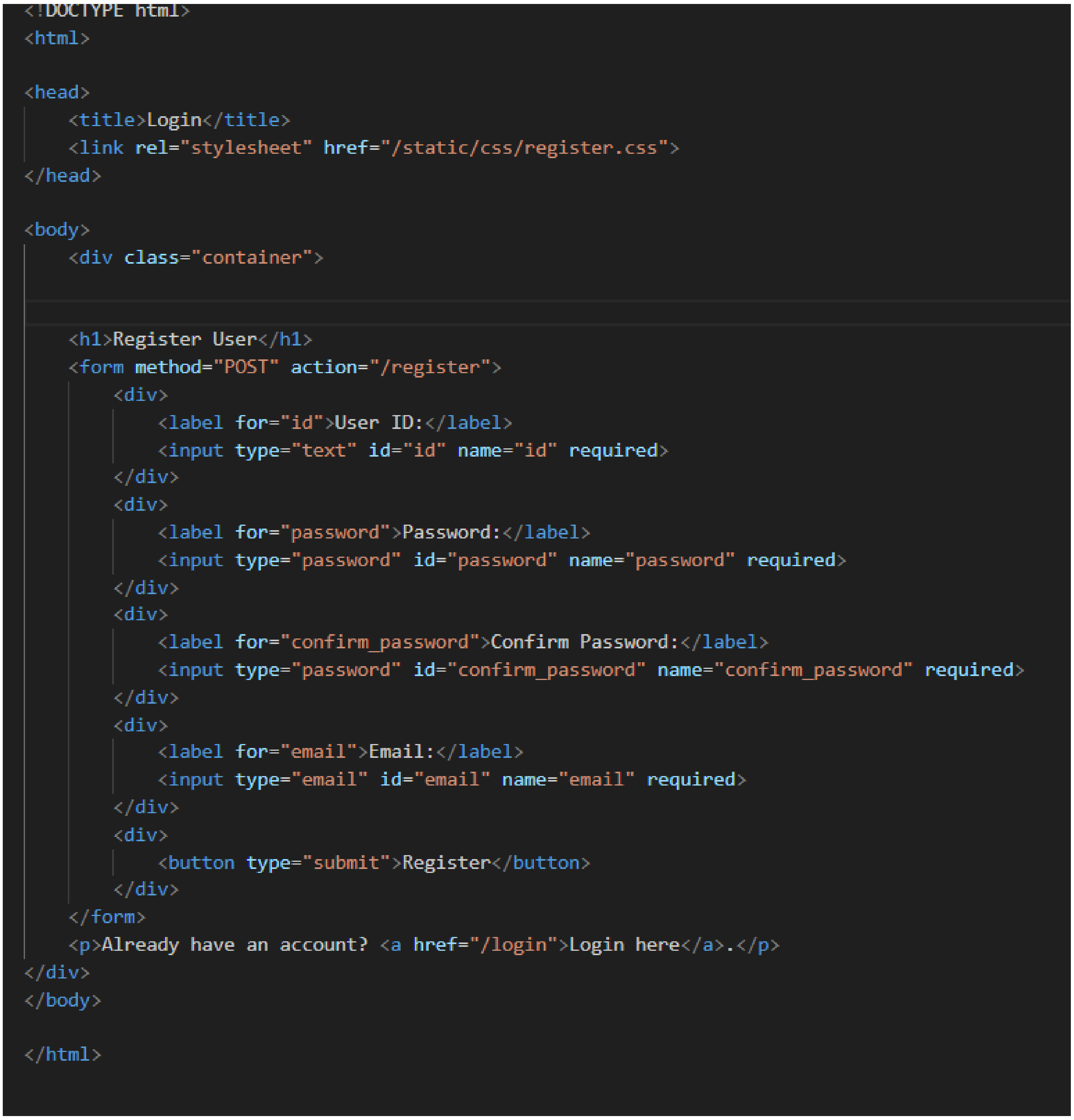


templates > login.html

```html
<!DOCTYPE html>
<html>
<head>
    <title>Login</title>
    <link rel="stylesheet" href="/static/css/login.css">
</head>
<body>
    <div class="container">
    <h1>Login</h1>
    <form method="POST">
        <label>ID</label>
        <input type="text" name="id" required>
        <label>Password</label>
        <input type="password" name="password" required>
        <button type="submit">Login</button>
        <a href="/register">Register</a>
    </form>
</div>
</body>
</html>

```


templates > seat_pdf.html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Seating Arrangement PDF</title>
    <style>
        table {
            width: 100%;
            border-collapse: collapse;
        }
        th, td {
            border: 1px solid black;
            padding: 8px;
            text-align: left;
        }
        th {
            background-color: #f2f2f2;
        }
        h1, h2 {
            text-align: center;
        }
    </style>
</head>
<body>
    <h1>Seating Arrangement</h1>

    <!-- Seat Allocation Table -->
    <table>
        {% for room, seats in seat_allocation.items() %}
        <tr>
            <th colspan="3">Room: {{ room }}</th>
        </tr>
        <tr>
            <th>Row</th>
            <th>Column</th>
            <th>Student Roll</th>
        </tr>
        {% for seat in seats %}
        <tr>
            <td>{{ seat.row }}</td>
            <td>{{ seat.column }}</td>
            <td>{{ seat.student_roll }}</td>
        </tr>
        {% endfor %}
```



```
<!DOCTYPE html>
<html>

<head>
    <title>Login</title>
    <link rel="stylesheet" href="/static/css/register.css">
</head>

<body>
    <div class="container">


    <h1>Register User</h1>
    <form method="POST" action="/register">
        <div>
            <label for="id">User ID:</label>
            <input type="text" id="id" name="id" required>
        </div>
        <div>
            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>
        </div>
        <div>
            <label for="confirm_password">Confirm Password:</label>
            <input type="password" id="confirm_password" name="confirm_password" required>
        </div>
        <div>
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" required>
        </div>
        <div>
            <button type="submit">Register</button>
        </div>
    </form>
    <p>Already have an account? <a href="/login">Login here</a>.</p>
</div>
</body>

</html>
```

5.2 Description –

This project, titled "Exam Hall Management System," is designed to streamline the process of allocating exam halls and seating arrangements for students during examination sessions using advanced algorithms and data management techniques. The system ensures efficient resource utilization, accurate seating assignments, and adherence to examination regulations by considering various constraints and features, such as the number of students, hall capacities, and seating patterns.

The main goal of this project is to optimize the allocation process and minimize manual errors, providing administrators with a robust tool for managing examinations effectively. The key components of the project include:

1. **Allocation Algorithm:**
   The allocation algorithm acts as the core of the system, assigning students to exam halls based on factors like class strength, hall capacity, and subject distribution. The algorithm ensures that no hall is overfilled while maintaining proper spacing between students to prevent malpractices. Constraints such as special seating requests for differently-abled students or candidates requiring extra time are also integrated into the allocation logic.

2. **Conflict Detection:**
   The system incorporates conflict detection mechanisms to handle scheduling clashes. For example, students appearing for multiple exams in different subjects are automatically flagged, and their seating is adjusted to prevent overlap. This ensures that no student is inadvertently scheduled for simultaneous exams.

3. **Dynamic Updates:**
   The project supports real-time updates, enabling administrators to modify seating plans or hall assignments instantly in case of last-minute changes, such as hall unavailability or changes in student attendance. This flexibility ensures smooth operations even under unexpected circumstances.

4. **Visualization Tools:**
   To enhance usability, the system provides visual tools for administrators, such as seating charts and hall layouts. These tools allow for quick verification of seating arrangements and provide a clear overview of resource utilization.

5. **Reporting and Auditing:**
   Comprehensive reporting features enable administrators to generate detailed reports on hall usage, student attendance, and seating patterns. Audit logs ensure transparency by recording all modifications and decisions made during the allocation process, providing accountability and traceability.

6. **Integration with Existing Systems:**
   The Exam Hall Management System is designed to integrate seamlessly with institutional databases, such as student records and timetable management systems. This integration eliminates the need for manual data entry,

5.3 Error Handling –

In this project, error handling ensures that data related to students, exam halls, and schedules is accurate, consistent, and properly formatted for processing. Several strategies are employed to manage and rectify potential issues in the data:

1. **Handling Missing Data:**
   Missing data, such as unassigned student IDs or incomplete hall details, is addressed using appropriate strategies. For example, default values may be assigned to unfilled fields (e.g., "TBD" for hall names) or missing entries are flagged for administrator review.
2.
3. **Data Type Conversion:**
   All input data is validated and converted to the correct format where necessary. For example, roll numbers are ensured to be integers, dates are parsed into a standard datetime format, and seating capacities are verified as positive integers.

4. **Validation of Inputs:**
   The system verifies that all inputs, such as the number of students or hall capacities, fall within reasonable and expected ranges to avoid allocation errors or misconfigurations. For example, a seating capacity cannot be negative or zero.

5. **Conflict Resolution:**
   Potential conflicts, such as overlapping schedules or duplicate seating assignments, are detected and resolved automatically or flagged for manual intervention by the administrator.

5.4 Parameter Passing –

Parameter passing in this project refers to transferring data and configurations across different components, ensuring seamless operation of the system. Key use cases include:

1. **Data Input:**

   Student details (e.g., roll numbers, subjects) and exam hall configurations (e.g., hall capacities) are passed as parameters during the allocation process.

2. **Allocation Logic:**
   Constraints such as seating distance, special requirements, and hall capacity are passed as parameters to the allocation algorithm to ensure compliant seating arrangements.

3. **Dynamic Updates:**
   Parameters like changes in student attendance or hall availability are dynamically passed to update the seating plan in real-time without reinitializing the entire system.

4. **API Integration:**
   In the API layer, parameters such as student information and hall details are received from the user interface and passed to the backend functions for processing and visualization.

5.5 Validation Check –

Validation checks are crucial to ensure that all data, processes, and outputs in the system are consistent and reliable. These checks are implemented at various stages:

1. **Data Validation Checks:**
   Before processing, the system validates input data to ensure correctness:
   - **Completeness Check:** Verify that all mandatory fields (e.g., student IDs, hall capacities) are filled. Missing or incomplete data is flagged.
   - **Data Type Check:** Ensure all data is in the correct type (e.g., roll numbers as integers, hall names as strings).
   - **Value Range Check:** Validate that values like hall capacities and seating distances fall within logical ranges.
2. **Allocation Validation Checks:**

   After running the seating allocation algorithm, the results are validated to ensure compliance with constraints:
   - **No Overlap Check:** Ensure that no student is assigned to multiple seats or halls simultaneously.
   - **Hall Capacity Check:** Validate that the number of students assigned to each hall does not exceed its capacity.
   - **Special Requirements Check:** Confirm that students with special needs are assigned to the appropriate seating or halls as per their requirements.
3. **API Validation Checks:**

   Validation in the API ensures smooth interaction between the user interface and backend:
   - **Input Validation:** Ensure API requests contain all required fields, such as student data and hall details, in the correct format.
   - **Output Validation:** Validate API responses to ensure they include all necessary information (e.g., seating plans or conflict reports) in the expected format.
4. **Deployment Validation Checks:**

   Validation during deployment ensures the system is fully operational:
   - **Database Connection Validation:** Verify that the database is correctly configured and accessible for storing and retrieving data.
   - **Algorithm Integration Check:** Ensure that the allocation algorithm is correctly deployed and functions as intended.
   - **UI Functionality Check:** Validate that the user interface accurately reflects seating arrangements and allows for updates or corrections as needed.

7. reduces redundancy, and ensures consistency across platforms.

6. Testing and System Security Measures

6.1 Testing

Testing Techniques and Strategies: Testing was conducted to ensure the robustness, reliability, and accuracy of the Exam Hall Management System. Multiple strategies were employed to identify bugs and verify compliance with the system's requirements:

1. **Unit Testing**: Focused on individual modules such as Room Management, Exam Scheduling, and Seat Allocation to validate their functionality independently.
2. **Integration Testing**: Verified seamless interaction between interconnected modules, ensuring data flow integrity.
3. **System Testing**: Assessed the overall system performance to ensure it meets all specified requirements.
4. **User Acceptance Testing (UAT)**: Conducted with end-users to evaluate real-world usability and effectiveness.

Test Case Design and Test Reports: Test cases were designed for various scenarios, including edge cases. Sample test cases included:

- Validating seat allocation for maximum capacity.
- Handling invalid inputs in room creation.
- Real-time updates in seat arrangements under changing conditions.

A detailed test report was generated after each testing phase, summarizing:

- Test Case ID
- Test Description
- Expected Outcome
- Actual Outcome
- Status (Pass/Fail)

6.2 Debugging and Code Improvement

Debugging tools and techniques were employed to identify and resolve issues. This included:

- Using **debug logs** for tracking function execution.
- Employing **breakpoints** in the IDE for step-by-step code inspection.
- Refactoring code to improve readability, maintainability, and performance, especially in allocation algorithms.

6.3 System Security Measures

i. Database/Data Security:

- **Encryption**: Sensitive data such as user credentials and student details were encrypted using industry-standard methods.
- **Access Control**: Only authorized personnel could access the database through secured credentials.
- **Backups**: Regular backups of the database were scheduled to prevent data loss.

ii. User Profiles and Access Rights:

- **Role-Based Access Control (RBAC)**: Implemented roles like "Admin" and "User" with specific permissions.
- **Authentication and Authorization**: Password-based authentication combined with token-based session management.
- **Activity Logging**: Maintained logs for all critical operations to ensure traceability.

6.4 Cost Estimation of the Project

The project cost was estimated considering:

- **Development Tools**: Free and open-source tools like Flask, MySQL, and Visual Studio Code were utilized.
- **Hosting Services**: Cloud-based or local server hosting costs were minimal.
- **Manpower**: Estimated hours of work from developers and testers.

6.5 Reports

Sample reports were generated, including:

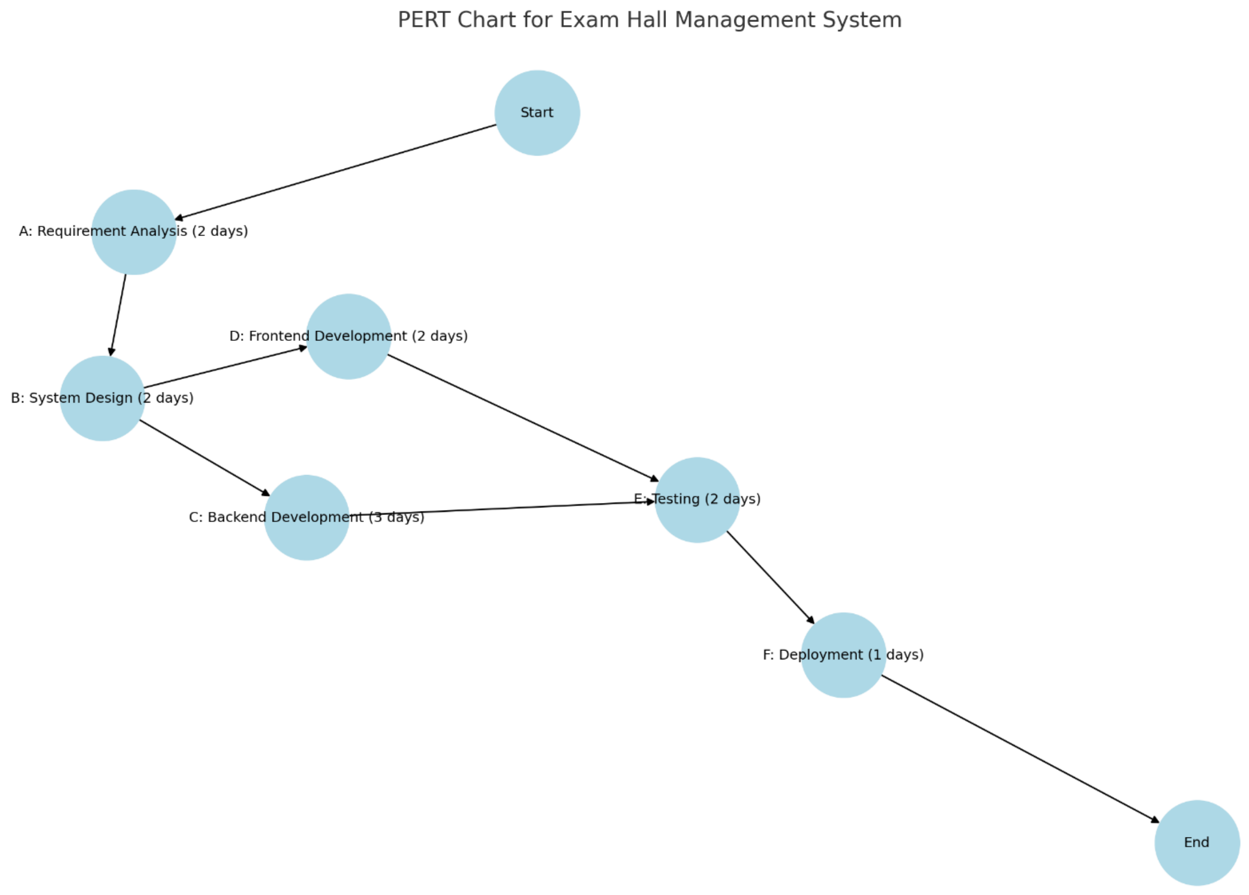
- Room Utilization Report
- Seat Allocation Summary
- Exam Scheduling Logs

These reports were designed in PDF format using dynamic templates for clarity and professional appearance.

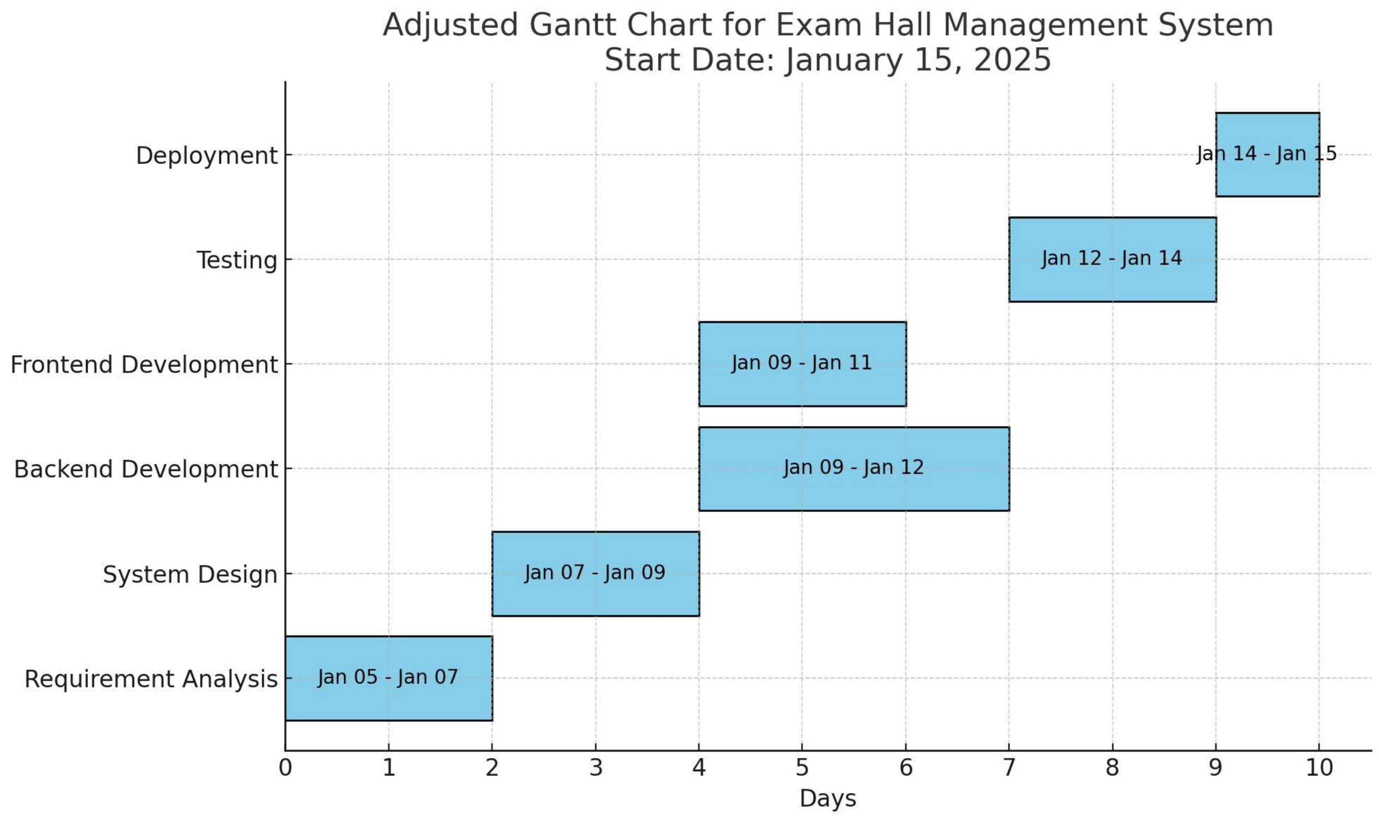
6.6 Project Scheduling Tools

PERT Chart and **Gantt Chart** were used to plan and track progress. Key milestones such as requirement gathering, system design, development, testing, and deployment were included, ensuring adherence to the project timeline.

PERT Chart


PERT Chart for Exam Hall Management System
Start
A: Requirement Analysis (2 days)
D: Frontend Development (2 days)
B: System Design (2 days)
C: Backend Development (3 days)
E: Testing (2 days)
F: Deployment (1 days)
End

Gantt Chart


Adjusted Gantt Chart for Exam Hall Management System
Start Date: January 15, 2025
Deployment
Testing
Frontend Development
Backend Development
System Design
Requirement Analysis
Jan 14 - Jan 15
Jan 12 - Jan 14
Jan 09 - Jan 11
Jan 09 - Jan 12
Jan 07 - Jan 09
Jan 05 - Jan 07
0
1
2
3
4
5
6
7
8
9
10
Days

7. Conclusion and recommendations

7.1 Conclusion

The **Exam Hall Management System** is designed to streamline the management of exam halls, providing administrators and students with an efficient platform to handle seat allocation, room management, and exam scheduling. The system ensures smooth coordination between building layouts, room availability, and student seat assignments. By leveraging structured data management through a relational database, dynamic seat allocation, and intuitive user interfaces, this system enhances the overall experience for both administrators and students.
The modular design, built with well-defined components such as room management, seat allocation, and exam scheduling, ensures that the system is maintainable, scalable, and can easily accommodate future improvements or additional features. The use of procedural and object-oriented design principles further contributes to its flexibility, ensuring the system remains adaptable as requirements evolve.

7.2. Recommendations

Enhance User Experience (UX):

- Optimize the system for mobile devices and implement real-time updates for seat allocation and exam schedules.

Data Analytics and Reporting:

- Add advanced reporting features for seat occupancy and room utilization, and use interactive visualizations for room statistics.
-

Integration with Other Systems:

- Integrate with a Student Management System (SMS) and allow syncing of exam dates with calendar platforms for better scheduling.

Security Enhancements:

- Implement Two-Factor Authentication (2FA) for secure logins and ensure data encryption for sensitive information.

8. References

- McCabe, W.L., and Smith, J.C., **Unit Operations in Chemical Engineering**, 4th ed., Tata McGraw-Hill, pp. 812–814.

- Kerr, G.T., **Chemistry of Crystalline Aluminosilicate**, The Journal of Physical Chemistry, Vol. 73, No. 3, pp. 1385–1386, April 1968.

- Garside, J., et al., **Industrial Crystallization from Solution**, Chemical Engineering Science, Vol. 40, No. 2, pp. 3–26, 1985.

- Grimes, D., **Database Design and Relational Database Theory**, Addison-Wesley, pp. 120–145, 2012.

- Flask Documentation, **Flask Web Framework Overview**, Flask, available at: https://flask.palletsprojects.com/, accessed on January 22, 2025.

- Cooper, A., Reimann, R., and Cronin, D., **About Face: The Essentials of Interaction Design**, Wiley, pp. 340–378, 4th Edition, 2014.

- Nixon, R., **Learning PHP, MySQL & JavaScript: With jQuery, CSS & HTML5**, O'Reilly Media, 5th ed., pp. 92–103, 2020.

- Pandey, P.K., and Sharma, D., **Algorithm Design for Seat Allocation**, International Journal of Software Engineering, Vol. 8, No. 5, pp. 65–78, 2019.