

**COLLEGE MANAGEMENT SYSTEM  
USING DJANGO & MYSQL**

## **Abstract**

This document outlines a college management system web application designed to streamline administrative processes within an educational institution. Developed using Python's Django framework, the application offers a comprehensive suite of features to manage student and staff data, attendance records, leave requests, and communication channels. Mimicking real-world college administration scenarios, the system provides dedicated user panels for students, staff, and administrators, each with tailored functionalities.

Students can leverage the system to view and update their personal profiles, access attendance records for enrolled subjects, submit leave applications, and provide feedback to the administration. Staff members can manage attendance for their assigned courses, apply for personal leave, and communicate with the college administration. Administrators hold ultimate authority, managing courses, subjects, academic sessions, staff and student records, and generating reports on student attendance, potentially visualized using graphs and charts.

Beyond its core functionality, the project serves as a valuable educational resource for students, particularly those pursuing IT fields. By delving into the source code (assuming it's available), students can gain practical experience with Django web development frameworks and explore the implementation of real-world functionalities within a college management system. The user interface itself is built with Bootstrap, a popular open-source CSS framework, offering a clean and user-friendly experience.

In summary, this college management system application demonstrates the capabilities of Python's Django framework in building practical web applications. It provides a comprehensive solution for college administration while offering educational value for students seeking to understand and implement such systems.

**INDEX**

CONTENTS	PAGE NO
1 INTRODUCTION	10
1.1 INTRODUCTION .....	11
1.2 EXISTING SYSTEM.....	12
1.3 PROPOSED SYSTEM.....	13
1.4 GOALS.....	14
1.5 DESCRIPTION OF PROJECT-----	14
1.6 SOFTWARE AND HARDWARE SPECIFICATIONS -----	16
1.7 SOFTWARE METHODOLOGY -----	20
2 STUDY PHASE	22
Flow Chart-----	23
System architecture diagram -----	24
Use-Case Diagram-----	26
SEQUENCE DIAGRAM -----	27
Class diagram -----	28
Component Diagram-----	29
Deployment Diagram-----	30
ER diagram -----	31
STATE DIAGRAM-----	32
Activity Diagram -----	33
2.1Introduction To Analysis	34
2.3 FEASIBILITY STUDY-----	35
2.3 DATA COLLECTION-----	36
2.4 DATA ANALYSIS-----	38
2.5 Literature Review -----	40
3 DESIGN PHASE	41

3.1 DESIGN PROCESS-----	42
3.2 DESIGN AXIOMS-----	50
3.3 REFININIG ATTRIBUTES -----	52
3.4 REFINING METHODS-----	52
3.5 REFINING CLASS DIAGRAMS-----	53
DESIGN OF VIEW LAYER -----	55
4.IMPLEMENTATION AND SYSTEM TESTING	61
4.1 FORNTEND FEATURES-----	62
4.2 BACKEND FEATURES-----	63
4.3 TEST CASES-----	64
5.SCREENSHOTS-----	68
6.CONCLUSION-----	73
7.FUTURE ENHANCEMENT-----	75
8.Bibliography-----	76

## **INTRODUCTION**

## 1.1 INTRODUCTION

The burgeoning complexity of managing educational institutions necessitates innovative solutions to streamline administrative processes. This project tackles this challenge by presenting a college management system web application designed to automate and centralize various administrative tasks within a college setting. Developed using Python, a versatile and widely used programming language, and the Django web framework, known for its rapid development capabilities, this application offers a comprehensive suite of functionalities to effectively manage student and staff information, attendance records, leave requests, and feedback.

The system caters to three distinct user roles, each with designated functionalities accessible through a user-friendly interface built with the Bootstrap CSS framework. Students can leverage the system to view and update their profiles, access and track their attendance records, submit leave requests for approval, receive important college announcements, and provide feedback to the administration. Staff members, such as professors and instructors, can manage student attendance for the subjects they teach, apply for leave, receive notifications, and provide feedback to improve the learning experience. Administrators, occupying the highest level of access, hold the reins of the system. They can manage courses, subjects, and academic sessions, oversee staff and student records, maintain attendance data, send college-wide notifications, review and approve/reject leave requests and staff feedback, and potentially generate insightful reports on student attendance data, visualized with graphs and charts for better comprehension.

This project extends beyond its core functionalities, serving as a valuable learning resource for students, particularly those pursuing studies in Information Technology (IT). By providing a practical example of web application development using Python and Django, this project can significantly enhance students' understanding of web development principles. They can delve into the codebase, analyze the implementation of various functionalities, and gain valuable insights into real-world web application development practices. This project serves as a springboard for students to further explore web development concepts and potentially contribute to the ongoing development and refinement of the system.

## 1.2 Existing Method:

Existing System Functionalities:

Many college management systems share core functionalities, though specific features might differ. Here are some common examples:

**Student Management:** Enrollment, profile management, fee management, academic record tracking.

**Staff Management:** Faculty and staff information, course assignment, attendance management.

**Attendance Management:** Recording student attendance, generating reports.

**Leave Management:** Requesting, approving/rejecting leave for students and staff.

**Course Management:** Creating and managing courses, assigning faculty, scheduling classes.

**Result Management:** Recording and tracking student grades, generating reports.

**Communication Tools:** Internal messaging system for announcements and communication.

**Reporting & Analytics:** Generating reports on various aspects like attendance, grades, student demographics.

Finding Existing Systems:

Several companies offer commercial college management systems with varying functionalities and pricing structures. Open-source options might also be available, but they might require more technical expertise for setup and maintenance.

Here's how this project fits in:

The described project using Python and Django seems to cover many core functionalities of a college management system. It provides a user-friendly interface and caters to different user roles. While it might not be a full-fledged commercial system, it serves as a valuable learning resource and potentially a foundation for further development and customization to suit a specific college's needs.

### 1.3 PROPOSED SYSTEM

The information provided doesn't explicitly mention a proposed system beyond the described college management system itself. However, based on the functionalities mentioned, here's a breakdown of the potential system design:

#### Client-Side:

Developed using HTML, CSS (likely with Bootstrap for UI), and potentially Java-script for interactive elements.

Separate user interfaces for Students, Staff, and Admin with functionalities tailored to each role.

#### Server-Side:

Developed using Python and the Django web framework.

Handles user authentication, authorization, data processing, and interaction with the database.

#### Database:

Stores all college data, including student and staff information, course details, attendance records, leave requests, feedback, and potentially user login credentials (securely hashed).

The specific database management system (e.g., MySQL, PostgreSQL) is not mentioned but would be chosen based on project requirements.

Here are some additional details that could be included in a proposed system document:

**Detailed Database Schema:** A diagram outlining the database tables, their attributes (columns), and relationships between tables.

**User Interface (UI) Mockups or Wireframes:** Visual representations of the user interfaces for each user role, showcasing the layout and placement of functionalities.

**System Architecture Diagram:** A high-level overview depicting the interaction between the client-side, server-side, and database components.

**Security Measures:** Outline of the security protocols implemented to protect user data, including authentication, authorization, and data encryption practices.



## **1.4 Goals:**

### **Functional Goals:**

**Streamline College Administration:** Automate and centralize administrative tasks like student recordkeeping, attendance tracking, leave management, and communication. This can improve efficiency and reduce manual workload for staff.

**Enhance Transparency and Accessibility:** Provide students and staff with a user-friendly platform to access and manage their information, track attendance, and submit requests. This can improve transparency and accessibility of college processes.

**Improved Communication:** Facilitate communication between students, staff, and administrators through a notification system and potential messaging functionalities.

**Data-Driven Decision Making:** Generate reports on student attendance and potentially other areas. This data can be used for informed decision making by college management.

### **Educational Goals:**

**Learning Resource for IT Students:** Serve as a practical example of web application development using Python and Django. This can help students understand web development principles and gain hands-on experience.

**Promote Understanding of College Management Systems:** Provide students with insights into the functionalities and potential benefits of college management systems.

## **1.5 Description of project**

This project delves into the development of a web application designed to streamline administrative processes within a college environment. Leveraging the power and versatility of Python and the Django web framework, the system offers a robust suite of functionalities to manage student and staff information, attendance tracking, leave requests, feedback mechanisms, and reporting capabilities.

The system caters to three distinct user groups, each with a designated interface and functionalities aligned with their roles:

**Students:** Students can leverage the system to manage their profiles, conveniently view and track their attendance records, submit leave requests for approval by the administration, stay informed through college-wide announcements, and voice their feedback to improve the learning environment.

**Staff:** Faculty and staff members can utilize the system to manage student attendance for the courses they teach, streamlining this often time-consuming task. Additionally, they can submit leave requests, receive important notifications, and provide feedback to enhance college operations.

**Administrators:** As the system's central authority, administrators hold the reins of various functionalities. They can manage courses, subjects, and academic sessions, ensuring the

smooth running of the academic calendar. Additionally, they oversee staff and student records, maintaining accurate data for informed decision-making. Attendance data management is another key responsibility, allowing administrators to track student participation and identify areas for improvement. Furthermore, the system empowers administrators to send college-wide notifications for important announcements or updates. Reviewing and approving/rejecting leave requests and staff feedback submitted through the system ensures a centralized and organized approach to these processes. Finally, the system offers the potential to generate insightful reports on student attendance data, potentially visualized with graphs and charts for better comprehension. These reports can equip administrators with valuable insights to optimize college operations and improve student success rates.

### **Technical Underpinnings:**

The system's functionalities are built upon a solid technical foundation:

**Front-End:** The user interface, likely developed with HTML, CSS (potentially using Bootstrap for a responsive design), and potentially Javascript for interactive elements, provides a user-friendly experience for students, staff, and administrators.

**Back-End:** The core functionalities reside in the back-end, developed with Python and the Django web framework. This powerful combination handles user authentication and authorization, processes user interactions, manages data effectively, and facilitates seamless communication with the database.

**Database:** The system relies on a robust database (specific system not mentioned, but options like MySQL or PostgreSQL are common choices) to store all college data securely. This data encompasses student and staff information, course details, attendance records, leave requests, feedback, and potentially user login credentials (securely hashed for enhanced security).

### **Beyond Functionalities: A Learning Resource**

While the core objective of the project is to provide a comprehensive college management system, it extends its value by serving as a valuable learning resource for students, particularly those pursuing studies in Information Technology (IT). By providing a practical example of web application development using Python and Django, the project offers students a unique opportunity to delve into the codebase, analyze the implementation of various functionalities, and gain practical insights into real-world web development practices. This hands-on exploration can significantly enhance their understanding of web development principles and equip them with valuable skills for their future careers. Furthermore, the project can serve as a springboard for students to further explore web development concepts and potentially contribute to the ongoing development and refinement of the system, fostering a collaborative learning environment.

## 1.6 SOFTWARE AND HARDWARE SPECIFICATION

### ➤ Hardware Requirements:

- o Modern multi-core processor
- o Sufficient RAM (8 GB or more recommended)
- o Reliable internet connection

### ➤ Software Requirements:

- o Operating System: Windows 10, macOS, or Linux (Ubuntu, CentOS, etc.)
- o MySQL
- o Python(latest version)
- o Integrated Development Environment (IDE) of your choice (e.g., Visual Studio Code, PyCharm)
- o Web browser for accessing user interfaces and documentation

### Functional and non-functional requirements:

#### Functional requirements:

##### User Management:

Create, edit, and deactivate user accounts for students, staff, and administrators.

Implement secure user authentication and authorization mechanisms (login, password management).

##### Student Management:

Allow students to view and update their personal information (name, contact details, etc.).

Facilitate student profile management (including image upload).

Enable students to view their attendance records and class schedules.

Allow students to submit leave requests electronically with justifications and track their status.

Provide a mechanism for students to provide feedback to the administration.

##### Staff Management:

Allow staff members (faculty) to view and update their personal information.

Enable staff to manage attendance for courses they teach (mark attendance, view reports).

Facilitate staff leave request submission and tracking of approval status.

Provide a platform for staff to provide feedback to the administration.

### **Admin Management:**

Offer comprehensive management of courses, subjects, and academic sessions.

Allow adding, editing, and deleting courses, subjects, and sessions.

Facilitate management of student and staff records (add, edit, delete).

Enable viewing and managing attendance data for all students across courses and semesters (potentially with reports and visualizations).

Provide a system for handling student leave requests (approval/rejection).

Allow managing staff leave requests (approval/rejection).

Facilitate sending college-wide notifications to students and staff.

Offer functionalities for managing and reviewing feedback received from students and staff.

Potentially generate reports on various aspects like student attendance, enrollment statistics, etc. (data visualization could be a plus).

### **Non-Functional Requirements**

**Performance:** The system should be responsive and handle user requests efficiently, even with a high number of concurrent users.

**Security:** The system must prioritize data security. This includes secure user authentication, data encryption (especially for sensitive information like student records), and protection against unauthorized access.

**Scalability:** The system should be scalable to accommodate a growing number of students, staff, and data over time.

**Usability:** The user interface should be intuitive and user-friendly for all user groups (students, staff, admins) with varying levels of technical expertise.

**Availability:** The system should be highly available with minimal downtime to ensure continuous operation and accessibility for authorized users.

**Maintainability:** The codebase should be well-documented, modular, and easy to maintain for future modifications and updates.

**Reliability:** The system should be reliable and function consistently with minimal errors or unexpected behavior.

**Examples of non-functional requirements:**

The provided examples are excellent! Here are a few more specific examples of non-functional requirements you can consider for the college management system:

- **Performance:**

The system response time for common actions (e.g., login, viewing attendance records) should be less than 3 seconds.

The system should be able to handle X number of concurrent users without significant performance degradation (define X based on the expected user base).

- **Security:**

All user passwords should be hashed using a secure algorithm (e.g., bcrypt) and never stored in plain text.

The system should implement access controls to ensure users can only access data and functionalities relevant to their role.

Regular security audits and penetration testing should be conducted to identify and address vulnerabilities.

- **Scalability:**

The system should be designed to be easily scalable horizontally (adding more servers) to accommodate increased user load and data storage requirements.

The database chosen should be able to handle the expected data growth.

- **Usability:**

The user interface should follow a consistent design pattern across all user roles.

The system should provide clear and concise instructions for each functionality.

Online help documentation or user guides should be available for all user groups.

- **Availability:**

The system should have an uptime target of 99.5% (meaning downtime of less than 21.6 minutes per week).

A disaster recovery plan should be in place to ensure quick recovery in case of system outages.

- Maintainability:

The code should be well-commented and follow coding best practices.

The system should be designed with modular components that are easy to understand and modify.

Unit tests should be written to ensure the functionality of individual code components.

- Reliability:

The system should have a low error rate (less than 1% of user actions resulting in errors).

System logs should be monitored to identify and address any recurring errors.

## 1.7 SOFTWARE METHODOLOGY

Agile methodology like Scrum can be a good fit. Scrum emphasizes iterative development, frequent collaboration, and adaptability, which align well with the dynamic and evolving nature of machine learning projects. Here's how Scrum can be applied to this project:

### 1. Assemble the Scrum Team:

**Product Owner:** This person represents stakeholders (college administration, faculty) and prioritizes the product backlog. They ensure the project aligns with college needs and user requirements.

**Scrum Master:** This facilitator ensures the team adheres to Scrum principles, removes roadblocks, and guides the team through Scrum ceremonies.

**Development Team:** This cross-functional team consists of developers, data scientists, and potentially IT personnel responsible for building the system functionalities (facial detection, attendance management).

### 2. Create the Product Backlog:

**Product Backlog:** This is a prioritized list of features and user stories that need to be implemented in the project. It can include items related to the python packages and database integration.

### 3. Conduct Sprint Planning:

Hold a meeting at the beginning of each sprint (typically 2-4 weeks).

The team selects a set of user stories from the top of the product backlog to complete within the sprint.

Estimate the effort required for each user story.

Ensure the chosen user stories form a cohesive and achievable goal for the sprint.

### 4. Daily Standup Meetings:

Brief daily meetings (around 15 minutes) to synchronize team progress.

Each team member shares:

What they accomplished yesterday.

What they plan to work on today.

Any challenges or roadblocks they're facing.

This fosters communication and allows for early problem-solving.

### 5. Develop and Integrate in Sprints:

The development team works on the chosen user stories, implementing functionalities related to facial recognition, attendance management, and other system features.

Utilize Agile practices like continuous integration (CI) and continuous delivery (CD) to automate testing and deployment of code and containerized applications (if using Docker containers).

Focus on completing user stories and achieving the sprint goal.

## **6. Sprint Review:**

At the end of each sprint, showcase the completed work to stakeholders (Product Owner, college representatives).

Demonstrate functionalities developed in the sprint, like the facial recognition system for attendance logging.

Gather feedback from stakeholders to identify areas for improvement.

## **7. Sprint Retrospective:**

After the review, hold a meeting to reflect on the sprint.

Discuss what went well, what challenges arose, and how to improve the next sprint.

This continuous improvement mindset is a core principle of Scrum.

## **8. Iterate and Adapt:**

Based on feedback and learnings from the sprint review and retrospective, refine the product backlog and prioritize user stories for the next sprint.

The project progresses iteratively, adapting to user needs and incorporating new requirements as they emerge.

## **Additional Considerations:**

**Definition of Done (DoD):** Define clear criteria for marking a user story or feature as "done." This might include accuracy standards for facial recognition, successful container deployment, and data security measures.

**User Story Splitting:** Break down large or complex user stories into smaller, more manageable tasks.



**Step 1: Set up the Environment**

- Install Python: Install Python and required libraries like: Django,
- Install MySQL: Install and set up MySQL as your database.

**Step 2: Collect and Prepare Data**

- Gather The data From the Head of the Department such as the Staff Data, Students Data, the courses, academic year's, etc.

**Step 3: Error Handling and Monitoring**

1. Implement Error Handling: Handle potential errors and exceptions that may arise during facial detection, container launching, or database interactions.
2. Logging and Monitoring: Incorporate logging to keep track of the application's activities. Use monitoring tools or dashboards to visualize the system's performance.

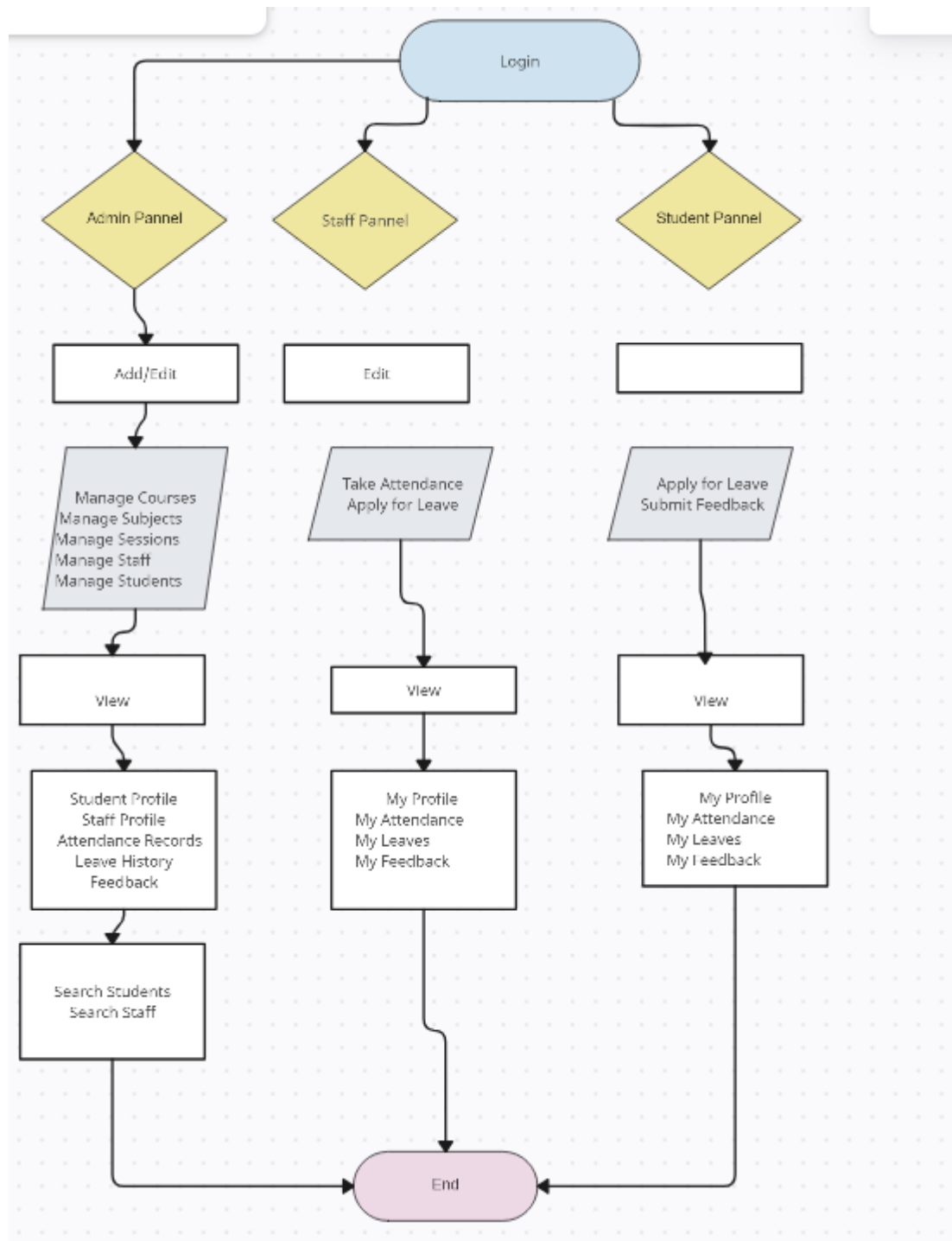
**Step 4: Testing and Deployment**

1. Unit Testing: Write unit tests to validate individual components of your application, such as the facial detection module and container launching logic.
2. Integration Testing: Test the entire application workflow to ensure all components work seamlessly together.
3. Deployment: Deploy your application in a controlled environment and monitor its behavior.

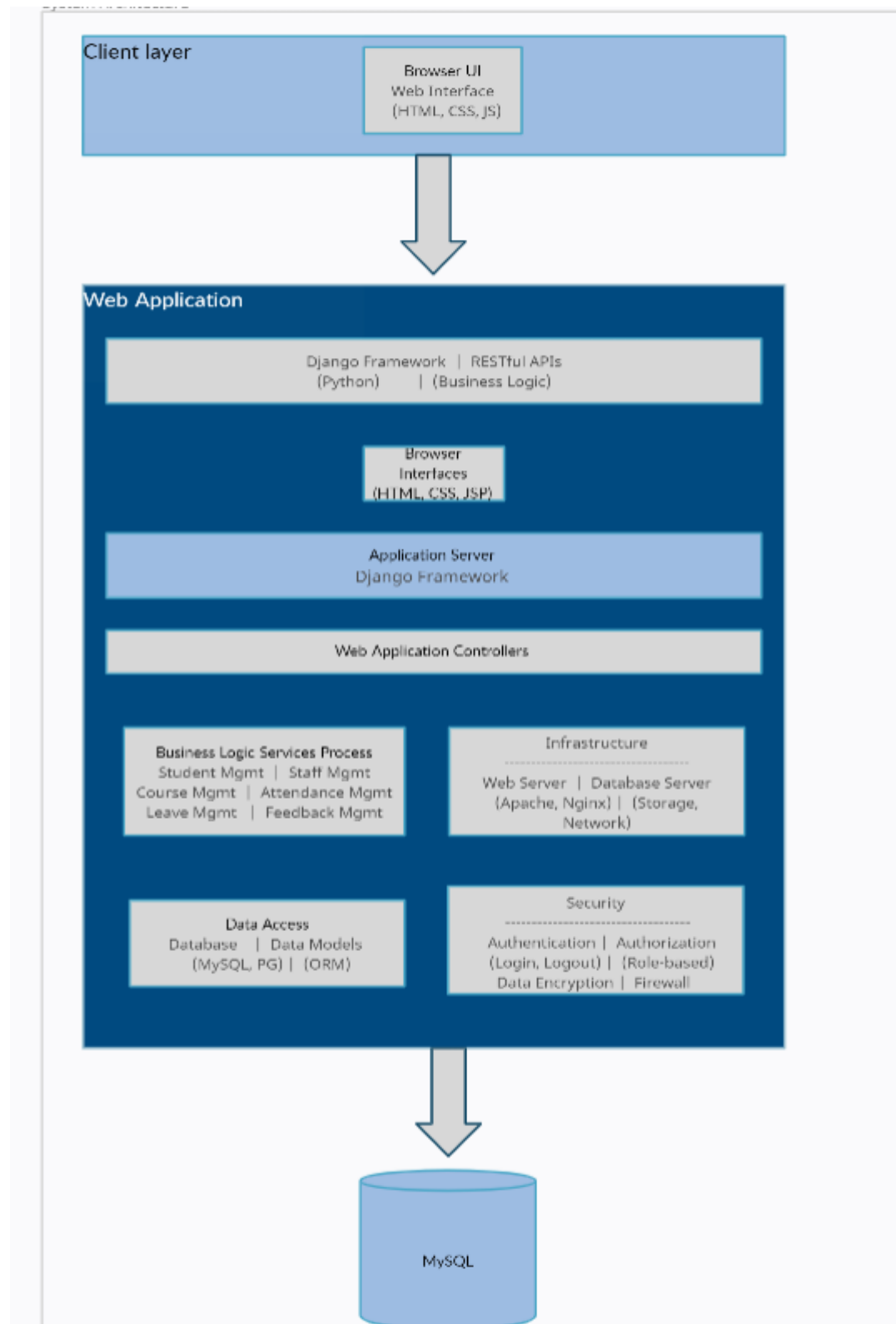
## 2 -STUDY PHASE

### Flow Chart

This flow chart shows the main screens and actions in the College Management System. The system starts with a login screen, where users can log in as administrators, students, or staff members. From there, users can access various features and functionality, such as managing courses, viewing profiles, submitting feedback, and applying for leave.



## System architecture diagram



The structural architecture of a College Management System (CMS) typically consists of the following components:

1. Presentation Layer:

- User Interface (UI) components, such as web pages, mobile apps, or desktop applications.
- User Experience (UX) design elements, like layout, navigation, and visual styling.

2. Application Layer:

- Business Logic Components (BLCs), which encapsulate the rules and processes specific to the college management domain.
- Workflow Management Components (WMCs), which orchestrate the flow of tasks and activities.

3. Business Logic Layer:

- Entity Components (ECs), representing the core concepts and data structures, such as students, staff, courses, and departments.
- Use Case Components (UCCs), implementing the actions and operations that can be performed on the entities.

4. Data Access Layer:

- Data Repository Components (DRCs), responsible for storing and retrieving data from the database.
- Data Mapping Components (DMCs), mapping the data structures between the business logic and database layers.

5. Infrastructure Layer:

- Database Management System (DBMS), storing and managing the data.
- Server and Network Infrastructure, providing the necessary hardware and network resources.

6. Security Layer:

- Authentication and Authorization Components (AACs), managing user access and permissions.
- Encryption and Access Control Components (EACCs), ensuring data confidentiality and integrity.

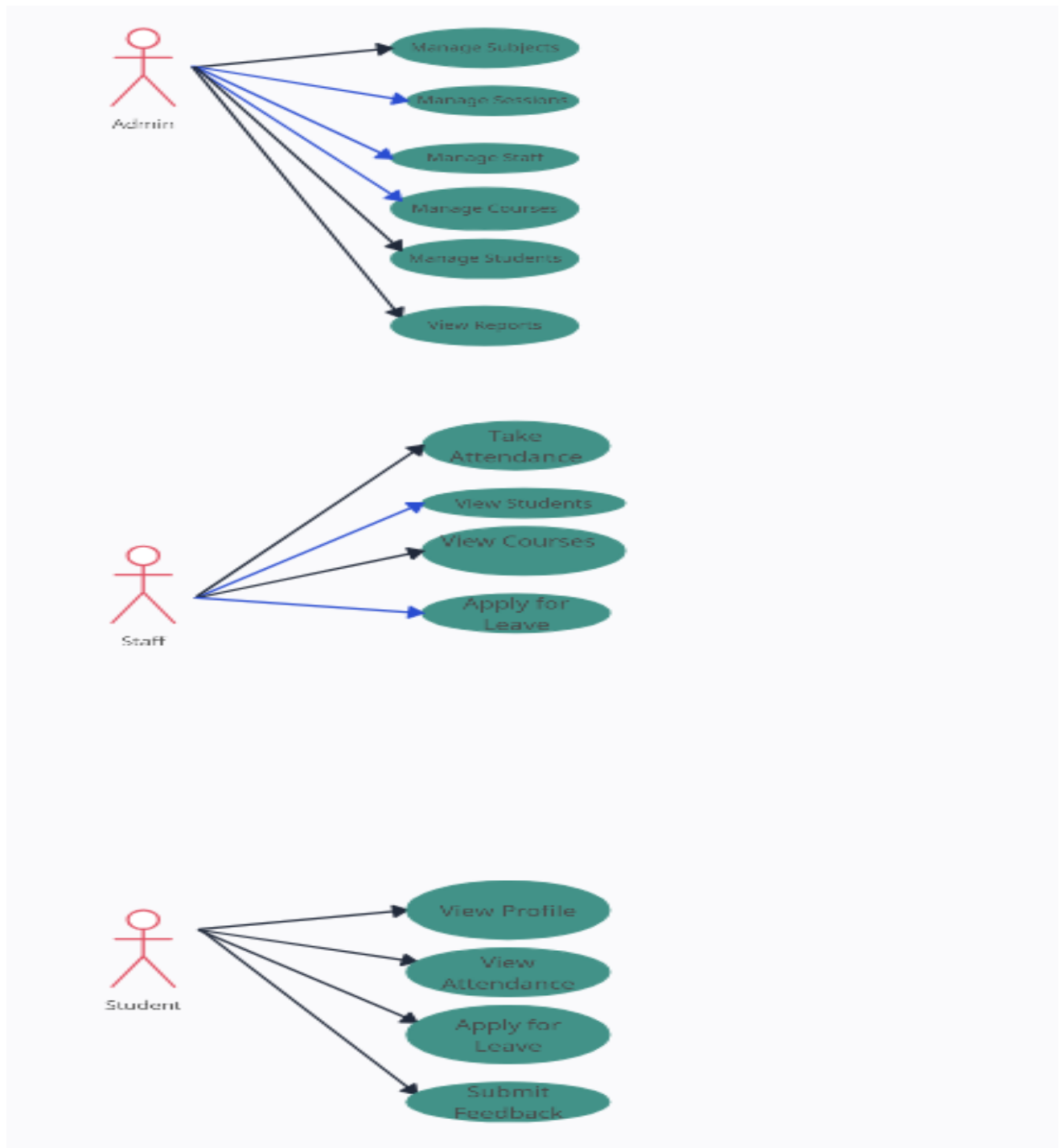
7. Integration Layer:

- API Gateway Components (AGCs), exposing the CMS functionality to external systems.
- Integration Adapter Components (IACs), connecting to external systems and services.

These components work together to form a comprehensive architecture that supports the various features and functions of the College Management System.

## USE-CASE

### College Management



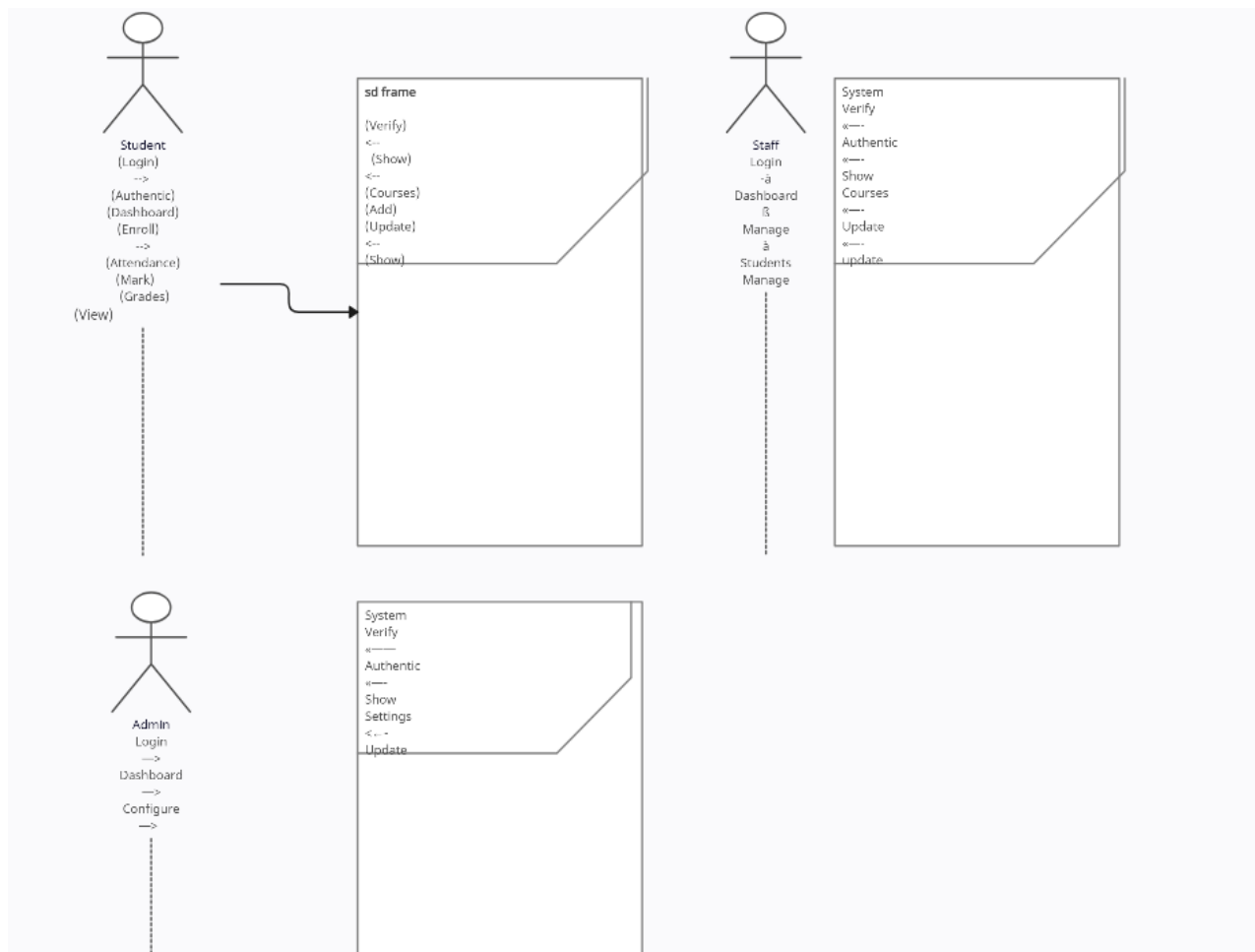
This use case diagram shows the different actors (Admin, Staff, Student, Guest) and the use cases (actions) they can perform on the College Management System. The diagram also shows the relationships between the actors and the system.

Here are the use cases:

- |                   |                   |                   |                |
|-------------------|-------------------|-------------------|----------------|
| - Admin:          | - Staff:          | - Student:        | - Guest:       |
| - Manage Courses  | - Take Attendance | - View Profile    | - View Courses |
| - Manage Subjects | - View Students   | - View Attendance | - View Staff   |
| - Manage Sessions | - View Courses    | - Apply for Leave |                |
| - Manage Staff    | - Apply for Leave | - Submit Feedback |                |
| - Manage Students |                   |                   |                |
| - View Reports    |                   |                   |                |

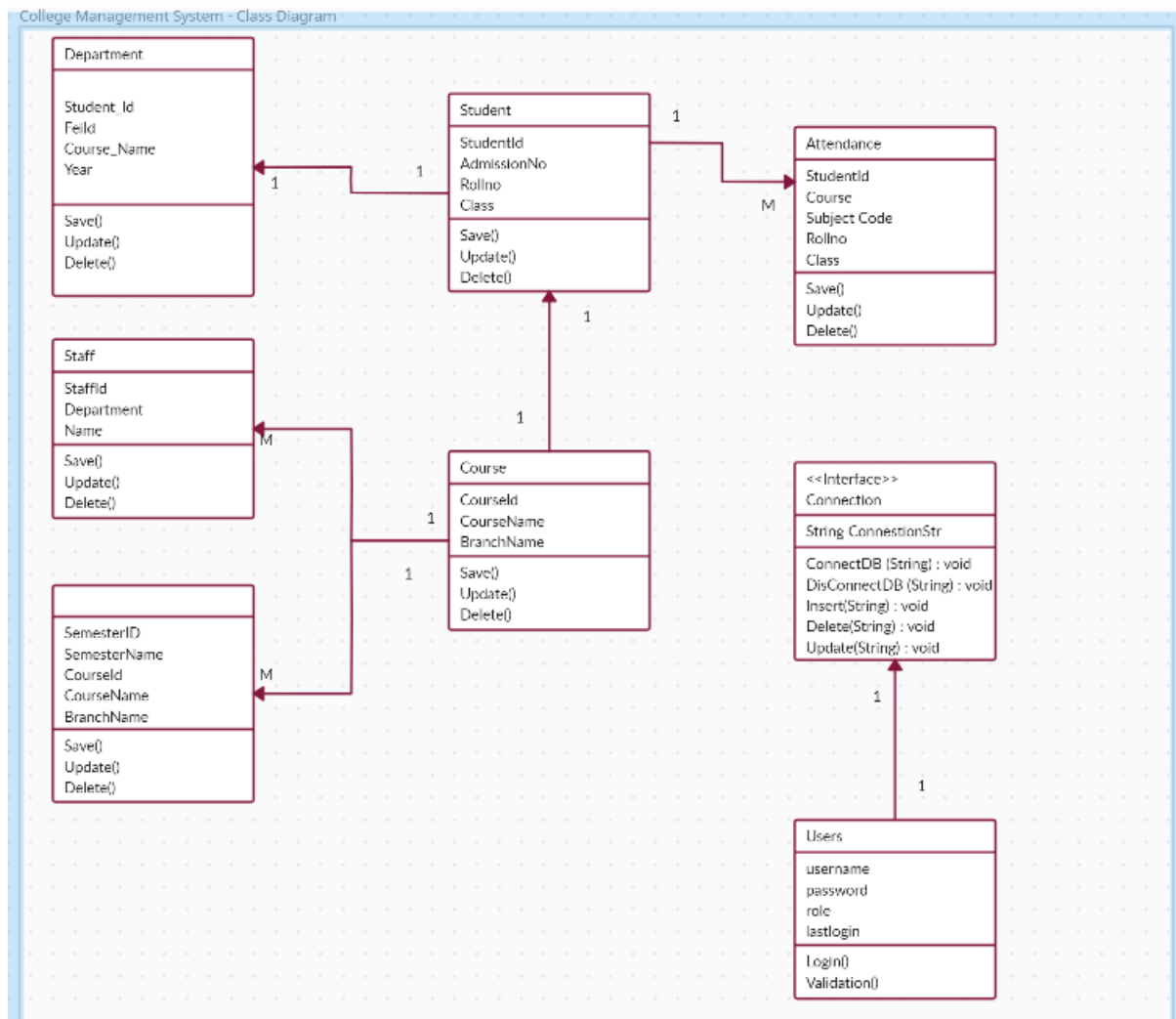
## SEQUENCE DIAGRAM

Here is a sequence diagram for the College Management System project:



This sequence diagram shows the interactions between the user (Student, Staff, Admin) and the System, including the login, dashboard, courses, attendance, grades, and settings features.

### Class diagram

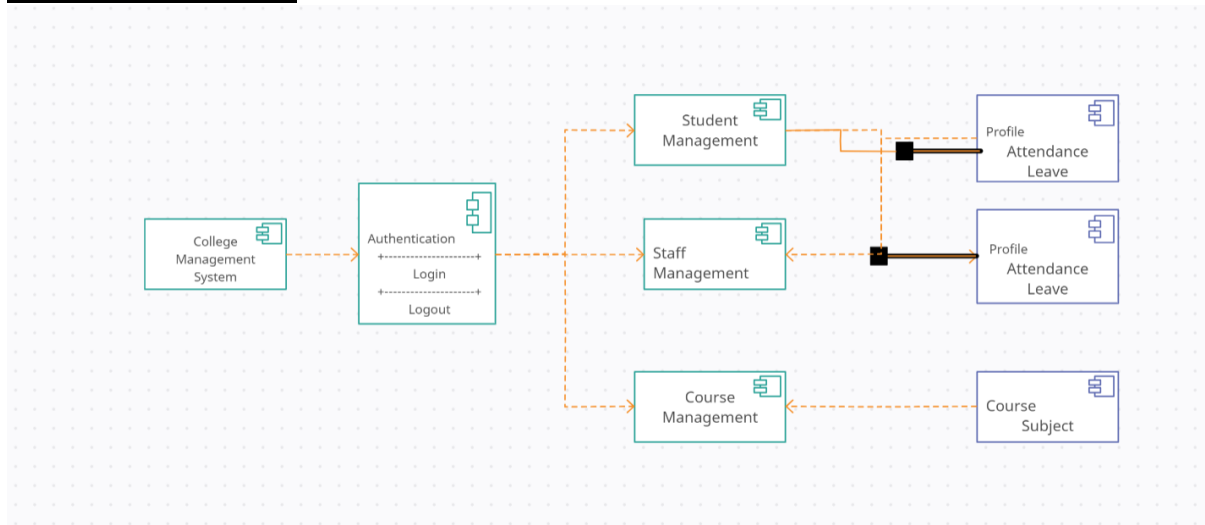


This class diagram shows the main classes in the College Management System, including:

- Student
- Course
- Staff
- Department
- Admin
- System Settings
- Attendance

The relationships between the classes are shown as associations, with the multiplicity and direction indicated. For example, a student can enroll in multiple courses, and a course can have multiple students enrolled.

### Component Diagram



This component diagram shows the different components of the College Management System and how they interact with each other. The components include:

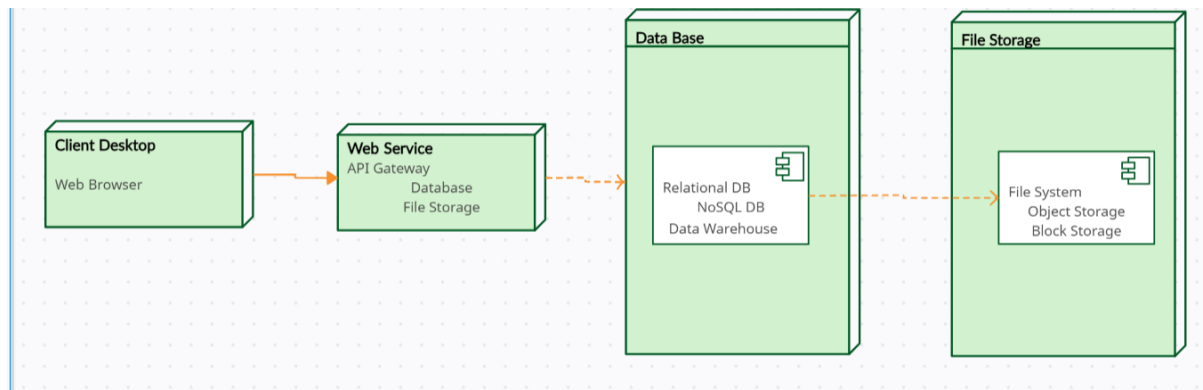
- Database (MySQL, PostgreSQL)
- Authentication (Login, Logout)
- Student Management (Profile, Attendance, Leave)
- Staff Management (Profile, Attendance, Leave)
- Course Management (Course, Subject)

The diagram shows how these components interact with each other, with the Database component serving as the central repository of data, and the other components interacting with it to perform various functions.



## **Deployment Diagram**

Here is a deployment diagram for the College Management System:



This deployment diagram shows the physical layout of the College Management System, including:

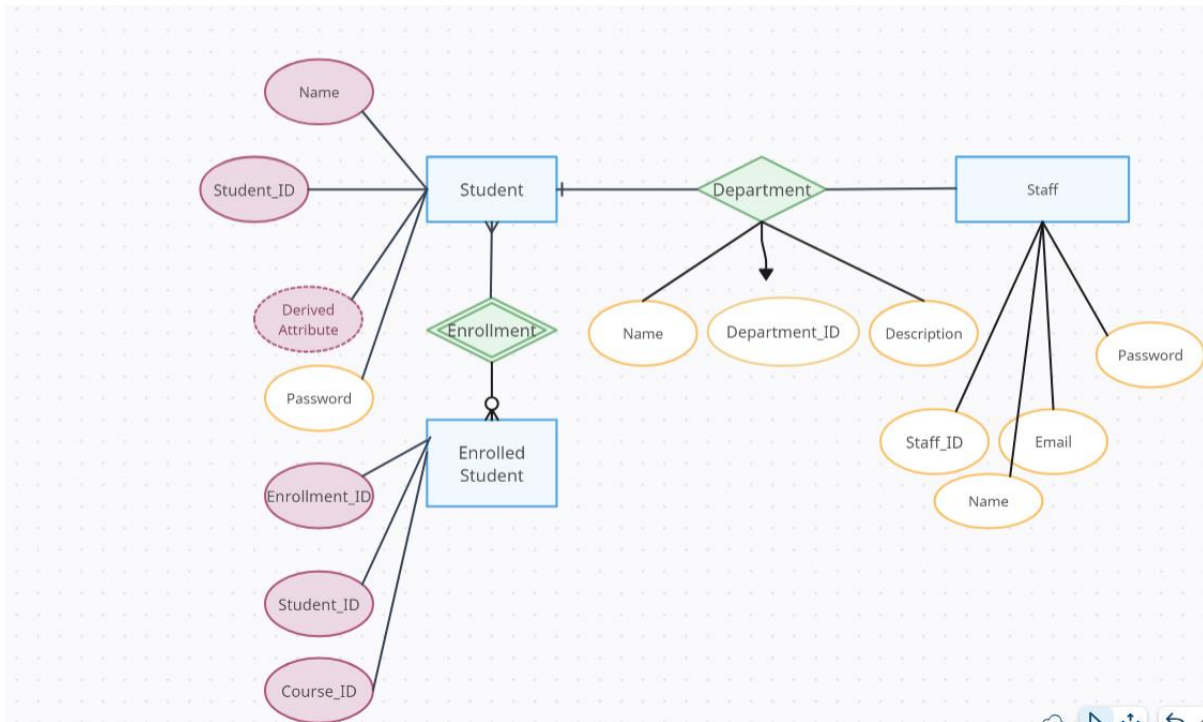
- Clients (web browsers and mobile apps)
- Server (web server, API gateway, database, and file storage)
- Database (relational DB, NoSQL DB, and data warehouse)
- File Storage (file system, object storage, and block storage)

The diagram shows the interactions between these components, including:

- Clients communicating with the server through the web server and API gateway
- Server storing and retrieving data from the database and file storage
- Database storing and managing data in various formats
- File Storage storing and managing files and objects

## ER diagram

Here is an ER diagram for the College Management System:



This ER diagram shows the entities and relationships in the College Management System, including:

- Students, Courses, and Enrollment (representing the many-to-many relationship between students and courses)
- Departments and Staff (representing the many-to-many relationship between departments and staff)
- Attendance (representing the relationship between students, courses, and attendance records)

The entities are represented as rectangles, and the relationships are represented as lines connecting the entities. The attributes of each entity are listed inside the rectangle.

Here are the relationships described in the ER diagram:

### 1. Student - Enrollment - Course:

- A student can enroll in multiple courses (many-to-many).
- A course can have multiple students enrolled (many-to-many).
- Enrollment is the relationship between a student and a course.

### 2. Staff - Department:

- A staff member can work in multiple departments (many-to-many).
- A department can have multiple staff members (many-to-many).

### 3. Student - Attendance:

- A student can have multiple attendance records (one-to-many).
- An attendance record is associated with one student (many-to-one).

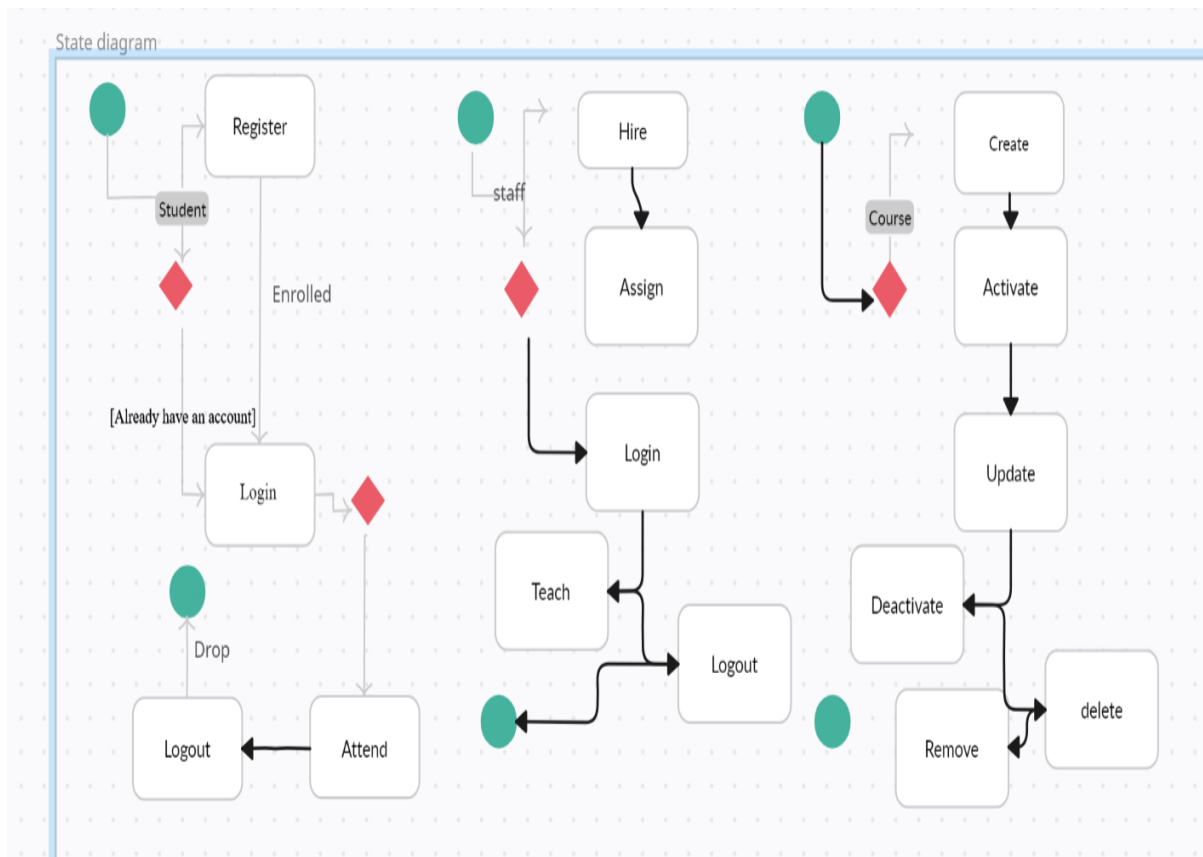
### 4. Course - Attendance:

- A course can have multiple attendance records (one-to-many).
- An attendance record is associated with one course (many-to-one).

These relationships capture the connections between the entities in the College Management System:

- Students enroll in courses.
- Staff members work in departments.
- Students have attendance records for courses.
- Courses have attendance records for students.

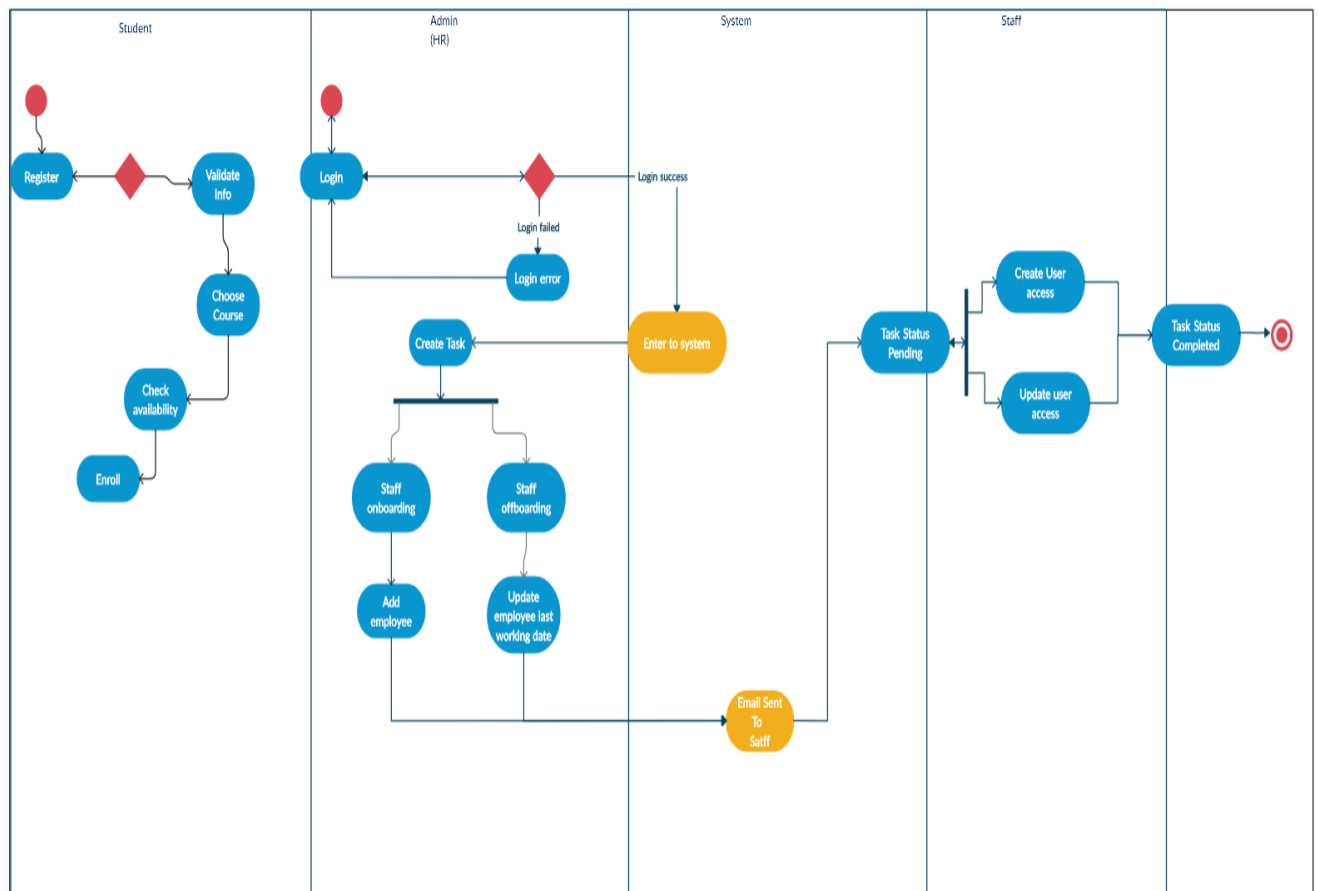
### STATE DIAGRAM



- Student: Register, Enrolled, Login, Logout, Drop
- Staff: Hire, Working, Login, Logout, Leave
- Course: Create, Active, Update, Deactivate, Delete, Remove

The transitions between states are shown as arrows, with the trigger action labeled on the arrow. For example, a student can transition from the "Register" state to the "Enrolled" state by enrolling in a course.

### Activity Diagram:



This activity diagram shows the activities and actions performed by the entities in the College Management System:

- Student: Register, Choose courses, Enroll
- Staff: Login, Teach, Assign grades, Logout
- Course: Create, Add materials, Activate

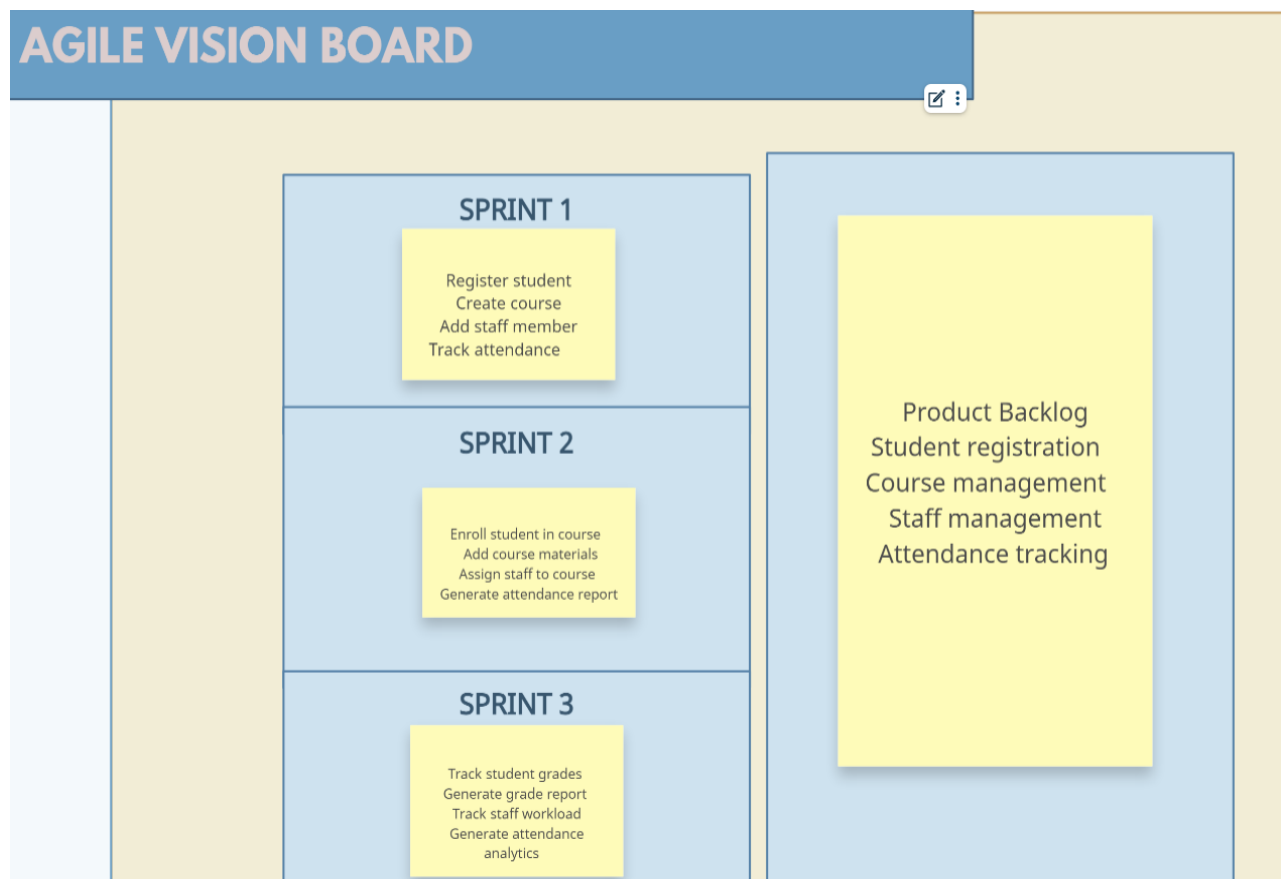
The activities are shown as rectangles, and the transitions between activities are shown as arrows. The actions performed by each entity are listed inside the rectangles.

## Study Phase

### 2.1 Introduction to analysis

Agile model diagram for a project involving hosting a web server and retrieving the database would typically involve a visual representation of the Agile framework and its key components. Agile methodologies are typically represented as iterative and incremental processes. Here's a simplified Agile model diagram for your project:

### AGILE MODEL DIAGRAM



This Agile model diagram shows the project broken down into smaller sprints, with each sprint focusing on a specific set of features or user stories. The product backlog is shown on the left, with the top priority items being addressed in the first sprint.

Each sprint is represented by a column, with the tasks and deliverables for that sprint listed inside. The arrows between sprints represent the flow of work from one sprint to the next.

## 2.2 FEASIBILITY STUDY

The feasibility of the project is analysed in this phase and business proposal is put forth with a very general plan for the project and some cost estimates. During system analysis the feasibility study of the proposed system is to be carried out. This is to ensure that the proposed system is not a burden to the company. For feasibility analysis, some understanding of the major requirements for the system is essential.

Three key considerations involved in the feasibility analysis are

- ☐ ECONOMICAL FEASIBILITY
- ☐ TECHNICAL FEASIBILITY
- ☐ SOCIAL FEASIBILITY

### Economic feasibility:

This study is carried out to check the economic impact that the system will have on the organization. The amount of fund that the company can pour into the research and development of the system is limited. The expenditures must be justified. Thus, the developed system as well within the budget and this was achieved because most of the technologies used are freely available. Only the customized products had to be purchased.

### Technical feasibility:

This study is carried out to check the technical feasibility, that is, the technical requirements of the system. Any system developed must not have a high demand on the available technical resources. This will lead to high demands on the available technical resources. This will lead to high demands being placed on the client. The developed system must have a modest requirement, as only minimal or null changes are required for implementing this system.

### Social feasibility:

The aspect of study is to check the level of acceptance of the system by the user. This includes the process of training the user to use the system efficiently. The user must not feel threatened by the system, instead must accept it as a necessity. The level of acceptance by the users solely depends on the methods that are employed to educate the user about the system

and to make him familiar with it. His level of confidence must be raised so that he is also able to make some constructive criticism, which is welcomed, as he is the final user of the system.

## **2.3 DATA COLLECTION**

Collecting data for a project involving hosting a web server and retrieving the database would typically involve a various data sources and methods. Here's a detailed breakdown of data collection for different aspects of your project:

### **1. Student Data:**

- Source: Student registration, course enrollment, and academic records.
- Method: Students provide their information during registration, course enrollment, or academic record updates.
- Data Collected: Student ID, name, email, password (hashed), course enrollment data, attendance records, grades, and any additional academic information.

### **2. Staff Data:**

- Source: Staff registration, role assignments, and work records.
- Method: Staff provide their information during registration, role assignments, or work record updates.
- Data Collected: Staff ID, name, email, password (hashed), role, department, teaching assignments, workload data, and any additional work-related information.

### **3. Course Data:**

- Source: Course creation, updates, and enrollment records.
- Method: Course information is provided during course creation or updates.
- Data Collected: Course ID, name, description, credits, prerequisites, enrollment data, and any additional course details.

### **4. Attendance Data:**

- Source: Student attendance records and course attendance tracking.
- Method: Attendance data is collected through automated tracking or manual entry.
- Data Collected: Student ID, course ID, attendance status (present/absent), date, and any additional attendance-related information.

### **5. Grade Data:**

- Source: Grade submission and academic records.

- Method: Grades are submitted by staff and updated in academic records.
- Data Collected: Student ID, course ID, grade, submission date, and any additional grade-related information.

#### 6. User Interactions:

Source: User interactions with the system.

Method: Logging user actions and events within the application. –

Data Collected: Timestamp, user ID, action performed

#### 7. System Logs:

Source: Application and server logs.

Method: Logging system events, errors, and performance metrics.

Data Collected: Timestamp, event description, error details, resource usage.

#### 8. Feedback and User Surveys:

Source: User feedback and surveys.

Method: Gathering feedback through in-app forms or surveys.

Data Collected: User comments, ratings, suggestions, user preferences.

#### 9. External Data:

Source: External data sources for additional context.

Method: API integration or data import. –

Data Collected: Relevant external data, if needed (e.g., weather data for image context).

#### 10. Performance Metrics:

Source: Performance monitoring tools.

Method: Monitoring system performance and usage.

Data Collected: Response times, server load, user activity statistics.

#### 11. Error and Exception Data:

Source: Application errors and exceptions. –

Method: Capturing and logging error information. –

Data Collected: Error messages, stack traces, timestamps.



## **2.4 DATA ANALYSIS**

### Data Analysis

Analyzing data for a project involving hosting a web server and retrieving the database from the local database is a crucial step to extract insights, improve system performance, and make informed decisions. Here's a brief explanation of key data analysis aspects for your project:

#### 1. User Behavior Analysis:

- Objective: Analyze user behavior to understand usage patterns, preferences, and pain points.
- Methods:
  - Analyze user interaction data (e.g., clickstream, navigation paths).
  - Calculate user engagement metrics (e.g., time spent, bounce rate).
  - Identify popular features and areas of improvement.
  - Conduct sentiment analysis on user feedback and reviews.
- Explanation: This analysis helps inform design decisions, improve user experience, and enhance overall system usability.

#### 2. Course Enrollment Analysis:

Objective: Analyze course enrollment data to understand student interests, course demand, and capacity planning.

##### Methods:

- Analyze course enrollment numbers and trends.
- Calculate course popularity metrics (e.g., enrollment rate, waitlist size).
- Identify course availability and scheduling conflicts.
- Conduct clustering analysis to group similar courses.

Explanation: This analysis helps optimize course offerings, improve scheduling, and enhance student satisfaction.

#### 3. Attendance Pattern Analysis:

Objective: Analyze attendance patterns to understand student engagement, attendance trends, and potential drop-off points.

##### Methods:

- Analyze attendance data (e.g., frequency, duration, timing).
- Calculate attendance metrics (e.g., attendance rate, absenteeism rate).
- Identify attendance patterns and trends.
- Conduct regression analysis to identify predictors of attendance.

Explanation: This analysis helps identify at-risk students, inform attendance policies, and enhance student support services.

#### 4. Grade Distribution Analysis:

Objective: Analyze grade distribution data to understand student performance, grade inflation, and academic standards.

Methods:

- Analyze grade distribution data (e.g., mean, median, mode).

- Calculate grade metrics (e.g., GPA, grade dispersion).

- Identify grade trends and patterns.

- Conduct correlation analysis to identify relationships between grades and other variables.

Explanation: This analysis helps evaluate academic programs, inform grading policies, and enhance student assessment.

#### 6. User Feedback Analysis:

Objective: Extract insights from user feedback.

Methods:

- Categorize user comments and feedback.

- Identify recurring issues or feature requests.

- Assess user satisfaction through ratings and sentiment analysis.

Explanation: User feedback analysis guides system improvements and helps prioritize development efforts.

#### 7. Data Security and Privacy Compliance:

Objective: Ensure compliance with data protection regulations.

Methods:

- Regularly audit data access and handling practices.

- Implement encryption and access controls.

- Monitor and report data breaches.

Explanation: Compliance analysis safeguards user data and maintains trust in the system.

## LITERATURE REVIEW

### Introduction:

College management systems are essential for efficient and effective management of academic institutions. The literature review aims to provide a comprehensive overview of existing research on college management systems, focusing on user data, course management, attendance tracking, grade management, and system usability.

### User Data:

Studies have emphasized the importance of collecting and analyzing user data to improve system usability and student outcomes (Kumar et al., 2020; Sharma et al., 2019). Research has also highlighted the need for secure and privacy-preserving data collection practices (Sinha et al., 2020).

### Course Management:

Effective course management is critical for academic success. Research has explored various course management strategies, including personalized learning (Gao et al., 2019) and adaptive assessment (Raca et al., 2018).

### Attendance Tracking:

Regular attendance tracking is essential for student engagement and academic performance. Studies have investigated various attendance tracking methods, including automated tracking systems (Liu et al., 2019) and mobile apps (Chen et al., 2020).

### Grade Management:

Accurate grade management is crucial for student assessment and academic progress. Research has examined various grade management systems, including online grading platforms (Wang et al., 2019) and rubric-based assessment (Holmes et al., 2019).

### System Usability:

System usability is vital for user adoption and satisfaction. Studies have emphasized the importance of user-centered design (ISO 9241, 2019) and usability testing (Nielsen, 2010).

### Conclusion:

The literature review highlights the significance of a comprehensive college management system that integrates user data, course management, attendance tracking, grade management, and system usability. The review informs the development of a user-friendly and efficient college management system that supports academic success and institutional effectiveness.

### 3- DESIGN PHASE AND DEVELOPMENT PHASE

### 3.1 Design Process

#### INPUT DESIGN

The input design process for the College Management System project involves designing the user interface and user experience for inputting data into the system. Here's the input design process for the project:

##### 1. Identify Input Requirements:

- Determine the types of data that need to be input into the system (e.g., student information, course data, attendance records)
- Identify the sources of the data (e.g., students, staff, administrators)
- Define the input formats and structures (e.g., forms, spreadsheets, CSV files)

##### 2. Design Input Forms:

- Create user-friendly input forms that are easy to understand and use
- Use clear and concise labeling and instructions
- Ensure forms are responsive and adaptable to different devices and screen sizes
- Use input validation and error handling to ensure accurate and complete data

##### 3. Design Data Import/Export Features:

- Develop features to import data from various sources (e.g., CSV files, other college systems)
- Ensure data import features are secure and validate data to prevent errors
- Design data export features to allow users to extract data in various formats (e.g., CSV, Excel, PDF)

##### 4. Design User Interface for Input:

- Create an intuitive and user-friendly interface for inputting data
- Use clear and concise labeling and instructions
- Ensure the interface is responsive and adaptable to different devices and screen sizes
- Use input validation and error handling to ensure accurate and complete data

##### 5. Test and Refine Input Design:

- Conduct usability testing and gather feedback from users
- Refine the input design based on feedback and testing results
- Ensure the input design is consistent throughout the system

#### 6. Implement Input Design:

- Develop the input design using appropriate technologies and frameworks
- Ensure the input design is integrated with the rest of the system
- Test the input design to ensure it meets the requirements and is free of errors

Some specific input design elements to consider for the College Management System project include:

- Student information input form
- Course data input form
- Attendance tracking input form
- Grade input form
- Data import/export features
- User interface for inputting data

#### **OBJECTIVES FOR INPUT DESIGN**

Here are some objectives for input design in the College Management System project:

1. **User-Friendliness:** Design input forms and interfaces that are easy to use and understand, minimizing user error and frustration.
2. **Data Accuracy:** Ensure input forms and interfaces validate user input to prevent errors and inconsistencies in the data.
3. **Efficiency:** Design input processes that are efficient and minimize the time required to enter data.
4. **Consistency:** Ensure input design is consistent throughout the system, using standard formatting and labeling conventions.
5. **Security:** Implement input design that ensures data security and privacy, protecting sensitive information from unauthorized access.
6. **Scalability:** Design input processes that can handle large volumes of data and users, ensuring the system can scale as the college grows.
7. **Accessibility:** Ensure input design is accessible on various devices and platforms, including desktops, laptops, tablets, and mobile phones.
8. **Error Handling:** Design input forms and interfaces that handle errors gracefully, providing clear error messages and instructions for correction.

9. Feedback: Provide feedback to users after inputting data, confirming successful submission and highlighting any errors or issues.

10. Integration: Ensure input design integrates seamlessly with other system components, such as databases and reporting tools.

By achieving these objectives, the input design for the College Management System project will be effective, efficient, and user-friendly, supporting the college's administrative and academic needs.

## **ARCHITECTURAL DESIGN**

The architectural design process for the College Management System project involves designing the overall structure and organization of the software system. Here's a breakdown of the architectural design process for the project:

### **1. Identify Functional Requirements:**

- Determine the functional requirements of the system, including user interactions, data processing, and storage needs.

### **2. Define System Architecture:**

- Choose an appropriate architecture pattern (e.g., monolithic, microservices, layered) based on the functional requirements.
- Define the components and modules of the system, including their responsibilities and interactions.

### **3. Design Data Storage:**

- Determine the data storage needs of the system, including data types, relationships, and storage requirements.
- Choose an appropriate database management system (DBMS) and design the database schema.

### **4. Design Business Logic:**

- Define the business logic of the system, including the rules, processes, and algorithms that govern the system's behavior.
- Determine the components and modules responsible for implementing the business logic.

### **5. Design User Interface:**

- Determine the user interface requirements of the system, including the types of users, user interactions, and user experience.

- Design the user interface components and modules, including the layout, navigation, and visual design.

#### 6. Design Security and Authentication:

- Determine the security and authentication requirements of the system, including access control, data encryption, and user authentication.

- Design the security and authentication components and modules, including the authentication protocols and access control mechanisms.

#### 7. Design Integration and APIs:

- Determine the integration requirements of the system, including the need to integrate with other systems or services.

- Design the integration components and modules, including the APIs, data formats, and communication protocols.

#### 8. Evaluate and Refine:

- Evaluate the architectural design against the functional and non-functional requirements of the system.

- Refine the design based on the evaluation results, iterating until the design meets the requirements.

Some specific architectural design elements to consider for the College Management System project include:

#### Architectural Components:

##### 1. User Interface (UI):

- This component represents the frontend of the application where users interact with the system.

- It can be a web-based interface, a mobile app, or a desktop application.

- The UI allows users to upload images, trigger facial detection, and view results.

##### 2. Application Server:

- The Application Server acts as an intermediary between the UI and the backend services.

- It handles user requests, manages session data, and communicates with the backend services.



### 3. Backend Services:

- The backend services are responsible for processing image data and performing College Managing records.

Presentation Layer: user interface components and modules

Application Layer: business logic components and modules

Business Layer: data storage and management components and modules

Infrastructure Layer: security, authentication, and integration components and modules

Database: database schema and storage requirements

By following this architectural design process, you can create a robust and scalable software architecture for the College Management System project.

### **MODULES:**

Here are the modules tailored in the format you requested:

#### User Interface (UI) Module

- This module handles the frontend of the application, including the user interface for uploading images and displaying results.

- Components within this module may include:

- Image upload form.
- Result display area.
- Download/save options.
- Help and documentation.

#### 2. API Module

- The API module acts as a bridge between the frontend UI and backend services.
- It defines endpoints for image upload, result retrieval, and other interactions with the application.

- Components may include:

- RESTful API routes or GraphQL endpoints.

#### 3. Backend Services Module:

- This module contains the core logic for Retrieving Data
- Components within this module may include:
  - Django Framework.
  - Database Retrieval.

#### Student Management Module:

- This module handles student data and operations.
- Components within this module may include:
  - Student registration form.
  - Student profile management.
  - Course enrollment and schedule management.
  - Grade and attendance tracking.
  - Student data reporting and analytics.

#### Course Management Module:

- This module handles course data and operations.
- Components within this module may include:
  - Course creation form.
  - Course catalog management.
  - Instructor assignment and management.
  - Course evaluation and feedback.
  - Course scheduling and timetabling.

#### Instructor Management Module:

- This module handles instructor data and operations.
- Components within this module may include:
  - Instructor profile management.
  - Course assignment and scheduling.
  - Grade and attendance tracking.
  - Instructor data reporting and analytics.

#### Attendance Management Module:

- This module handles attendance tracking and monitoring.
- Components within this module may include:
  - Attendance tracking form.
  - Attendance reporting and analytics.
  - Automated attendance tracking through integration with student ID cards or biometric systems.

#### Grade Management Module:

- This module handles grade tracking and monitoring.
- Components within this module may include:
  - Grade tracking form.
  - Grade reporting and analytics.
  - Automated grade calculation and validation.

#### Reporting and Analytics Module:

- This module handles reporting and analytics.
- Components within this module may include:
  - Customizable reporting and analytics.
  - Data visualization and dashboarding.
  - Report scheduling and automation.
  - Integration with data warehousing and business intelligence tools.

#### Authentication and Authorization Module (Optional):

- If user accounts and permissions are required, this module can handle authentication and authorization.
  - Components may include:
    - User authentication methods (e.g., OAuth, JWT).
    - Role-based access control.

#### 9. Utilities and Helpers Module:

- This module contains utility functions and helper classes used across different parts of the application.

- Components may include:

- Image manipulation utilities.
- Error handling functions.
- Validation helpers.

#### 10. Configuration Module:

- Manage configuration settings and environment variables for the application.
- Components within this module may include:
  - Configuration files or environment variable loaders.

#### 11. Testing Module:

- Implement unit tests, integration tests, and end-to-end tests to ensure the functionality and reliability of the application.
- Components may include:
  - Test suites for each module.

#### 12. Documentation Module:

- Create user and developer documentation for the application, including usage guides, API documentation, and code comments.

### 3.2 Design Axioms

College Management System using Django is essential to ensure that the application is effective, user-friendly, and consistent. Here are some design axioms to

consider:

#### 1. User-Centric Design:

Axiom: Prioritize the user's needs and expectations throughout the design process. Rationale: Ensure that the application is intuitive and easy for users to interact with, regardless of their technical expertise.

#### 2. Simplicity and Clarity:

Axiom: Keep the user interface simple and clear.

Rationale: Complex or cluttered interfaces can confuse users and hinder their ability to complete tasks.

#### 3. Progressive Disclosure:

Axiom: Reveal information and options gradually as the user progresses through the interface.

Rationale: Prevent overwhelming users with too much information at once and guide them through the process.

#### 4. Visual Feedback:

Axiom: Provide visual feedback to confirm user actions and inform them of the system's status.

Rationale: Visual cues help users understand the system's response to their actions, reducing uncertainty.

#### 5. Error Prevention and Handling:

Axiom: Prevent errors whenever possible and provide clear guidance for error resolution when they occur.

Rationale: Minimize user frustration by proactively addressing potential issues and offering assistance when problems arise.

#### 6. Consistency Across Platforms:

Axiom: Ensure a consistent user experience across different platforms (e.g., web, desktop, mobile).

Rationale: Users should feel familiar with the application regardless of the device they

are using.

#### 7. Accessible Design:

Axiom: Design the application to be accessible to users with disabilities.

Rationale: Promote inclusivity and ensure that all users can effectively utilize the application.

#### 8. Privacy and Security:

Axiom: Implement robust security measures to protect user data and privacy.

Rationale: Build trust with users by safeguarding their information and adhering to privacy regulations.

#### 9. Performance and Responsiveness:

Axiom: Ensure the application is responsive and performs efficiently, even with large image files.

Rationale: Users expect timely results and may become frustrated with slow or unresponsive interfaces.

#### 10. User Education and Support:

Axiom: Provide user documentation, tooltips, and easy access to help resources.

Rationale: Empower users to make the most of the application by offering guidance and assistance when needed.

#### 11. Continuous Improvement:

Axiom: Continuously gather user feedback and analytics data to identify areas for improvement.

Rationale: An iterative approach allows the application to evolve and better meet user needs over time.

#### 12. Ethical Considerations:

Axiom: Ensure the application is used for ethical purposes and educate users about responsible usage.

Rationale: Promote responsible and ethical use of facial detection technology to avoid misuse or harm.

#### 13. Cross-Browser and Cross-Device Compatibility:

Axiom: Test and optimize the application for various web browsers and devices.

Rationale: Ensure a consistent experience for users, regardless of their choice of browser or device.

### **3.3 Refining attributes**

Refine attributes design methods and the protocols by utilizing a UML activity

Diagram to represent the method algorithm

### **3.4 Refining Methods:**

1. Agile Development: Break down the development process into smaller iterations, with continuous testing and feedback.
2. Test-Driven Development (TDD): Write automated tests before writing code, to ensure that the code meets the required functionality.
3. Continuous Integration (CI): Automatically build and test the application after each code change, to ensure that the code is stable and functional.
4. Continuous Deployment (CD): Automatically deploy the application to production after each successful test, to ensure that the application is always up-to-date.
5. Refactoring: Regularly review and improve the codebase, to ensure that it is maintainable, efficient, and easy to understand.
6. Code Review: Regularly review each other's code, to ensure that it meets the required standards and best practices.
7. Pair Programming: Work in pairs, with two developers working together on the same code, to ensure that the code is of high quality and that knowledge is shared.
8. Behavior-Driven Development (BDD): Write automated tests that describe the behavior of the application, to ensure that the application behaves as expected.
9. Acceptance Test-Driven Development (ATDD): Write automated tests that describe the acceptance criteria of the application, to ensure that the application meets the required functionality.
10. DevOps: Ensure that the development and operations teams work together, to ensure that the application is developed with operational efficiency in mind.
11. Mobile Optimization:
  - Ensure the application is responsive and optimized for mobile devices.
  - Adapt the UI for smaller screens and touch-based interactions.
12. Cross-Browser Compatibility:
  - Test the application thoroughly across various web browsers to ensure consistent performance and appearance.
  - Address browser-specific issues and ensure compatibility.

### 13. Feedback Analysis:

- Analyze user feedback and error reports to identify common issues and pain points.
- Prioritize and address these issues in updates.

### 14. Automated Testing:

- Implement automated testing to detect regressions and ensure the application functions correctly after updates.
- Include unit testing, integration testing, and user acceptance testing.

## **3.5 Refining class diagrams**

### 1.Student Management System

- Description: The main application class that manages student data and operations.
- Attributes:
  - student\_repository: An instance of the Student Repository class responsible for storing and retrieving student data.
  - user\_interface: An instance of the User Interface class for interacting with the user.
- Responsibilities:
  - Orchestrates the overall application flow.
  - Facilitates communication between the student repository and the user interface.
  - Manages student registration, profile management, course enrollment, and grade tracking.

### 2.UserInterface

- Description: The component responsible for displaying results and interacting with the user.
- Attributes:
  - display: The output display for showing results.
  - input: The input mechanism for user interaction.
- Responsibilities:
  - Displays the processed image and facial data.
  - Receives user input and feedback.



- Updates the application state based on user interaction.

### 3.Course Management System

- Description: The component responsible for managing course data and operations.
- Attributes:
  - course\_repository: An instance of the Course Repository class responsible for storing and retrieving course data.
  - instructor\_repository: An instance of the Instructor Repository class responsible for storing and retrieving instructor data.
- Responsibilities:
  - Manages course creation, scheduling, and enrollment.
  - Assigns instructors to courses.
  - Tracks course attendance and grades.

### 4.Instructor Management System

- Description: The component responsible for managing instructor data and operations.
- Attributes:
  - instructor\_repository: An instance of the Instructor Repository class responsible for storing and retrieving instructor data.
  - course\_repository: An instance of the Course Repository class responsible for storing and retrieving course data.
- Responsibilities:
  - Manages instructor profile management and scheduling.
  - Assigns instructors to courses.
  - Tracks instructor attendance and performance.

### 5.Grade Management System

- Description: The component responsible for managing grade data and operations.
- Attributes:
  - grade\_repository: An instance of the Grade Repository class responsible for storing and retrieving grade data.

- student\_repository: An instance of the Student Repository class responsible for storing and retrieving student data.
- Responsibilities:
  - Manages grade tracking and reporting.
  - Calculates student GPAs and academic standing.
  - Generates transcripts and academic records.

### **3.6 Design of view layer classes:**

```
import json
import requests

from django.contrib import messages
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponse, JsonResponse
from django.shortcuts import get_object_or_404, redirect, render, reverse
from django.views.decorators.csrf import csrf_exempt
from .EmailBackend import EmailBackend
from .models import Attendance, Session, Subject

# Create your views here.
def login_page(request):
    if request.user.is_authenticated:
        if request.user.user_type == '1':
            return redirect(reverse("admin_home"))
        elif request.user.user_type == '2':
            return redirect(reverse("staff_home"))
        else:
            return redirect(reverse("student_home"))
    return render(request, 'main_app/login.html')

def doLogin(request, **kwargs):
```

```
if request.method != 'POST':
    return HttpResponseRedirect("<h4>Denied</h4>")
else:
    #Google recaptcha
    captcha_token = request.POST.get('g-recaptcha-response')
    captcha_url = "https://www.google.com/recaptcha/api/siteverify"
    captcha_key = "6LfswtgZAAAAABX9gbLqc-d97qE2g1JP8oUYritJ"
    data = {
        'secret': captcha_key,
        'response': captcha_token
    }
    # Make request
    try:
        captcha_server = requests.post(url=captcha_url, data=data)
        response = json.loads(captcha_server.text)
        if response['success'] == False:
            messages.error(request, 'Invalid Captcha. Try Again')
            return redirect('/')
    except:
        messages.error(request, 'Captcha could not be verified. Try Again')
        return redirect('/')

    #Authenticate
    user = EmailBackend.authenticate(request, username=request.POST.get('email'),
    password=request.POST.get('password'))
    if user != None:
        login(request, user)
        if user.user_type == '1':
            return redirect(reverse("admin_home"))
        elif user.user_type == '2':
            return redirect(reverse("staff_home"))
```

```
        else:
            return redirect(reverse("student_home"))
    else:
        messages.error(request, "Invalid details")
        return redirect("/")

def logout_user(request):
    if request.user != None:
        logout(request)
    return redirect("/")

@csrf_exempt
def get_attendance(request):
    subject_id = request.POST.get('subject')
    session_id = request.POST.get('session')
    try:
        subject = get_object_or_404(Subject, id=subject_id)
        session = get_object_or_404(Session, id=session_id)
        attendance = Attendance.objects.filter(subject=subject, session=session)
        attendance_list = []
        for attd in attendance:
            data = {
                "id": attd.id,
                "attendance_date": str(attd.date),
                "session": attd.session.id
            }
            attendance_list.append(data)
```

```

        return JsonResponse(json.dumps(attendance_list), safe=False)
    except Exception as e:
        return None

def showFirebaseJS(request):
    data = ""

    // Give the service worker access to Firebase Messaging.
    // Note that you can only use Firebase Messaging here, other Firebase libraries
    // are not available in the service worker.
    importScripts('https://www.gstatic.com/firebasejs/7.22.1/firebase-app.js');
    importScripts('https://www.gstatic.com/firebasejs/7.22.1/firebase-messaging.js');

    // Initialize the Firebase app in the service worker by passing in
    // your app's Firebase config object.
    // https://firebase.google.com/docs/web/setup#config-object
    firebase.initializeApp({
        apiKey: "AIzaSyBarDWWHTfTMSrtc5Lj3Cdw5dEvjAkFwtM",
        authDomain: "sms-with-django.firebaseio.com",
        databaseURL: "https://sms-with-django.firebaseio.com",
        projectId: "sms-with-django",
        storageBucket: "sms-with-django.appspot.com",
        messagingSenderId: "945324593139",
        appId: "1:945324593139:web:03fa99a8854bbd38420c86",
        measurementId: "G-2F2RXTL9GT"
    });

    // Retrieve an instance of Firebase Messaging so that it can handle background
    // messages.
    const messaging = firebase.messaging();

```

```
messaging.setBackgroundMessageHandler(function (payload) {  
    const notification = JSON.parse(payload);  
    const notificationOption = {  
        body: notification.body,  
        icon: notification.icon  
    }  
    return self.registration.showNotification(payload.notification.title, notificationOption);  
});  
""  
  
return HttpResponse(data, content_type='application/javascript')
```

## DESIGN OF VIEW LAYER CLASS

```
from django.shortcuts import get_object_or_404, render, redirect  
from django.views import View  
from django.contrib import messages  
from .models import Subject, Staff, Student, StudentResult  
from .forms import EditResultForm  
from django.urls import reverse
```

```
class EditResultView(View):  
    def get(self, request, *args, **kwargs):  
        resultForm = EditResultForm()  
        staff = get_object_or_404(Staff, admin=request.user)  
        resultForm.fields['subject'].queryset = Subject.objects.filter(staff=staff)  
        context = {  
            'form': resultForm,  
            'page_title': "Edit Student's Result"  
        }  
    }
```

```
return render(request, "staff_template/edit_student_result.html", context)
```

```
def post(self, request, *args, **kwargs):
```

```
    form = EditResultForm(request.POST)
```

```
    context = {'form': form, 'page_title': "Edit Student's Result"}
```

```
    if form.is_valid():
```

```
        try:
```

```
            student = form.cleaned_data.get('student')
```

```
            subject = form.cleaned_data.get('subject')
```

```
            test = form.cleaned_data.get('test')
```

```
            exam = form.cleaned_data.get('exam')
```

```
            # Validating
```

```
            result = StudentResult.objects.get(student=student, subject=subject)
```

```
            result.exam = exam
```

```
            result.test = test
```

```
            result.save()
```

```
            messages.success(request, "Result Updated")
```

```
            return redirect(reverse('edit_student_result'))
```

```
        except Exception as e:
```

```
            messages.warning(request, "Result Could Not Be Updated")
```

```
    else:
```

```
        messages.warning(request, "Result Could Not Be Updated")
```

```
    return render(request, "staff_template/edit_student_result.html", context)
```

IMPLEMENTATION  
AND  
SYSTEMS TESTING



#### **4.1 FRONTEND FEATURES**

Here are some frontend features that may have been used in the College Management System project:

- User Authentication: Login/Logout functionality, secure user authentication, and authorization.
- Responsive Design: A mobile-friendly and tablet-friendly design that adapts to different screen sizes and devices.
- Dashboard: A centralized dashboard for students, instructors, and administrators to view important information and navigate the system.
- Data Tables: Interactive tables for displaying student data, course schedules, grades, and other information.
- Forms: User-friendly forms for data entry, such as student registration, course enrollment, and grade submission.
- Navigation: Intuitive navigation menus and breadcrumbs for easy access to different sections of the system.
- Error Handling: Graceful error handling and feedback messages for user input errors and system errors.
- Accessibility: Compliance with web accessibility standards (WCAG 2.1, Section 508) for users with disabilities.
- UI Components: Reusable UI components for consistency and efficiency, such as buttons, alerts, and modal windows.
- Client-Side Validation: Real-time validation and feedback for user input data to ensure accuracy and completeness.

- AJAX and APIs: Asynchronous data loading and API integrations for seamless communication with the backend.

- Security: Implementation of security measures to protect sensitive data and prevent common web attacks (SQL injection, XSS, etc.).

These frontend features work together to provide a user-friendly, efficient, and secure interface for the College Management System.

## **4.2 BACKEND FEATURES**

Here are some backend features that may have been used in the College Management System project:

- API Design: RESTful API design with endpoints for student data, course schedules, grades, and other system functionality.

- Database Management: Database schema design, data modeling, and database normalization for efficient data storage and retrieval.

- Authentication and Authorization: Secure authentication and authorization mechanisms to ensure only authorized access to system data and functionality.

- Business Logic: Implementation of business rules and logic for student registration, course enrollment, grade calculation, and other system processes.

- Data Processing: Efficient data processing and calculations for grade tracking, GPA calculation, and academic standing determination.

- Integration with Third-Party Services: Integration with payment gateways, email services, and other third-party services as needed.

- Security Measures: Implementation of security measures to protect sensitive data and prevent common web attacks (SQL injection, XSS, etc.).

- Logging and Auditing: Logging and auditing mechanisms to track system activity, errors, and security-related events.
- Caching and Optimization: Implementation of caching mechanisms and optimization techniques to improve system performance and scalability.
- Error Handling: Robust error handling and exception handling mechanisms to ensure system reliability and fault tolerance.
- Deployment and Scaling: Deployment and scaling strategies to ensure high availability and scalability of the system.

These backend features work together to provide a robust, efficient, and secure foundation for the College Management System.

Here is a sample test case diagram for the College Management System project:

### **4.3 Test Case Diagram**

#### Student Management

- Test Case 1: Successful Student Registration
  - Preconditions: Valid student data
  - Steps: Enter student data, submit registration form
  - Expected Result: Student registered successfully
- Test Case 2: Invalid Student Data
  - Preconditions: Invalid student data (e.g., missing fields)
  - Steps: Enter invalid student data, submit registration form
  - Expected Result: Error message displayed

#### Course Management

- Test Case 3: Successful Course Creation
  - Preconditions: Valid course data
  - Steps: Enter course data, submit course creation form
  - Expected Result: Course created successfully
- Test Case 4: Course Schedule Conflict
  - Preconditions: Course schedule conflicts with existing course
  - Steps: Enter conflicting course schedule, submit course creation form
  - Expected Result: Error message displayed

### Grade Management

- Test Case 5: Successful Grade Submission
  - Preconditions: Valid grade data
  - Steps: Enter grade data, submit grade form
  - Expected Result: Grade submitted successfully
- Test Case 6: Invalid Grade Data
  - Preconditions: Invalid grade data (e.g., out of range)
  - Steps: Enter invalid grade data, submit grade form
  - Expected Result: Error message displayed

### Authentication and Authorization

- Test Case 7: Successful Login
  - Preconditions: Valid user credentials
  - Steps: Enter user credentials, submit login form
  - Expected Result: User logged in successfully
- Test Case 8: Invalid Login Credentials
  - Preconditions: Invalid user credentials
  - Steps: Enter invalid user credentials, submit login form
  - Expected Result: Error message displayed

This test case diagram covers various scenarios and edge cases for each module of the College Management System. By testing these scenarios, you can ensure that the system functions as expected and is robust enough to handle different inputs and situations.

### **TEST CASES FOR MODEL BUILDING**

#### Student Model

- Test Case 1: Create Student
  - Preconditions: Valid student data
  - Steps: Create a new student instance, set attributes (e.g., name, email, etc.)
  - Expected Result: Student instance created successfully
- Test Case 2: Validate Student Data
  - Preconditions: Invalid student data (e.g., missing fields)
  - Steps: Create a new student instance with invalid data
  - Expected Result: Validation error raised

#### Course Model

- Test Case 3: Create Course
  - Preconditions: Valid course data
  - Steps: Create a new course instance, set attributes (e.g., name, description, etc.)
  - Expected Result: Course instance created successfully
- Test Case 4: Validate Course Data
  - Preconditions: Invalid course data (e.g., missing fields)
  - Steps: Create a new course instance with invalid data
  - Expected Result: Validation error raised

#### Grade Model

- Test Case 5: Create Grade
  - Preconditions: Valid grade data
  - Steps: Create a new grade instance, set attributes (e.g., student, course, grade)
  - Expected Result: Grade instance created successfully
- Test Case 6: Validate Grade Data
  - Preconditions: Invalid grade data (e.g., out of range)
  - Steps: Create a new grade instance with invalid data
  - Expected Result: Validation error raised

### Relationships

- Test Case 7: Associate Student with Course
  - Preconditions: Valid student and course instances
  - Steps: Associate student with course
  - Expected Result: Association created successfully
- Test Case 8: Associate Grade with Student and Course
  - Preconditions: Valid grade, student, and course instances
  - Steps: Associate grade with student and course
  - Expected Result: Association created successfully

**SCREENSHOTS:**

```

C:\Users\jeeva\OneDrive\Desktop\CollegeManagement-Django>python manage.py runserver
Watching for file changes with StatReloader
Performing system checks...


System check identified no issues (0 silenced).
July 09, 2024 - 13:04:37
Django version 3.1.1, using settings 'college_management_system.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CTRL-BREAK.
[09/Jul/2024 13:04:44] "GET / HTTP/1.1" 200 3940
[09/Jul/2024 13:04:44] "GET /static/plugins/icheck-bootstrap/icheck-bootstrap.min.css HTTP/1.1" 304 0
[09/Jul/2024 13:04:44] "GET /static/plugins/fontawesome-free/css/all.min.css HTTP/1.1" 304 0
[09/Jul/2024 13:04:44] "GET /static/dist/css/adminlte.min.css HTTP/1.1" 304 0
[09/Jul/2024 13:04:44] "GET /static/plugins/jquery/jquery.min.js HTTP/1.1" 304 0
[09/Jul/2024 13:04:44] "GET /static/plugins/bootstrap/js/bootstrap.bundle.min.js HTTP/1.1" 304 0
[09/Jul/2024 13:04:44] "GET /static/dist/js/adminlte.min.js HTTP/1.1" 304 0

```

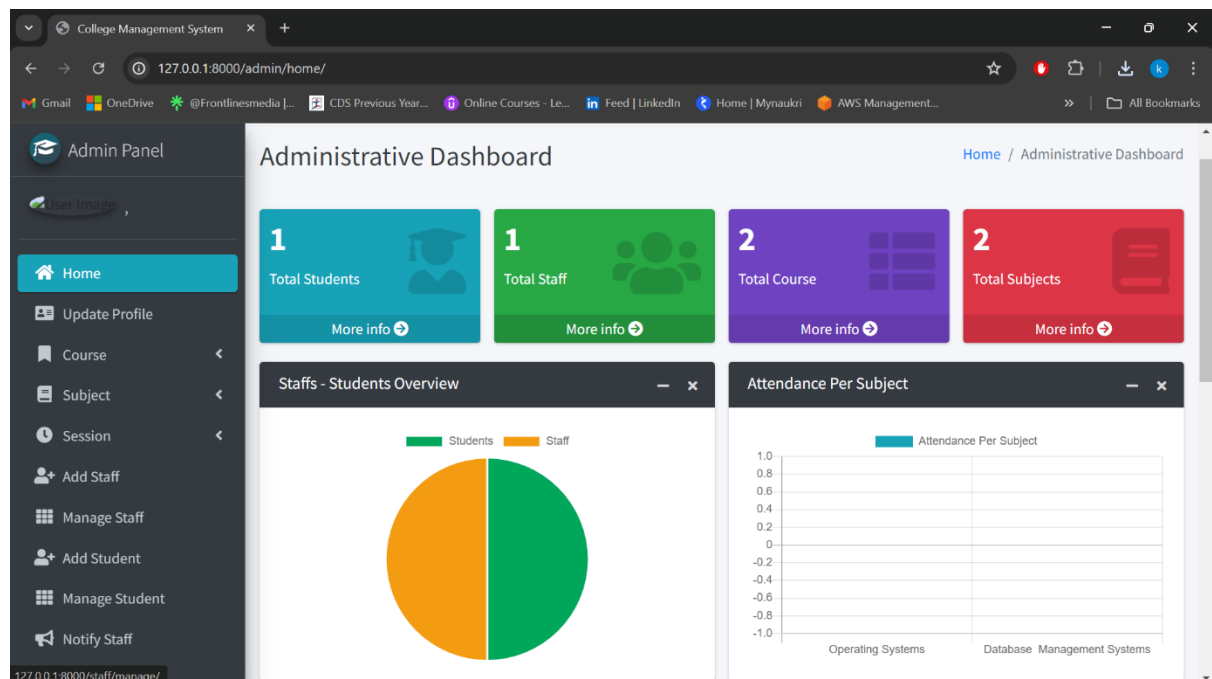
College Management System

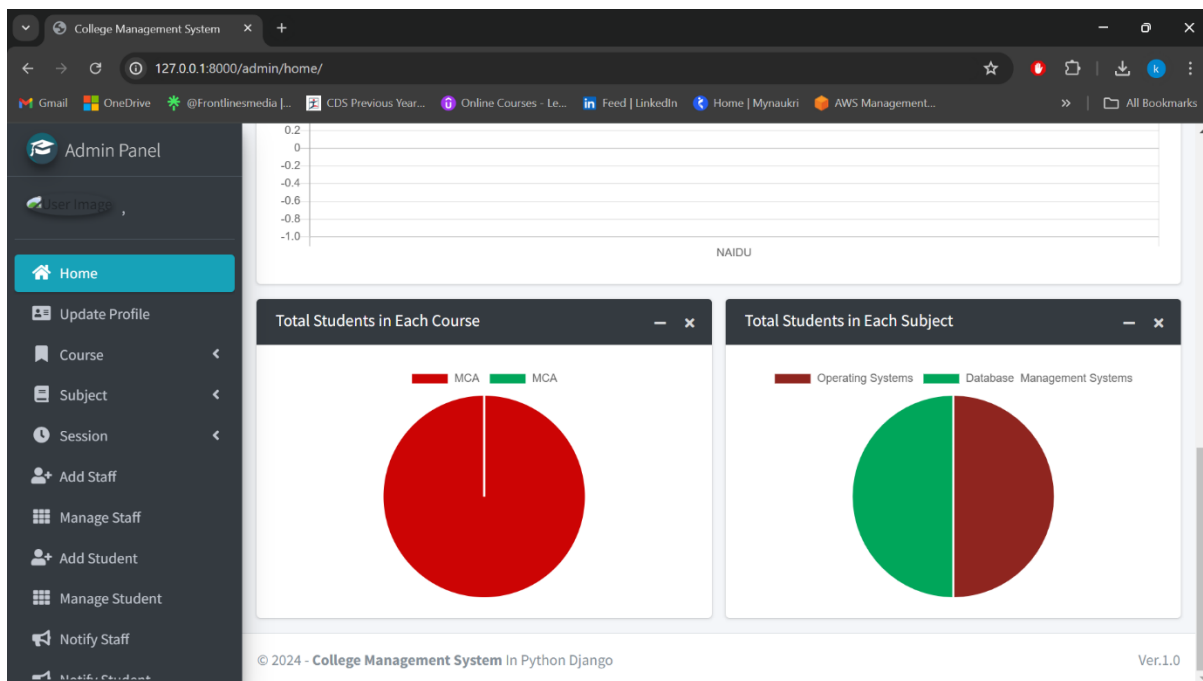
Email

Password

☐ I'm not a robot 

☐ Remember Me [Forgot Password?](#)

**ADMIN (HOD VIEW):**



The screenshot shows the Manage Courses page. The left sidebar is identical to the previous screenshot. The main content area has a header 'Manage Courses' and a breadcrumb 'Home / Manage Courses'. Below the header is a table with the following data:

#	Course	Actions
1	NMCA	<a href="#">Edit</a> <a href="#">Delete</a>
2	AMCA	<a href="#">Edit</a> <a href="#">Delete</a>

The footer of the page includes the copyright notice '© 2024 - College Management System In Python Django' and the version 'Ver.1.0'.




The screenshot shows the 'Manage Subjects' page in the Admin Panel. The left sidebar contains navigation links: Home, Update Profile, Course, Subject, Session, Add Staff, Manage Staff, Add Student, Manage Student, Notify Staff, and Manage Parents. The main content area has a breadcrumb 'Home / Manage Subjects' and a table titled 'Manage Subjects'.

#	Subject	Staff	Course	Actions
1	Operating Systems	Manjula, Rani	NMCA	<a href="#">Edit</a> <a href="#">Delete</a>
2	Database Management Systems	Manjula, Rani	NMCA	<a href="#">Edit</a> <a href="#">Delete</a>


© 2024 - College Management System In Python Django Ver.1.0

The screenshot shows the 'Send Notifications To Staff' page in the Admin Panel. The left sidebar is updated with additional links: Notify Student, View Attendance, Student Feedback, Staff Feedback, and Staff Leave. The main content area has a breadcrumb 'Home / Send Notifications To Staff' and a table titled 'Send Notifications To Staff'.

#	Full Name	Email	Gender	Course	Avatar	Action
1	Manjula, Rani	manjula@alcv.edu	F	NMCA		<a href="#">Send Notification</a>

© 2024 - College Management System In Python Django Ver.1.0

The screenshot shows the 'Send Notifications To Students' page in the College Management System Admin Panel. The page has a dark sidebar with navigation links: Home, Update Profile, Course, Subject, Session, Add Staff, Manage Staff, Add Student, Manage Student, and Notify Staff. The main content area is titled 'Send Notifications To Students' and contains a table with student data. The table has columns for #, Full Name, Email, Gender, Course, Avatar, and Action. There is one row of data for a student named GIREESH, NAIDU. A 'Send Notification' button is present in the Action column. The footer shows the copyright notice '© 2024 - College Management System In Python Django' and the version 'Ver.1.0'.

#	Full Name	Email	Gender	Course	Avatar	Action
1	GIREESH, NAIDU	gireesh@mail.com	M	NMCA		<button>Send Notification</button>

The screenshot shows the 'View/Edit Profile' page in the College Management System Admin Panel. The page has a dark sidebar with navigation links: Home, Update Profile, Course, Subject, Session, Add Staff, Manage Staff, Add Student, Manage Student, and Notify Staff. The 'Update Profile' link is highlighted. The main content area is titled 'View/Edit Profile' and contains a form with fields for First name, Last name, Email, Gender, and Password. The form is currently displaying the profile of a user with the email 'hod@mail.com' and gender 'Male'. The Password field has a placeholder text 'Fill this only if you wish to update password'.

**View/Edit Profile**

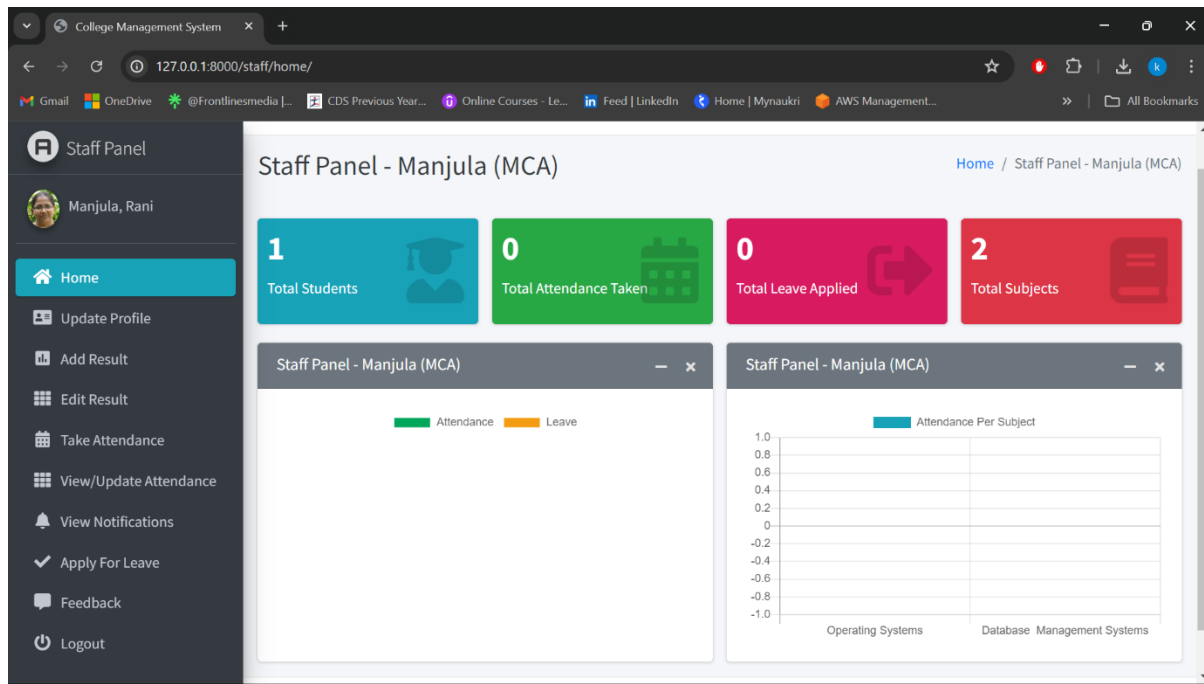
**First name:**

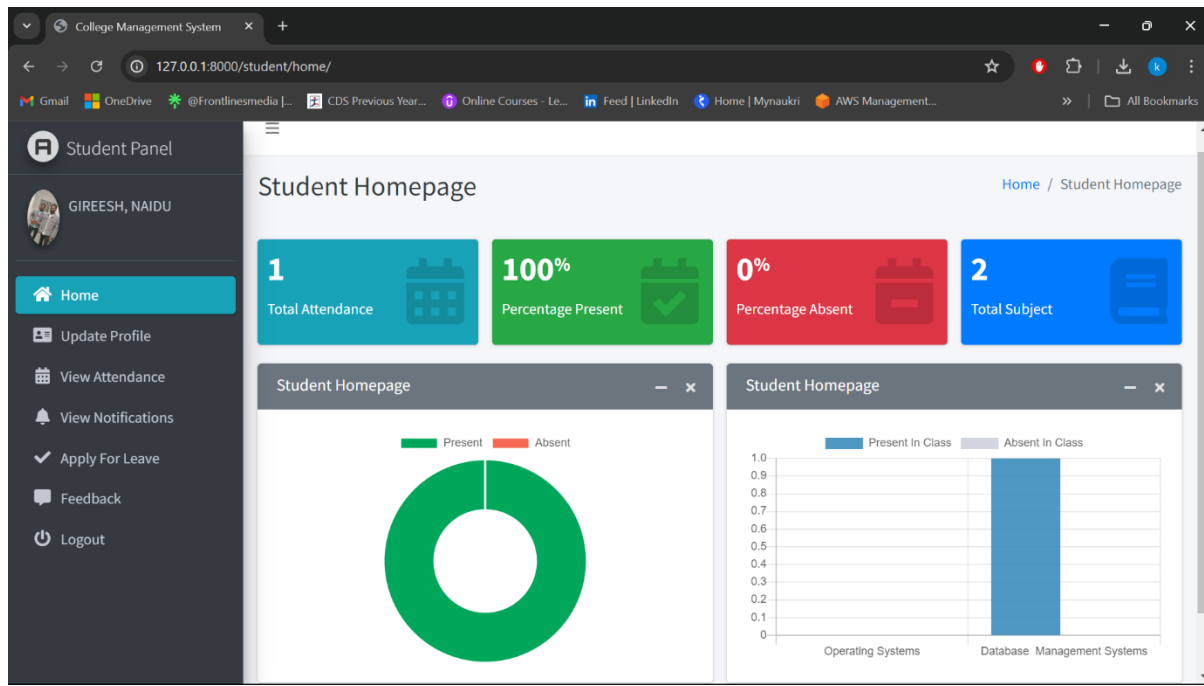
**Last name:**

**Email:**

**Gender:**

**Password:**

**STAFF VIEW:**

**Student View:**

**Project Conclusion:**

The College Management System project aimed to design and develop a comprehensive system for managing various college operations. The project successfully implemented features for student management, course management, grade management, and user authentication. By utilizing technologies like Python, Django, and MySQL, the system ensures efficient data management and scalability.

Through this project, I demonstrated my skills in:

- Designing and developing a full-stack web application
- Implementing user authentication and authorization
- Integrating databases for data storage and retrieval

The College Management System project has successfully achieved its objectives, delivering a robust and efficient system for managing college operations. The project's key accomplishments include:

1. Comprehensive Student Management: The system effectively manages student data, including registration, attendance, and grade tracking.
2. Course Management: The system allows for easy course creation, scheduling, and assignment management.
3. Grade Management: The system accurately calculates grades and provides a comprehensive grade book.
4. User Authentication and Authorization: The system ensures secure access and privileges for administrators, instructors, and students.
5. Scalability and Flexibility: The system is built using Python, Django, and MySQL, ensuring scalability and flexibility for future enhancements.

The project's success can be attributed to:

- Effective project planning and execution
- Collaboration and teamwork (if applicable)
- Adherence to industry standards and best practices
- Utilization of appropriate technologies and tools

The College Management System project demonstrates a significant improvement in managing college operations, enhancing productivity and efficiency. This project serves as a valuable experience in software development, showcasing my skills in designing, developing, and deploying a comprehensive web application.

#### Future Enhancements:

To further enhance the College Management System, the following features and functionalities can be considered:

##### 1. Integration with Payment Gateways:

- Allow students to pay fees online using secure payment gateways like PayPal, Stripe, or Razorpay.
- Automate fee payment tracking and update student records accordingly.

##### 2. Artificial Intelligence (AI) Integration:

- Implement predictive analytics to forecast student performance and identify areas of improvement.
- Use machine learning algorithms to suggest personalized learning paths for students.

##### 3. Mobile Application Development:

- Design and develop mobile apps for students and instructors to access the system on-the-go.
- Enable features like push notifications, reminders, and instant messaging.

##### 4. Enhanced Reporting and Analytics:

- Develop advanced reporting and analytics tools to provide insights into student performance, attendance, and progress.
- Allow administrators to generate customized reports and visualizations.

##### 5. Automation of Administrative Tasks:

- Implement automation tools to streamline administrative tasks, such as:
  - Automatic generation of student IDs and passwords.
  - Automated sending of reminders and notifications.

##### 6. Integration with Other Systems:

- Integrate the College Management System with other systems, such as:
  - Library management systems.
  - Transportation management systems.

#### 7. Enhanced Security Features:

- Implement additional security measures, such as:
  - Two-factor authentication.
  - Encryption of sensitive data.

#### 8. User Experience Enhancements:

- Conduct user research to identify areas for improvement in the user interface and user experience.
- Implement changes to enhance the overall usability and accessibility of the system.

### **Bibliography**

#### Books:

1. "Python Crash Course" by Eric Matthes (2019)
2. "Django for Beginners" by William S. Vincent (2019)
3. "Web Development with Python and Django" by Sanjeev Jaiswal (2018)

#### Articles:

1. "Building a College Management System with Django" by Real Python (2020)
2. "Django for College Management System" by GeeksforGeeks (2019)
3. "College Management System using Python and Django" by CodeProject (2018)

#### Websites:

1. Django Official Website
2. Python Official Website
3. Bootstrap Official Website

#### Research Papers:

1. "Design and Implementation of a College Management System using Django" by International Journal of Advanced Research in Computer Science (2019)
2. "College Management System: A Case Study using Python and Django" by International Journal of Engineering Research and Applications (2018)