MINI PROJECT REPORT ON

COLLEGE MANAGEMENT SYSTEM

Submitted in Partial Fulfillment of the Requirements for

The Award of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE & ENGINEERING

BY

B. RAKESH	**(22C31A0523)**
A. NAVYA	**(22C31A0503)**
A. SAI VARDHAN	**(22C31A0504)**
G.KARTHIK	**(22C31A0553)**
A. ABHINAYA	**(22C31A0514)**

UNDER THE GUIDANCE OF

Dr. RAZIYA BEGUM

Associate Professor, Department of CSE

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BALAJI INSTITUTE OF TECHNOLOGY AND SCIENCE

Laknepally, Narsampet, Warangal (Rural)-506331, Telangana State, India

(Autonomous)

Accredited by NBA (UG-CE, ECE, ME, CSE, EEE Programs) & NAAC A+ Grade (Affiliated by JNTU Hyderabad and Approved by the AICTE, New Delhi) NARSAMP ET, WARANGAL - 506331.

BALAJI INSTITUTE OF TECHNOLOGY AND SCIENCE

Laknepally, Narsampet, Warangal (Rural)-506331, Telangana State, India

(Autonomous)

Accredited by NBA (UG-CE, ECE, ME, CSE, EEE Programs) & NAAC A+ Grade (Affiliated by JNTU Hyderabad and Approved by the AICTE, New Delhi) NARSAMP ET, WARANGAL - 506331.

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **B. RAKESH (22C31A0523), A. NAVYA (22C31A0503), A. SAI VARDHAN (22C31A0504), G. KARTHIK (22C31A0553), A. ABHINAYA (22C31A0514)** of B. Tech **III– II** Sem has satisfactorily completed the Mini Project entitled " **COLLEGE MANAGEMENT SYSTEM Using Django Framework Python** ", as part of curriculum in Computer Science and Engineering during academic year 2024 – 2025.

Internal guide:

Dr. RAZIYA BEGUM

Associate Professor, CSE

HOD

Dr. BANDI KRISHNA

Associate Professor, CSE

Project Coordinator

Mr. J. CHAITANYA

Assistant Professor, CSE

External Examiner

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our guide, **Mrs. RAZIYA BEGUM** whose knowledge and guidance gave motivated us to achieve goals. We never thought possible. She has consistently been a source of motivation, encouragement and inspiration. The time we have spent working under her supervision has truly been a pleasure.

We thank our HoD, **Dr. BANDI KRISHNA** of his effort and guidance and all senior faculty members for their help during our course. Thanks to programmers and non-teaching staff of **CSE** Department of our college.

We heartily thank to our Principal **Dr. V. S. HARIHARAN** for giving this great opportunity and his support to complete our project.

We would to appreciate the guidance given by project coordinator **Mr. J. CHAITANYA** as well as the panels especially in our project presentation that has improved our presentation skills by their comment and tips.

Finally Special thanks to our parents for their support and encouragement throughout our life and this course. Thanks to all my friends and well-wishers for their constant support.

B. RAKESH	**22C31A0523**
A. NAVYA	**22C31A0503**
A.SAI VARDHAN	**22C31A0504**
G. KARTHIK	**22C31A0553**
A.ABHINAYA	**22C31A0514**

ABSTRACT

The College Management System is a web-based application developed using the Django framework. It is designed to streamline and automate various administrative and academic processes within a college environment. The system offers essential features such as student and staff information management, automated attendance tracking, leave application handling, and real-time notifications. These features collectively reduce the need for manual work and contribute to a more efficient and organized management process.

The proposed system is structured into three main panels: Admin, Staff, and Student, each with specific functionalities tailored to the respective users. The Admin panel provides full control over the system. Administrators can add, update, and manage student and staff profiles, create and assign courses and subjects, and monitor the academic performance and attendance of all students. Admins also have the authority to handle leave applications, respond to feedback, and send notifications to both staff and students. This centralized control ensures smooth functioning and effective supervision of college activities.

The Staff panel is primarily designed for faculty members and teachers. Staff can take and manage student attendance digitally and apply for leave through the system. They also receive important updates and notifications from the admin, allowing them to stay informed. The staff panel helps minimize paperwork, saves time, and ensures accurate record-keeping.
The Student panel allows students to access their personal and academic information. They can view and update their profiles, check their attendance status, view exam results, and apply for leave when necessary. Students also receive notifications about academic deadlines, events, and announcements, keeping them actively engaged with college updates.

Overall, the College Management System provides a user-friendly, centralized, and efficient platform that improves communication and coordination among students, staff, and administrators. By automating daily operations and maintaining accurate records, the system supports better time management, reduces manual effort, and contributes to a more productive academic environment.

TABLE OF CONTENTS

1. INTRODUCTION

The aim of the College Management System is to develop a user-friendly and efficient web-based platform using Django to streamline the management of student and staff-related activities in an educational institution. The system seeks to enhance administrative efficiency by providing an integrated solution for handling student and staff information, attendance tracking, leave applications, and notifications. By dividing the system into three panels—Admin, Staff, and Student—it ensures smooth coordination and role-specific functionalities. The Admin panel allows complete control over student and staff records, course and subject management, and handling of leave applications and feedback. The Staff panel enables teachers to manage attendance and apply for leave, while the Student panel allows students to view profiles, track attendance, apply for leave, and receive notifications. By digitizing these processes, the system aims to improve time management, reduce administrative workload, and enhance communication between students, staff, and administrators, ultimately fostering a more organized and efficient academic environment.

1.1 Problem Statement

Overseeing the administrative and academic functions of an educational institution simultaneously is often difficult and time-intensive. Ineffective manual systems of keeping records of students and staff, tracking attendance, processing leave applications, and sending notifications often result in mistakes, inefficiency, and communication barriers. The absence of a central system for retention of student and staff information commonly leads to backlogs in processing leave applications and tracking academic progress. Moreover, manual attendance tracking can also lead to inaccuracies, making records unreliable. To mitigate the issues outlined above, an automated web-based system was designed using Django framework called College Management System, which provides a single, automated interface for all institutional activities. The system automatically tracks attendance and record management, replaces manual leave applications with automated ones, and facilitates instantaneous notifications to enhance students', staff, and administrator's productivity. In addition to improved efficiency, the system also enhances accuracy and reduces communication gaps in the institution.

.

1.2 Objectives

This project aims at creating a web-based application that can efficiently manage student and staff records and administrative functions of educational institutions using Django. The system is designed to have a systematic and friendly layout that allows Admin, Staff and Students to access the system based on their predefined roles. The Administrative Panel guarantees the full scope of the institution's workings by allowing student and staff record and course, subject, leave application and feedback management. The Staff Panel allows teachers to manage student attendance as well as leave applications to ensure that all records are accurate. On the other hand, the student panel allows students to make profile changes, monitor their attendance, make leave applications and receive notifications. The system also aims to improve institutions' time management by automating tracking and managing attendance and communication which reduces chances of error. The College Management System is designed to bring students and staff together with administrators to ensure the entire institution is facilitated in improved coordination and organization academically.

2. SYSTEM ANALYSIS

2.1 Introduction

System analysis is a crucial phase in the software development life cycle where the system is studied thoroughly to understand its components, user requirements, functionalities, and limitations. It involves breaking down the system into smaller parts to analyze how each component interacts and contributes to the overall objective. The main goal of system analysis is to identify what the system is supposed to do and how it can solve the existing problems in an efficient and scalable way.

During this phase, analysts work closely with stakeholders to understand what the system should achieve, what inputs and outputs are needed, and what processes should be included. It involves breaking down complex systems into smaller, manageable modules and analyzing each one to design a better, optimized solution.

System analysis acts as a bridge between the initial project idea and the actual development phase. It ensures that the development team has a clear understanding of user requirements, system environment, and performance expectations, laying the foundation for designing a reliable and scalable system.

2.2 Analysis Model

The **Waterfall Model** is referred to as the base model in software development because it was the first structured approach introduced for managing the software development life cycle (SDLC). It laid the foundation for all subsequent models by clearly defining the concept of phased development, where each stage—such as requirements gathering, design, coding, testing, deployment, and maintenance—follows a logical and sequential order.

This model introduced the idea of producing specific deliverables at the end of each phase, making the process systematic and manageable. It also established the importance of documentation and early planning, which became core principles in many future methodologies. Even though more flexible and iterative models like Agile and Spiral were later developed to address its limitations, they still inherit the basic framework from Waterfall. The fundamental structure of modern SDLC models—clearly defined phases, emphasis on testing, and maintenance—can all be traced back to the

Waterfall approach. Thus, it is considered the base model because it shaped the way software engineering is understood and practiced, forming the core upon which other models have been built.

The Agile Model is a software development methodology that emphasizes iterative progress, collaboration, flexibility, and continuous feedback. Unlike traditional models like the Waterfall Model, Agile focuses on delivering small, functional increments rather than a complete product at once. The project starts with a product backlog, listing key features like user management, workout tracking, nutrition plans, progress monitoring, and appointment booking. Development is divided into sprints (2-4 weeks each), where core features are built, tested, and improved iteratively. Daily stand-ups help track progress, while sprint reviews gather feedback for refinements. Users get frequent updates, ensuring a seamless experience. The Agile approach enables faster delivery, better quality, and easy adaptability for future integrations like AI and wearable devices.

In this model the sequence of activities performed in a software development are:

- **Requirement Gathering & Product Backlog Creation:** In this stage we will identify and document functional and non-functional requirements and organize features into a prioritized **product backlog**.
- **Sprint Planning:** In this stage we will select tasks from the backlog for the current sprint (2-4 weeks). And define clear sprint goals and deliverables.
- **Sprint Execution (Development & Testing):** Develop and integrate selected features incrementally and conduct daily stand-up meetings to track progress and perform continuous testing to ensure software quality.
- **Sprint Review & Feedback Collection:** Demonstrate the developed features to stakeholders. And gather feedback for further refinement.
- **Sprint Retrospective & Continuous Improvement:** Analyze what went well and what can be improved. And adjust the approach for the next sprint.
- **Deployment & Maintenance:** Deploy functional increments to users.Monitor application performance and optimize as needed.

At the end, this iterative cycle repeats, allowing for continuous improvements, adaptability, and user-driven enhancements. By following the Agile model, College Management System aims to deliver a highly adaptable and user-centric college management platform, continuously evolving to meet user needs effectively.

2.3 Modules of the System

The system after careful analysis has been identified to be presented with the following modules:

The modules involved are:

- Administration
- Staff
- Student

Admin Module

- **Update Login and Personal Details:**

Admin can manage their own account details, including login credentials and personal information, for security and record keeping.

- **Staff Registration:**

Admin register/add staff manually by entering their details, including name, mail, address,session, and course. Only registered staff can login and access the staff panel features.

- **Student Registration:**

Admin register/add student manually by entering their details, including name, mail, address, course, session and section. Only registered student can login and access the student panel features.

- **Attendance Monitoring:**

The attendance data will be recorded and it can be viewed by students and as well as staff. Admin can monitor the attendance of the students for any subject taken by any staff.

- **Course Management:**

Admin can add course and also admin can manage the course. The number of courses totally depend on the admin.

- **Session Management:**

Admin can add session and manage session. Session is nothing but from which year the academic year started and when it will be end .

- **Section Management:**

Admin can add sections in a particular course and the number of sections also totally depend on the admin.

- **Subject Management:**

Admin can add any number of subjects and assign the subjects to respective staff.

- **Notify Student/ Staff:**

Admin can send notifications to students and as well as Staff

- **Staff/Student Feedback:**

Admin can view the feedback given from the students and as well as from the staff.

- **Staff Leave:**

When the staff sends a leave notification to the admin, then admin has the right to approve their leave or reject their leave.

Staff Module

- **Update Profile:**

Staff can manage their personal details and update their profile.

- **Attendance Management:**

Staff can record attendance of students of their consultant subject and they can also update the attendance of the students.

- **View Notifications:**

Staff can view the notifications sent from the admin.

- **Apply For Leave:**

Staff can apply for leave and that will go to admin, and admin will approve or reject their leave.

- **Feedback:**

Staff can share their views on improvement of college and send feedback to the admin.

- **Student Leave:**

When the student sends a leave notification to the staff, then staff has the right to approve their leave or reject their leave

Student Module

- **Update Profile:**

Student can manage their personal details and update their profile.

- **Apply For Leave:**

Student can apply for leave, that can be approved by either Staff or admin.

- **Feedback:**

Students can send feedback to admin regarding staff, academics or any college related issue to admin.

- **Dashboard and User Interface:**

The system uses a clean, user-friendly dashboard for smooth user experience. It conatains a responsive user interface.

2.4 Existing System

The existing system for college management typically relies on manual processes or basic software solutions. Administrators often manage student registrations, fee calculations through paper records or spreadsheets. This approach can lead to inefficiencies. Due to manual work there is lots of time waste. There will be maintenance problem due to this and the data may be lost as it is saved in files. Further there will be lack of centralized digital platform and this will limits the ability of both students, faculty, and administration to efficiently manage the college. Overall it will lead to increased burden for administration, faculty and as well as students and a poor user

experience. Overall the existing system is cumbersome and lacks automation and does not meet the modern needs.

Disadvantages

- Manual Student Attendance
- Lots of Time taking process
- Maintenance Problem
- Absence of Section selection
- No role-based access
- No unique id for students

2.5 Proposed System

The College Management System is a web-based application designed with Django Framework aimed to address the shortcomings presented by the manual system in educational institutions. This system automates some of the administrative activities which lessen the burden on the administration without compromising effective and efficient operation. Its design is simple to enable easy access to all functions provided by the college. It is divided into three main parts Admin, Staff, and Student for ease of use and confidentiality of information.

Through Admin panel, the administrators has full power to the system, allowing to manage student and staff records, attendance, allocation of subjects and classes, reporting leaves, notifying and feedback, within the institution facilitating communication effectively. The Staff panel is designed for teachers to manage students' attendance, post and update attendance marks, assess the students academically, and also report sick leaves for themselves or students. The Student panel enables students to check their attendance records, modify their details, apply for leave, and receive notifications.

This system automates the attendance monitoring together with the other functions that are performed manually like the management of students and staff data. It improves efficiency and eliminates errors in work performed by the institution.

Advantages

- Increased Productivity
- Role-based access
- Better Security & Centralization
- User-Friendly interface
- Economical
- Improved Inter-Connectivity
- Higher Reliability

3. FEASIBILITY STUDY

A feasibility study is a high-level analysis aimed at determining the practicality of the entire project. It helps answer several important questions, such as: What is the problem this system aims to solve? Are there feasible solutions to address it? Are there feasible solutions to address it? Is the problem significant enough to need a solution? Feasibility study is conducted once the problem is clearly understood. The primary objective of the feasibility study is to evaluate technical, operational, and economic feasibility, assuming unlimited resources and time. This study identifies potential obstacles and ensures that the project remains viable and useful for the organization.

The following feasibilities are considered for the project in order to ensure that the project is variable and it does not have any major obstructions.

1. **Technical Feasibility**
2. **Operational Feasibility**
3. **Economic Feasibility**

3.1 Technical feasibility

Technical feasibility is one of the first assessments conducted after identifying the project requirements. This analysis helps determine whether the current technical resources and the proposed technological solutions can meet the needs of the project. The technical feasibility study

assesses whether the technology, resources, and skills available can effectively support the development and operation of the project. The technical feasibility study determines if the available technologies, resources, and skills are sufficient to develop and maintain College Management System. This web-based solution to replace manual processes and efficient college management with a secure, scalable, and user-friendly online platform it will handle tasks such as student/staff registration, attendance tracking, and management, Using Django Framework and HTML, CSS, JavaScript and SQLite.

Project requirement and goal alignment

The system is designed to connect staff, student, and admin efficiently. With clearly separated roles and functionalities, the project aligns with its goal of enabling secure, organized, and real-time attendance tracking, student/staff leave management, and admin approval mechanisms.

Scalability

The College Management System is designed with scalability in mind. Future features like real-time notifications, chat between students and staff, or even mobile app integration can be added without major changes to the core system. Django's modular architecture and REST API support make it easy to scale horizontally or vertically as needed

Data Security and Privacy

Since the system handles sensitive user data such as login credentials, and contact information.

1. User Authentication:

Secure login for Staff, Students, and Admins using Django's authentication system.

2. Data Encryption:

Sensitive information is stored securely and encrypted where needed (e.g., passwords using hashing).

3. CSRF & HTTPS Support:

Django provides built-in CSRF protection and HTTPS configurations

4. **Role-Based Access Control:**

Role-based views and permissions restrict sensitive actions.

User friendly Interface

The platform uses a modern responsive design compatible with various screen sizes. Clear navigation, simple UI components, and dashboard-based access improve the experience for staff, students and admin.

Compatibility with Existing Systems

The system is web-based platform built using the Django framework. It can be accessed from any device with a browser (mobile, tablet, or PC), and works smoothly on:

- Local servers (within colleges or offices),
- Public hosting platforms (like Render, PythonAnywhere)
- It does not require internet if hosted on a local network, and it integrates easily into existing college or office infrastructure without needing high-end hardware.
- Django and SQLite/PostgreSQL require minimal server resources, so the system runs efficiently on standard hosting setups — both offline and online.

3.2 Operational feasibility

Operational feasibility assesses how well the proposed system fits into the existing environment and whether it will be accepted and efficiently used by students, staff, and administrators. It determines if the system will operate successfully once developed and deployed, focusing on usability, support, and overall user satisfaction.

The College Management System is designed to replace traditional, manual attendance tracking processes with a seamless digital platform. The system provides a user-friendly interface with role-based access, ensuring that Admins, Staff, and Students can easily navigate and perform their respective tasks with minimal training. By automating attendance tracking, leave applications, and notifications, the system significantly reduces errors and saves time for both staff and students.

Key Considerations include:

User Acceptance:

The system provides a user-friendly interface with intuitive navigation, making it easy for Admin, Student, Staff and access their respective dashboards.

Improved Efficiency:

Automates attendance tracking, leave applications, and notifications, significantly saving time.

Support and Maintenance:

Built with Django, the system is easy to maintain. Regular updates, bug fixes, and feature additions can be managed smoothly without affecting operations.

3.3 Economic feasibility

Economic feasibility evaluates the cost-effectiveness of the proposed system by analyzing the financial benefits it offers compared to the investment required for its development, deployment, and maintenance. It helps determine whether the project is a financially viable solution for the organization.

College Management System project is economically feasible, as it involves minimal cost and offers significant benefits. Since the platform is developed using open-source technologies like Django, Python, SQLite, HTML, CSS, and JavaScript, there are no licensing costs. The system is designed to run on basic infrastructure, making it highly affordable for real-world implementation.

Key Considerations include:

Low Development Cost

The system is developed using open-source technologies such as Python, Django, SQLite, HTML, CSS, and JavaScript, which eliminate the need for costly licenses or subscriptions. Most development tools and libraries used are free to use.

Minimal Hardware Requirements

The platform can be hosted on low-cost cloud platforms (like PythonAnywhere, Render) or even on local servers within institutions, making it affordable to deploy without expensive hardware.

Reduction in Manual Costs

Digitizing the college management process eliminates the need for paperwork, physical storage, and manual verification, leading to long-term cost savings.

Reusable and Scalable

The codebase is modular and scalable, which means the same system can be reused or expanded in future projects (like adding online payments or a mobile app) without incurring high redevelopment costs.

4. SOFTWARE REQUIREMENT SPECIFICATION

The Software Requirement Specification (SRS) for the College Management System outlines the detailed technical and functional requirements essential for the successful development and deployment of the platform. This document defines the system's behavior, features, performance expectations, and interfaces, serving as a reference point for developers, testers, project stakeholders, and maintenance teams.

The College Management System is a web-based application built using Django, designed to efficiently manage student and staff records along with various administrative functions. The system follows a role-based access model, where Admins, Staff, and Students have predefined permissions. The functional requirements include student and staff record management, attendance tracking, leave applications, course and subject management and a feedback system. The front-end will be built using Django templates or React with HTML, CSS, and JavaScript, ensuring a user-friendly and responsive design. Security measures such as CSRF protection, role-based authentication, and data encryption will be implemented to safeguard user data. The application will support email and in-system notifications for better communication.

This document ensures that all functional, technical, and operational requirements are captured and agreed upon, thereby minimizing ambiguity during development. It also supports change management, wherein any modifications to the specifications must undergo formal review and approval to preserve system integrity and user alignment.

4.1 Functional Requirements

The functional requirements define the specific behavior and functions that system must perform. The College Management System must provide to its users. These requirements ensure that the platform meets its intended purpose by enabling secure, efficient, and user-friendly interactions for staff, student, and admins.

1. User Management

Role-Based Authentication – Users (Admin, Staff, and Students) must log in with unique credentials.

User Registration & Profile Management – Admin can create and manage student and staff accounts, while users can update their profiles.

Role-Based Access Control – Each user type (Admin, Staff, Student) will have specific access permissions.

2. Student Management

Student Record Management – Admin can add, update, or delete student details such as name, department, and contact information.

Course & Subject Enrollment – Students can enroll in courses, and Admin can assign subjects to them.

Attendance Tracking – Students can view their attendance records, while Staff can mark attendance daily.

3. Staff Management

Teacher Assignment to Subjects – Admin can assign teachers to specific subjects and classes.

Staff Attendance Management – Staff members can mark their own attendance and apply for leave.

4. Attendance Management

Automated Attendance Tracking – Staff can mark student attendance for each class.

Attendance Reports & Analytics – Admin can generate attendance reports and track student participation.

5. Leave Management

Leave Application System – Staff and students can apply for leave through the system.

Leave Approval & Status Tracking – Admin has the authority to approve or reject leave requests, and users receive notifications.

6. Result & Academic Performance Management

Marks & Grade Entry – Staff can enter students' marks for subjects and generate results.

Result Viewing & Analysis – Students can access their results, and Admin can generate academic performance reports.

7. Notification & Communication System

System Notifications – Users receive alerts for attendance, leave approval, results, and other updates.

Admin Announcements – Admin can send important notices to Staff and Students.

8. Feedback & Complaint System

Student & Staff Feedback Submission – Users can submit feedback or complaints.

Admin Review & Response – Admin can view and respond to submitted feedback.

9. Security & Access Control

User Authentication & Data Protection – Secure login system using Django's authentication framework.

Role-Based Permissions – Restrict access to sensitive data based on user roles.

10. System Scalability & Future Enhancements

API Integration for Mobile Apps – The system should support API-based expansion for mobile compatibility.

Online Payment Gateway (Future Scope) – Support for fee payments and transactions.

Chat System (Future Scope) – Internal messaging for communication between students, staff, and administration.

4.2 Non-Functional Requirements

The non-functional requirements define the system's operational characteristics, such as performance, security, usability, and maintainability. These requirements ensure that College Management System meets quality standards, remains reliable, and delivers a seamless and user-friendly experience for all users, including students, staff, and administrators.

1. Security

The system implements secure user authentication and role-based access control. It Ensures basic web security using Django's built-in features. The platform includes **Cross-Site Request Forgery (CSRF) protection** to prevent unauthorized actions from external sites. Django's security middleware also helps in managing safe form submissions and session handling. Sensitive user data such as passwords is securely hashed and stored using Django's authentication.

2. Availability

It is maintained by ensuring that the system is consistently accessible to users with minimal downtime. Since it is a web-based platform, the application can be hosted on a reliable server with proper backup mechanisms and error handling to reduce service interruptions.

3. Usability

The platform is designed with a clean and simple user interface using HTML, CSS, and Bootstrap, making it easy to navigate even for users with minimal technical knowledge. Clear layouts, intuitive navigation, and mobile responsiveness ensure a smooth experience for staff, students, and admins alike.

4. Integrity

The system must ensure data accuracy, authenticity, and integrity, preventing unauthorized data modifications and ensuring reliable data storage and retrieval. Ensures all data is validated, accurate, and securely stored.

5. Maintainability

The codebase is modular and cleanly structured using Django's app-based architecture, making it easier to maintain and update. Future enhancements like adding new features or improving security can be done without affecting existing modules significantly.

6. Portability

The system should run on all major browsers and operating systems.

It should support deployment on local servers or public cloud platforms like Render, Heroku, or PythonAnywhere.

7. Data Backup and Recovery

Regular backups should be scheduled to prevent data loss.The system should support data recovery in case of server failure.

4.3 Performance Requirements

The performance requirements focus on ensuring that College Management System operates efficiently under normal and peak load conditions. The system should respond to user actions like login, attendance, or taking leave within a few seconds. It must handle multiple users accessing the platform simultaneously without crashing or slowing down. Pages should load quickly, notification system must update in real-time to maintain smooth user interaction.

Response Time

The system should respond to user actions within 2–3 seconds under normal network conditions. Pages like the home page, dashboard must load in under 3 seconds.

Concurrent Users

The system should support at least 500+ concurrent users without performance degradation, ensuring smooth operation during peak hours like result announcements.

System Uptime

The system must maintain an uptime of **99% or higher**, ensuring consistent availability for students and staff, admin.

Resource Utilization

The system should use CPU and memory resources efficiently, even under load, especially when hosted on platforms with limited resources like PythonAnywhere or Render.

4.4 Hardware Requirements

College Management is a web-based platform that requires minimal hardware resources for both development and deployment. The system can run efficiently on standard hardware used in most educational institutions or hosting services.

Server Requirements

1.Processor:

A high-speed processor, ideally Intel Core i5 or higher / multi-core. It handles multiple operations, user authentication, and simultaneous requests across different roles (User, Seller, and Admin).

2.RAM:

A minimum of 8 GB RAM is required to support smooth functioning of the Django backend services, database operations, and real-time data processing. For high traffic or scaling, 16 GB RAM is preferable.

3.Storage:

At least 50 GB of SSD (Solid-State Drive) storage is recommended for faster data retrieval, quick loading of media files (like pet images), and better performance of the web application. SSD is preferred over HDD for reduced latency and enhanced speed.

4. Network Connectivity:

A reliable high-speed internet connection (at least 10 Mbps upload/download) is necessary to ensure uninterrupted data transfer, quick page loading, and a smooth user experience during high traffic.

5.Security:

Secure Sockets Layer certificates are needed for secure HTTPS connections, ensuring that user data like login credentials and adoption requests are encrypted during transmission. Firewalls and regular backups are also recommended for data safety.

6. Database Server:

SQLite3 is used as a lightweight, serverless database system, requiring minimal hardware with basic storage and memory. It stores all data in a single .db file, ideal for prototyping but not suitable for large-scale production use

Client Device Requirements

1. Processor: Dual-core processor (e.g., Intel Core i3 or equivalent) or higher for smooth interaction with the system interface.

2. RAM: Minimum 4GB RAM for regular operations to ensure a seamless user experience when accessing the system.

3. Display: A device with a screen resolution of 1024x768 pixels or higher is recommended for proper UI layout and readability.

4. Network Access: A stable internet connection with a minimum speed of 1 Mbps is needed for smooth browsing, form submissions, and pet listing interactions

4.5 Software Requirements

Programming Language:

Python 3.9 or above – Python is used for server-side logic. Django, the chosen web framework, is built in Python and provides fast, secure development with minimal coding.

Python ensures clean syntax, rich library support, and quick implementation of web functionalities.

Frameworks & Libraries:

Django

Django is the primary framework for building the backend of the system. It provides a robust structure for handling HTTP requests, managing user authentication, performing database operations, and serving dynamic content.

Version: Django 5.1.5

Features:

- Model-View-Template (MVT) architecture for easy separation of concerns.
- Built-in authentication and session management.
- Highly scalable and secure framework with robust support for database operations.

Django REST Framework

It is Used to build RESTful APIs, enabling frontend and mobile app integration in future.

Frontend: HTML, CSS, JavaScript and Bootstrap

These technologies are used to develop the user interface (UI) of the platform.

HTML: Used for the structure of web pages, organizing content, form layouts, and navigation elements.

CSS: Enhances the user interface design, ensuring the system is visually appealing and accessible on various devices.

JavaScript for making web pages interactive, handling dynamic content, and providing a better user experience.

Bootstrap for responsive design and UI components, ensuring the website adapts to different screen sizes (mobile, tablet, desktop).

Database: SQLite (Development) / PostgreSQL (Production)

SQLite is used during development, while PostgreSQL will be used in production for scalability.

SQLite: Lightweight, file-based database suitable for local development.

PostgreSQL: A powerful, relational database management system used for handling large datasets and ensuring high performance in production environments.

Operating System:

Windows 10 or above: Most commonly used OS in development environments and supports all required tools.

Tools and IDEs:

Visual Studio Code / PyCharm: Code editors with features like debugging, auto-completion, and Git integration for efficient development.

Postman: API testing tool used to verify endpoints and HTTP request/response flow.

Git & GitHub: Version control system used to track changes and collaborate in teams.

PythonAnywhere / Render: Platforms used for deploying the Django application to a live server environment.

These tools streamline development and deployment workflows.

Additional Tools:

- **Pip**: Python package installer used to install dependencies like Django, DRF, etc.
- **Virtualenv**: Used to create isolated Python environments to manage project-specific packages and avoid conflicts.

5. SOFTWARE DESIGN

5.1 Introduction

System Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It provides a detailed plan that outlines how the system should function and how different parts of the system interact with each other. It acts as a bridge between system requirements and the final implementation. It includes planning the layout of web pages, database structure, user flow, and security features. A well-designed system ensures smooth functionality, scalability, security, and an easy user interface. This stage ensures that each part of the system is well-structured, maintainable, and scalable.

The system design for the College Management System includes planning for various modules such as user authentication, student and staff management, attendance tracking leave management, and notifications. It also involves deciding how data will be stored (using PostgreSQL/MySQL), how users will navigate the platform, and how security and performance will be maintained. The use of Django ensures modular development and separation of concerns, making the system easier to design and maintain. Overall, a well-designed system improves user experience, enhances performance, and prepares the platform for future enhancements.The design process for The Pet Adoption Hub involves four main components:

1.Architectural Design: It defines the overall structure and interaction of key components in the system. the system is built using a Client-Server Architecture with a Three-Tier Model: Frontend, Backend and Database.

2.Data Structure Design: Focuses on how the data will be organized and stored in the system. It ensures that data is structured efficiently, which facilitates easy retrieval and updating. the system will have several key entities, each corresponding to a table in the database such as User table, student table, staff table, course table, subject table, attendance table, leave table, notification table.

3.Interface Design: Defines the user interfaces that enable students, staff, and administrators to interact with the system. This involves creating clear, user-friendly screens and navigation paths for accessing system functionalities: Admin Dashboard, Staff Dashboard, Student Dashboard.

4.Procedural Design: The Procedural Design outlines the operations and workflows the system needs to perform. These operations align with the functional requirements of College Management System. This includes User Authentication, Staff/Student management, Attendance tracking, Leave management and Notification management, admin approval.

6. UML Diagram

Unified Modeling Language (UML) is a standardized modeling language used to visualize, design, and document the structure and behavior of software systems. It provides a set of graphical notations and diagrams that help developers and stakeholders understand how a system is built and how different components interact with each other. UML plays a vital role in the software development life cycle by making the design process clear, structured, and efficient.

Using UML, complex systems can be broken down into understandable visual models. It supports both structural modeling (like Class and Component Diagrams) and behavioral modeling (like Use Case, Sequence, and Activity Diagrams). These diagrams make communication within the development team more efficient, reduce ambiguity, and ensure that everyone shares a common understanding of the system.

For College Management System project, UML diagrams are used to illustrate the system's key components such as staff, student and admin interactions, attendance tracking, and admin controls. These diagrams ensure clarity in system functionality and guide the development process with a well-structured blueprint.

Relationships

In **UML (Unified Modeling Language) diagrams**, **relationships** represent how different elements (usually classes, objects, or components) are **connected or interact with each other**. Understanding these relationships is essential for designing robust object-oriented systems.

1. **Association**

A basic relationship where one class is connected to another, showing that objects of one class use or are aware of objects of another.

Example: A Student is associated with a Course.

Notation: Solid line connecting two classes, Can be one-way or bidirectional.

2. **Aggregation (Has-a Relationship)**

A special type of association that represents a **whole-part** relationship, where the part can exist independently of the whole.

Example: A Team has multiple Players, but a Player can exist without a Team.

Notation: Hollow diamond at the **whole** end (Team ◇── Player).

3. **Composition (Strong Has-a Relationship)**

A **stronger form of aggregation**, where the part **cannot exist independently** of the whole.

Example: A House is composed of Rooms. If the House is destroyed, the Rooms cease to exist.

Notation: Solid (black) diamond at the whole end (House ◆── Room).

4. **Inheritance / Generalization (Is-a Relationship)**

A relationship where one class (child) **inherits** the properties and behaviors of another class (parent).

Example: Dog is a subclass of Animal.

Notation: Solid line with a **hollow triangle** pointing to the parent class (Dog ─▷ Animal).

5. **Realization (Implements Relationship)**

Used when a class **implements an interface**.

Example: Printer class implements Printable interface.

Notation: Dotted line with a hollow triangle pointing to the interface (Printer - -▷ Printable).

6. **Dependency**

A **temporary relationship** where a change in one element **may affect** another, but not vice versa.

Example: A method in class A uses class B as a parameter or local variable.

Notation: Dotted arrow pointing from the dependent to the independent (A - - > B).

USECASE

A Use Case Diagram is a behavioral diagram in the Unified Modeling Language (UML) that represents the interactions between users (actors) and a system. It describes the various ways in

which users can interact with the system to achieve a specific goal. Each use case defines a specific functionality or feature of the system from the user's perspective, helping to capture the system's functional requirements.

In a Usecase diagram:

Actor:

An actor is anyone or anything that interacts with the system from outside, like a user, admin, or seller.

Use Case:

A use case is a specific action or function that the system performs in response to an actor's request.

System Boundary:

The system boundary shows what is inside the system (its features) and separates it from everything outside.

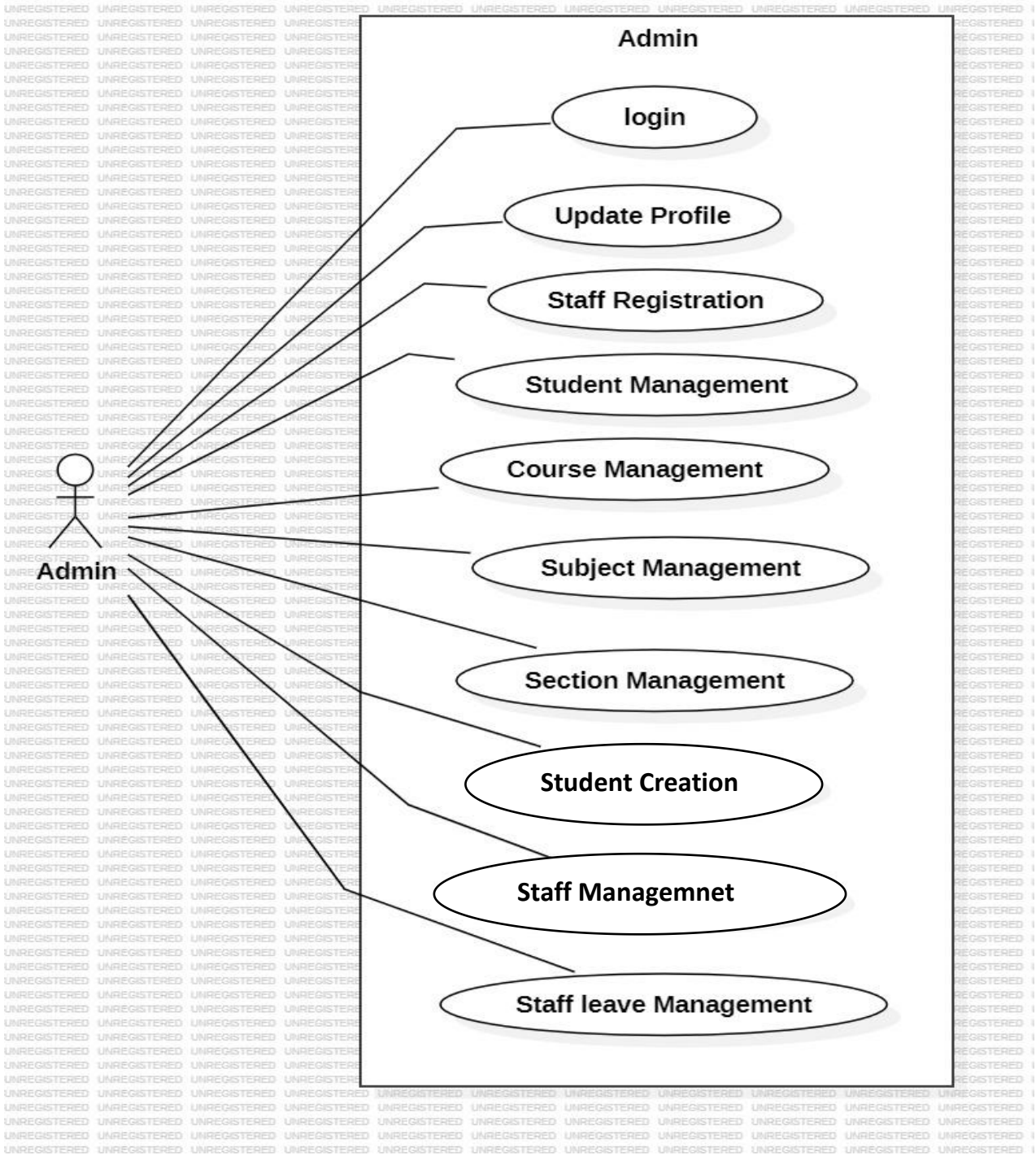

Fig 6.1: Use Case Diagram for admin

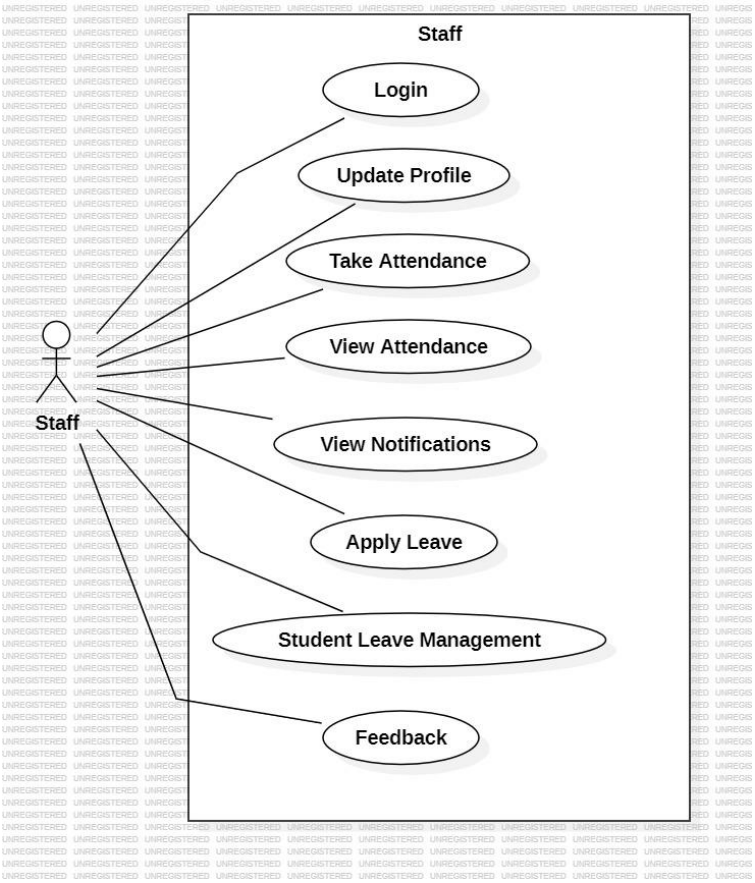

Fig 6.2: Use Case Diagram for Staff

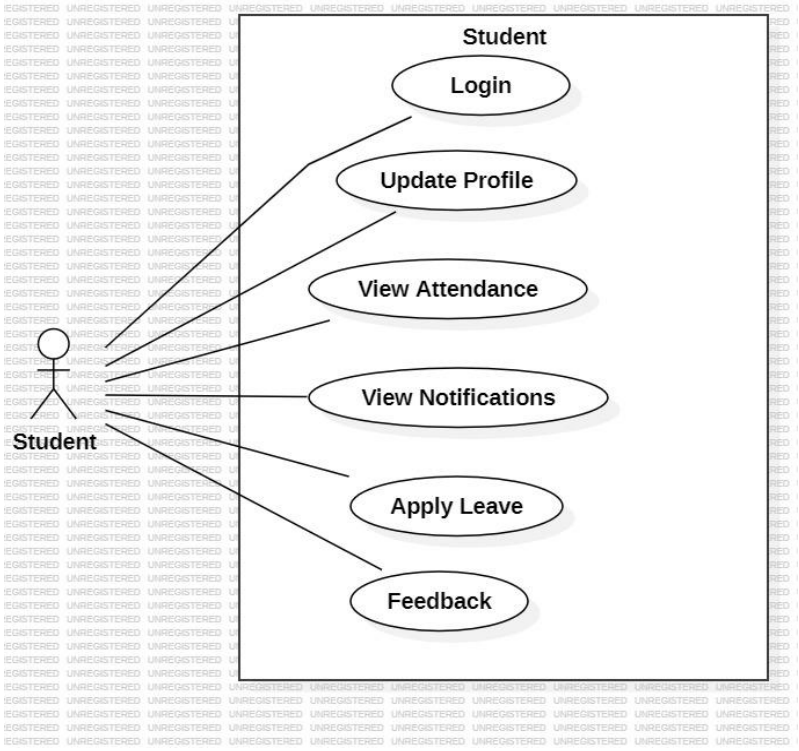

Fig 6.3: Use Case Diagram for Student

Class Diagram

A class diagram is a fundamental building block in object-oriented system design, providing a static view of the system's structure by showing its classes, interfaces, and their relationships. It essentially illustrates how different components of the system interact with one another. Each class in the diagram contains the following compartments:

1.Class Name: The title of the class, which identifies it uniquely within the system.

2.Attributes: The properties or data members associated with each class, describing the information it holds.

3.Methods/Functions: The Behaviours or operations that the class can perform, representing its actions

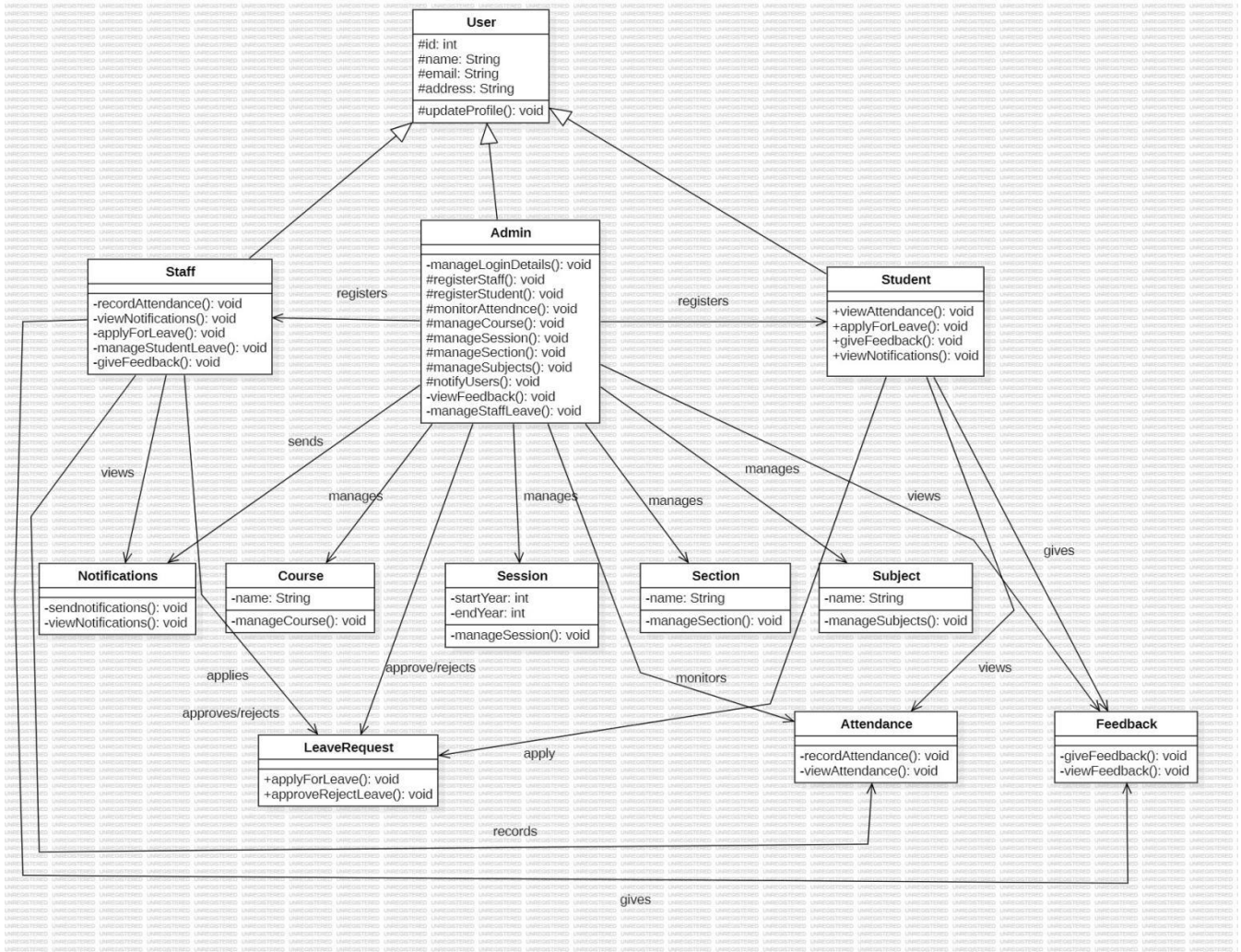

Fig 6.4: Class Diagram

Activity Diagram

An activity diagram is a type of behavioural diagram that represents the flow of control or the sequence of activities in a system. It shows the process from the start to the end, illustrating various possible decision paths, actions, and parallel activities that may occur within a system.

An activity diagram is valuable for visualizing dynamic aspects of the system, making it easier to understand complex workflows, identify potential bottlenecks, and refine the operational flow before implementation.

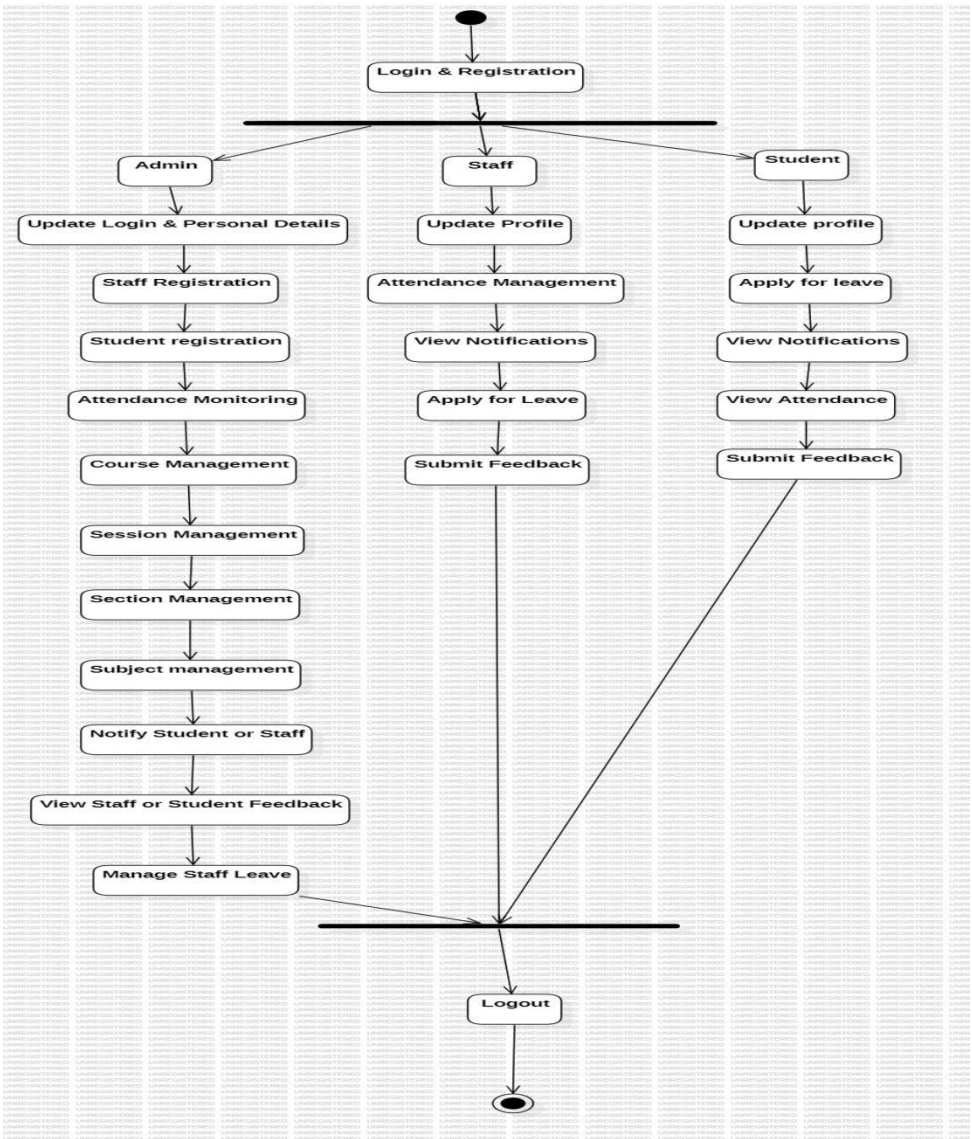

Fig 6.5: Activity Diagram

7.TECHNOLOGY DESCRIPTION

College Management System is a web-based platform designed using modern and reliable technologies that ensure seamless performance, security, and scalability. The choice of technologies was made based on project requirements, ease of development, community support, and future extensibility.

Frontend Technologies

The frontend of college management system is the part of the application that users interact with directly. It focuses on creating a user-friendly, responsive, and visually appealing interface to ensure a smooth and engaging user experience.

HTML5 (HyperText Markup Language)

HTML5 is the backbone of the web pages, providing the structural framework for all the content displayed in the browser. It is used to create and organize elements such as forms, buttons, tables, navigation bars, and image sections. In this project, HTML5 helps structure key pages like the homepage, login/signup pages, students listings, and dashboards.

CSS (Cascading Style Sheets)

CSS is used to control the appearance and layout of HTML elements. It enhances the visual appeal of the platform through styling techniques such as color schemes, spacing, borders, shadows, and animations. Media queries ensure responsiveness, allowing the website to adapt to different screen sizes for mobile, tablet, and desktop users.

JavaScript

JavaScript adds dynamic functionality and interactivity to the platform. It is used to perform client-side validation, toggle menus, show notifications, manage modals/popups, and enhance the overall user experience without requiring full-page reloads. This makes the interface more responsive and intuitive for staff, student and admin.

Python

Python is a high-level, interpreted programming language known for its readability, simplicity, and versatility. It is widely used in web development, data science, machine learning, and automation.

- **Readable Syntax**: Python's clean and simple syntax makes development faster and easier to understand.

- **Object-Oriented:** Python supports object-oriented programming, helping structure the application in reusable modules.
- **Extensive Libraries:** Offers rich libraries and frameworks for web development (like Django), data handling, API integrations, and more.
- **Cross-Platform:** Runs on multiple platforms including Windows, Linux, and macOS.

Django (High-Level Web Framework)

Django is an open-source Python framework that enables rapid development of secure and maintainable web applications. It follows the Model-View-Template (MVT) architecture**.**

Architecture (MVT):

- **Model:** Manages database operations and data structures using Django's ORM.
- **View:** Contains logic that handles requests and returns responses (HTML or JSON).
- **Template**: Handles the frontend interface that is shown to users.

Built-in Admin Panel: Auto-generated admin interface to manage users, sellers, pets, etc.

ORM (Object-Relational Mapper): Interact with the database using Python code instead of SQL.

Security: Built-in protection against SQL injection, CSRF, XSS, and clickjacking.

Authentication System: Handles user login, logout, registration, and role-based access.

Scalability: Suitable for both small apps and large-scale projects with high traffic.

URL Routing: Clean URL structure for navigating between pages and APIs.

Django REST Framework (DRF)

For API-based communication between the frontend and backend. It Handles CRUD operations via API endpoints. Sends and receives JSON data. Supports token-based authentication and permissions. Makes the platform future-ready for mobile apps or external integrations.

Database (SQLite3):

SQLite3 is a lightweight, fast, and serverless relational database engine that requires no separate server process. It stores the entire database in a single file, making it easy to set up and manage, especially for small to medium-sized applications. Django integrates seamlessly with SQLite3 using its Object Relational Mapper (ORM), allowing developers to perform database operations using Python code. It is an ideal choice for development environments due to its simplicity, portability, and minimal configuration needs.

Features:

- **Easy Setup:** No need for separate installation or server configuration.
- **Built-in with Django:** Works out-of-the-box for development projects.
- **Lightweight:** Stores the entire database in a single .db file.
- **Portable:** The database file can be easily transferred across systems.

8. TESTING

8.1 Introduction

Testing is a fundamental part of the software development process that ensures the quality, reliability, and functionality of an application. It involves systematically evaluating a software system or its components to identify any defects or errors and to verify that the system meets the specified requirements. The primary goal of testing is to detect bugs early in the development cycle and ensure that the final product is stable, secure, and performs as expected.

Software testing can be classified into various types, including manual testing and automated testing, as well as functional and non-functional testing. Common testing methods include unit testing, integration testing, system testing, and acceptance testing, each serving a unique purpose at different stages of development.

In the context of web applications like the College Management System, testing plays a crucial role in validating features such as user authentication, data management, attendance automation, and communication between different user panels (Admin, Staff, Student). Through thorough testing, developers can ensure a smooth user experience, minimize errors, and build a trustworthy and efficient software solution.

8.2 Types of Testing

1. Unit Testing

Unit testing involves testing individual components or functions of the software in isolation. The goal is to ensure that each small piece of code works as expected. Developers usually perform unit tests during the development phase using frameworks like `unittest` or `pytest` in Python.

2. Integration Testing

Integration testing checks how different modules or services work together. After individual units are tested, integration tests ensure that the interaction between them is smooth and that data is passed correctly between components.

3. System Testing

System testing validates the complete and integrated software system. It is conducted by QA teams to check whether the system meets the specified requirements. It covers end-to-end scenarios and mimics real-world usage.

4. Acceptance Testing

Also known as User Acceptance Testing (UAT), this is the final phase of testing where the system is tested by the end users or clients. The goal is to verify if the application meets business requirements and is ready for deployment.

5. Functional Testing

Functional testing checks if the application functions according to the specified requirements. It focuses on user interface, APIs, databases, security, and other functional aspects. It ensures that each feature performs correctly.

6. Non-Functional Testing

This type of testing evaluates non-functional aspects of the software, such as performance, scalability, usability, and security. It ensures that the software is efficient and can handle expected loads.

7. Regression Testing

Regression testing is performed after updates or changes in the codebase to ensure that existing functionalities still work correctly. It helps catch any bugs that may have been introduced during code modifications.

8. Smoke Testing

Smoke testing is a quick set of tests run after a build to ensure that the most important functions of the application are working. It's often called a “sanity check” before proceeding with deeper testing.

9. Load Testing

Load testing measures how the application behaves under a specific expected load. It helps identify performance bottlenecks and ensures the system can handle high traffic or user activity.

10. Security Testing

Security testing is done to identify vulnerabilities or threats in the software. It ensures that data is protected and the application is safe from unauthorized access and attacks.

8.3 Testing Strategy and Approach

1.Identify test objectives and goals to ensure alignment with project requirements and user expectations.

2. Define test scope, including functionalities, modules, and scenarios to be covered during testing.

3.Determine test environments, including development, staging, and production environments, to replicate real-world conditions.

4.Establish testing priorities based on criticality, complexity, and impact on the overall system

5. Develop test cases and scenarios covering various aspects such as functionality, usability, performance, security, and compatibility.

6. Assign responsibilities and roles to team members involved in the testing process, including testers, developers, and stakeholders.

7. Execute test cases systematically, following predefined test scripts and procedures to ensure consistency and thoroughness.

8. Record and document test results, including observations, defects, and recommendations for approvements.

9. Prioritize and triage defects based on severity, impact, and urgency, addressing critical issues first to minimize risks.

10. Conduct regression testing after each code change or update to ensure that existing functionalities remain unaffected.

11. Perform exploratory testing to uncover hidden defects or usability issues that may not be captured in predefined test cases.

12. Collaborate closely with developers to communicate findings, clarify requirements, and resolve issues promptly.

13. Continuously monitor and track testing progress, adapting strategies and approaches as needed to meet project timelines and goals.

14. Seek feedback from stakeholders and end-users to validate test coverage, usability, and overall satisfaction with the system.

8.4 TESTED MODULES

Login:

Test Case 1: The system was tested to ensure users can log in with valid credentials and are redirected to their respective dashboards.

Test Case 2: Attempts to log in with invalid credentials were correctly handled by displaying an error message.

Test Case 3: Logout functionality was verified to ensure users are securely logged out and redirected to the login page.

Student Profile Management:

Test Case 1: Students were able to view and update their profiles successfully.

Test Case 2: Any changes made to profile data were immediately reflected in the database.

Test Case 3: Profiles with missing or incomplete data prompted appropriate warning messages.

Staff Management (Admin Panel):

Test Case 1: The admin was able to add new staff members with all required details.

Test Case 2: Editing existing staff records updated the information correctly.

Test Case 3: Deleting staff members removed their records from the system without affecting related data.

Attendance Management:

Test Case 1: Staff were able to mark daily attendance for students, and this data was saved properly.

Test Case 2: Students could view their attendance records in a structured format.

Test Case 3: The system prevented staff from marking attendance multiple times for the same subject on the same day.

Result Management:

Test Case 1: Staff could enter student marks, and students could view their results along with calculated grades.

Test Case 2: Validation checks ensured that marks entered were within a valid range.

Test Case 3: Result data was accurately displayed to the respective students in their dashboard.

Leave Application Module:

Test Case 1: Both staff and students were able to apply for leave by submitting a form with appropriate details.

Test Case 2: Admins could view, approve, or reject leave requests, and the updated status was visible to the applicant.

Test Case 3: The system correctly handled multiple applications and displayed the history of previous leave requests.

Notification System:

Test Case 1: Admins were able to send notifications to students and staff, which appeared instantly in their dashboards.

Test Case 2: Notifications could be targeted to specific users or roles, ensuring effective communication.

Test Case 3: Users were notified about leave decisions, updates, and important announcements.

9.SAMPLE CODE

Models.py

```
from django.contrib.auth.hashers import make_password
from django.contrib.auth.models import UserManager
from django.dispatch import receiver
from django.db.models.signals import post_save
from django.db import models
from django.contrib.auth.models import AbstractUser

  class CustomUserManager(UserManager):
   def _create_user(self, email, password, **extra_fields):
      email = self.normalize_email(email)
      user = CustomUser(email=email, **extra_fields)
      user.password = make_password(password)
      user.save(using=self._db)
      return user

   def create_user(self, email, password=None, **extra_fields):
      extra_fields.setdefault("is_staff", False)
      extra_fields.setdefault("is_superuser", False)
      return self._create_user(email, password, **extra_fields)

   def create_superuser(self, email, password=None, **extra_fields):
      extra_fields.setdefault("is_staff", True)
      extra_fields.setdefault("is_superuser", True)

      assert extra_fields["is_staff"]
      assert extra_fields["is_superuser"]
      return self._create_user(email, password, **extra_fields)


class Session(models.Model):
   start_year = models.DateField()
   end_year = models.DateField()

   def __str__(self):
      return "From " + str(self.start_year) + " to " + str(self.end_year)

class Section(models.Model):
   name = models.CharField(max_length=120)
   created_at = models.DateTimeField(auto_now_add=True)
   updated_at = models.DateTimeField(auto_now=True)

   def __str__(self):
```

```
        return self.name

    class CustomUser(AbstractUser):
     USER_TYPE = ((1, "HOD"), (2, "Staff"), (3, "Student"))
     GENDER = [("M", "Male"), ("F", "Female")]


     username = None  # Removed username, using email instead
     email = models.EmailField(unique=True)
     user_type = models.CharField(default=1, choices=USER_TYPE, max_length=1)
     gender = models.CharField(max_length=1, choices=GENDER)
     profile_pic = models.ImageField()
     address = models.TextField()
     section = models.ForeignKey(Section, on_delete=models.DO_NOTHING, null=True,
     blank=True)
     hall_ticket_number = models.CharField(max_length=15, unique=False, blank=True, default=" ")
     fcm_token = models.TextField(default="")  # For firebase notifications
     created_at = models.DateTimeField(auto_now_add=True)
     updated_at = models.DateTimeField(auto_now=True)
     USERNAME_FIELD = "email"
     REQUIRED_FIELDS = []
     objects = CustomUserManager()

     def __str__(self):
        return self.first_name + " " + self.last_name


class Admin(models.Model):
     admin = models.OneToOneField(CustomUser, on_delete=models.CASCADE)



class Course(models.Model):
     name = models.CharField(max_length=120)
     created_at = models.DateTimeField(auto_now_add=True)
     updated_at = models.DateTimeField(auto_now=True)

     def __str__(self):
        return self.name



class Staff(models.Model):
     course = models.ForeignKey(Course, on_delete=models.DO_NOTHING, null=True,
     blank=False)
     admin = models.OneToOneField(CustomUser, on_delete=models.CASCADE)
```

```
    def __str__(self):
        return self.admin.first_name +' '+ self.admin.last_name

class Subject(models.Model):
    name = models.CharField(max_length=120)
    staff = models.ForeignKey(Staff,on_delete=models.CASCADE,)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    updated_at = models.DateTimeField(auto_now=True)
    created_at = models.DateTimeField(auto_now_add=True)
    def __str__(self):
        return self.name



class Student(models.Model):
    admin = models.OneToOneField(CustomUser, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.DO_NOTHING, null=True,
    blank=False)
    session = models.ForeignKey(Session, on_delete=models.DO_NOTHING, null=True)
    section = models.ForeignKey(Section, on_delete=models.DO_NOTHING, null=True)
    hall_ticket_number = models.CharField(max_length=15, unique=True, blank=True, default=" ")
    # Allow admin to enter

    def clean(self):
        from django.core.exceptions import ValidationError
        # check hall_ticket_number is alphanumeric or not
        if self.hall_ticket_number and not self.hall_ticket_number.isalnum():
            raise ValidationError("Hall ticket number must be alphanumeric.")
    def __str__(self):
        return self.admin.first_name +' '+ self.admin.last_name



class Attendance(models.Model):
    session = models.ForeignKey(Session, on_delete=models.DO_NOTHING)
    subject = models.ForeignKey(Subject, on_delete=models.DO_NOTHING)
    section = models.ForeignKey(Section, on_delete=models.DO_NOTHING)
    date = models.DateField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)



class AttendanceReport(models.Model):
    student = models.ForeignKey(Student, on_delete=models.DO_NOTHING)
```

```
    attendance = models.ForeignKey(Attendance, on_delete=models.CASCADE)
    status = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class LeaveReportStudent(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    date = models.CharField(max_length=60)
    message = models.TextField()
    staff = models.ForeignKey(Staff,on_delete=models.CASCADE,)
    status = models.SmallIntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class LeaveReportStaff(models.Model):
    staff = models.ForeignKey(Staff, on_delete=models.CASCADE)
    date = models.CharField(max_length=60)
    message = models.TextField()
    status = models.SmallIntegerField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class FeedbackStudent(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    feedback = models.TextField()
    reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class FeedbackStaff(models.Model):
    staff = models.ForeignKey(Staff, on_delete=models.CASCADE)
    feedback = models.TextField()
    reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class NotificationStaff(models.Model):
    staff = models.ForeignKey(Staff, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
```

```
class NotificationStudent(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


class StudentResult(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    subject = models.ForeignKey(Subject, on_delete=models.CASCADE)
    test = models.FloatField(default=0)
    exam = models.FloatField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)


@receiver(post_save, sender=CustomUser)
def create_user_profile(sender, instance, created, **kwargs):
    if created:
        if instance.user_type == 1:
            Admin.objects.create(admin=instance)
        if instance.user_type == 2:
            Staff.objects.create(admin=instance)
        if instance.user_type == 3:
            Student.objects.create(admin=instance)


@receiver(post_save, sender=CustomUser)
def save_user_profile(sender, instance, **kwargs):
    if instance.user_type == 1:
        instance.admin.save()
    if instance.user_type == 2:
        instance.staff.save()
    if instance.user_type == 3:
        instance.student.save()
```

Views.py

```
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,Section

# Create your views here.
def home(request):
    return render(request,'main_app/home.html')

  def about(request):
 return render(request,'main_app/about.html')

def contact(request):
    return render(request,'main_app/contact.html')

  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 HttpResponse("<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 = "6LfswtgZAAAAABX9gbLqe-d97qE2g1JP8oUYritJ"
        data = {
            'secret': captcha_key,
            'response': captcha_token
        }
        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('main_app/login.html')
        except:
            messages.error(request, 'Captcha could not be verified. Try Again')
            return redirect('main_app/login.html')

        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("main_app/login.html")


    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')
     section_id = request.POST.get('section')
     try:
        subject = get_object_or_404(Subject, id=subject_id)
        session = get_object_or_404(Session, id=session_id)
        section = get_object_or_404(Section, id=section_id)
        attendance = Attendance.objects.filter(subject=subject, session=session, section=section)
        print(attendance)
        attendance_list = []
        for attd in attendance:
            data = {
                    "id": attd.id,
                    "attendance_date": str(attd.date),
                    "session": attd.session.id,
```

```
                "section": attd.section.id
                }

            attendance_list.append(data)
        return JsonResponse(json.dumps(attendance_list), safe=False)
    except Exception as e:
        return e

  from .forms import ContactForm

def contact(request):
    if request.method == 'POST':
        form = ContactForm(request.POST)
        if form.is_valid():
            # Process the form data (e.g., save to database or send email)
            name = form.cleaned_data['name']
            email = form.cleaned_data['email']
            message = form.cleaned_data['message']

            return redirect('home')  # Redirect to success page
    else:
        form = ContactForm()

    return render(request, 'main_app/contact.html', {'form': form})

def showFirebaseJS(request):
    data = ""
importScripts('https://www.gstatic.com/firebasejs/7.22.1/firebase-app.js');
importScripts('https://www.gstatic.com/firebasejs/7.22.1/firebase-messaging.js');
  firebase.initializeApp({
    apiKey: "AIzaSyBarDWWHTfTMSrtc5Lj3Cdw5dEvjAkFwtM",
    authDomain: "sms-with-django.firebaseapp.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"
});
  const messaging = firebase.messaging();
messaging.setBackgroundMessageHandler(function (payload) {

    return self.registration.showNotification(payload.notification.title, notificationOption);
});
    """
     return HttpResponse(data, content_type='application/javascript')
```

10. SCREENSHOTS

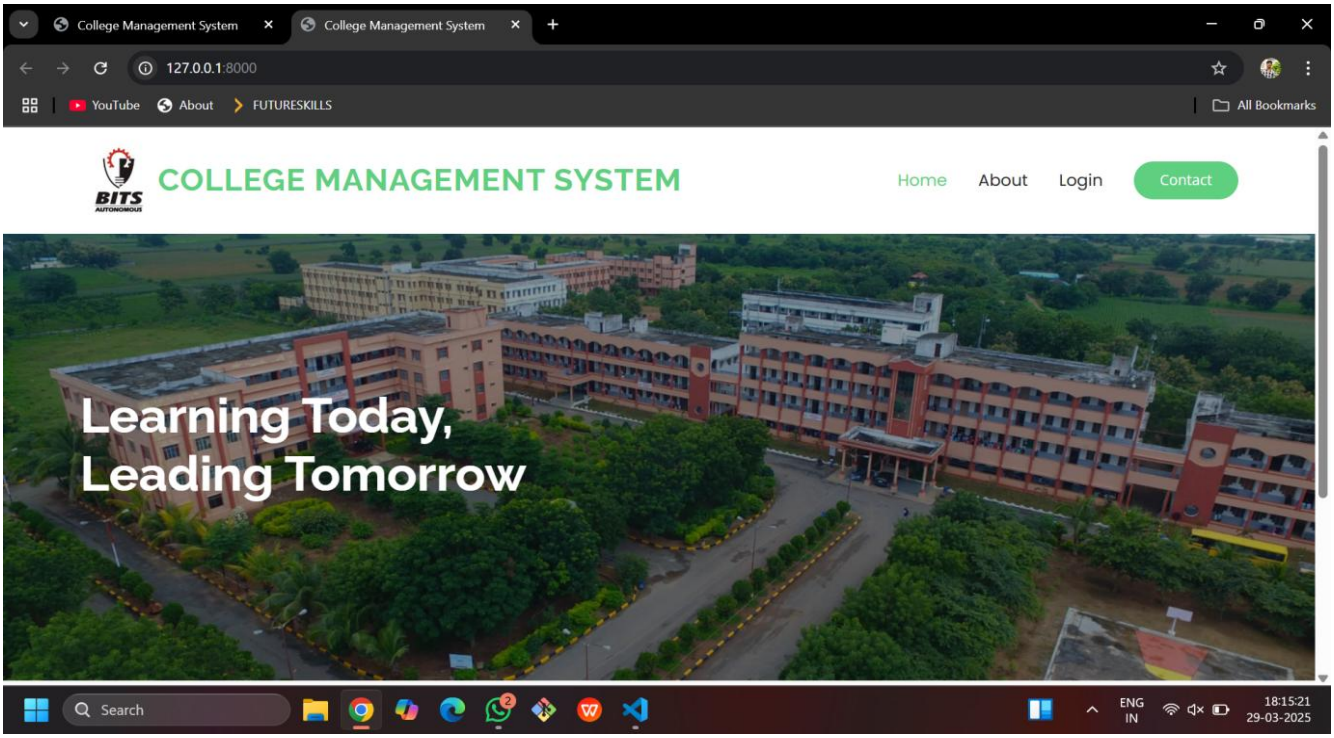

Fig 10.1 Home Page

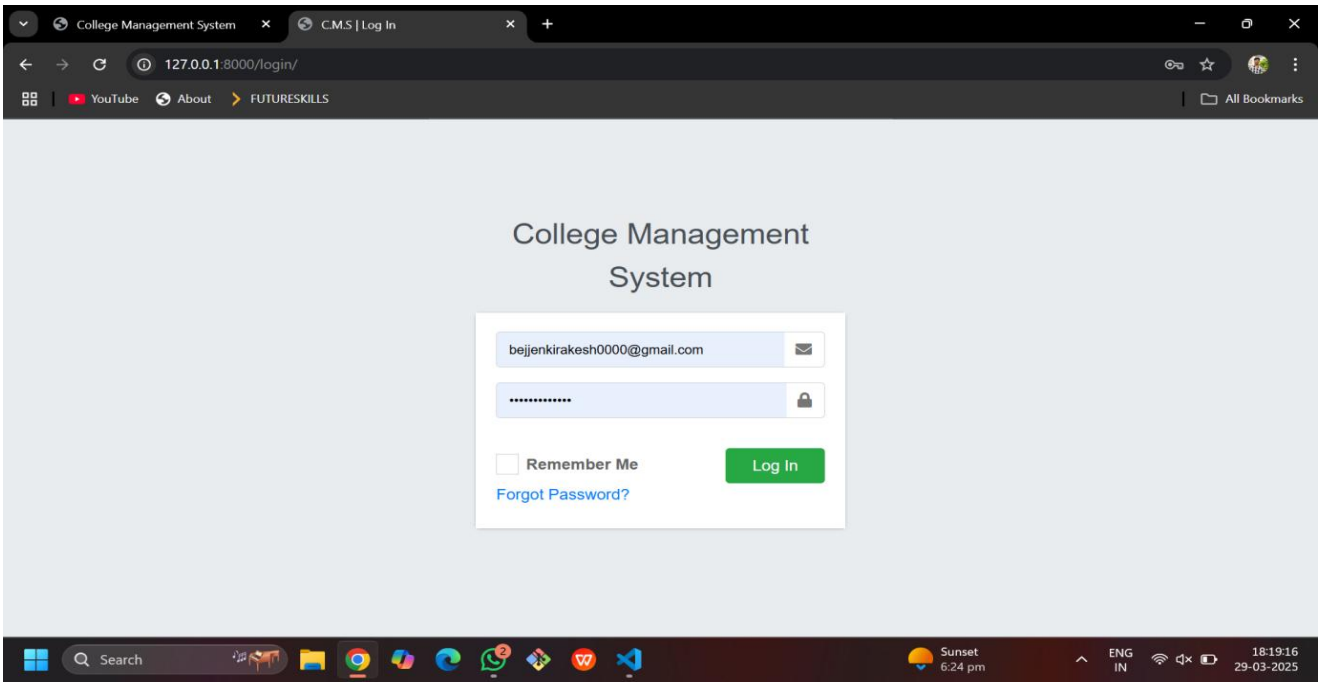

Fig 10.2 Login Page

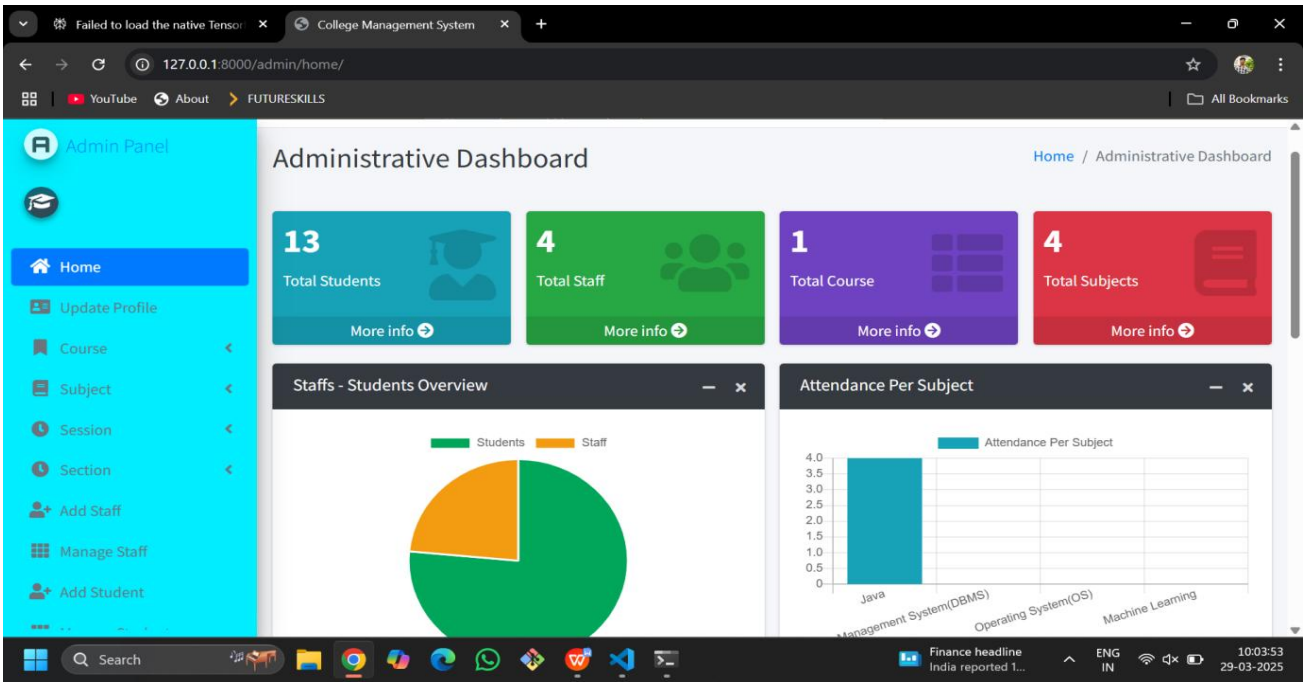

Fig 10.3 Admin Dashboard

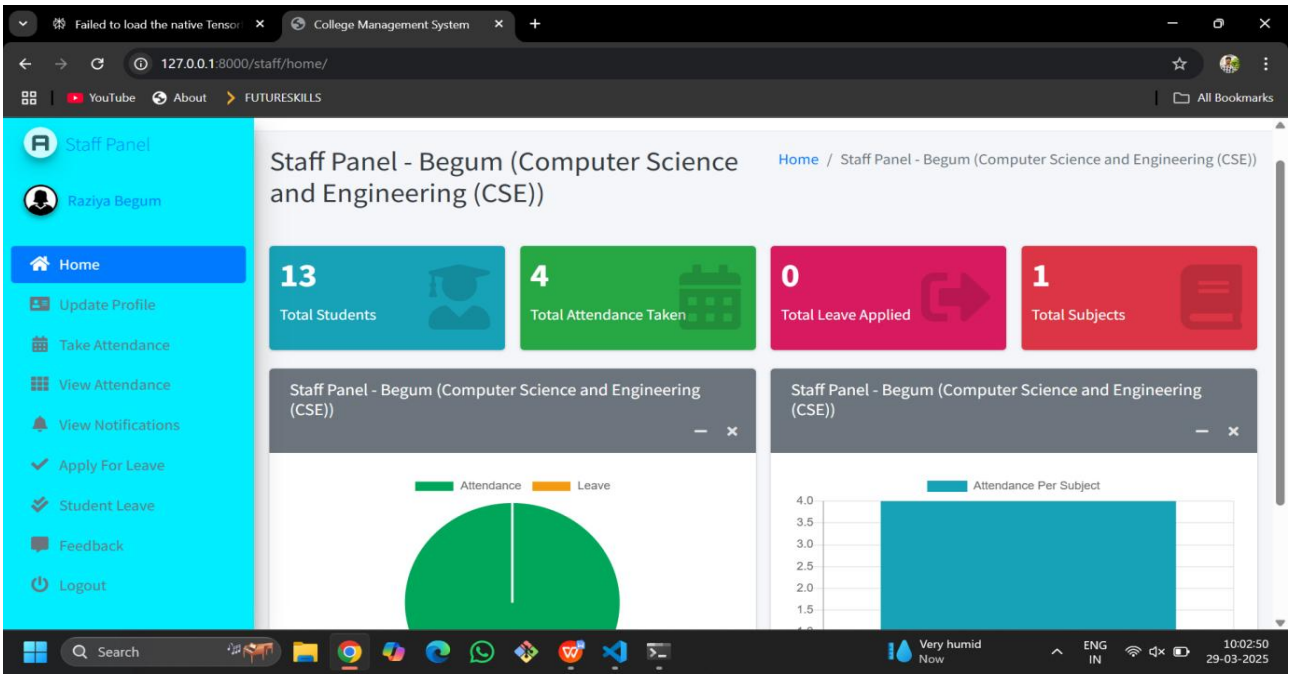

Fig 10.4 Staff Dashboard

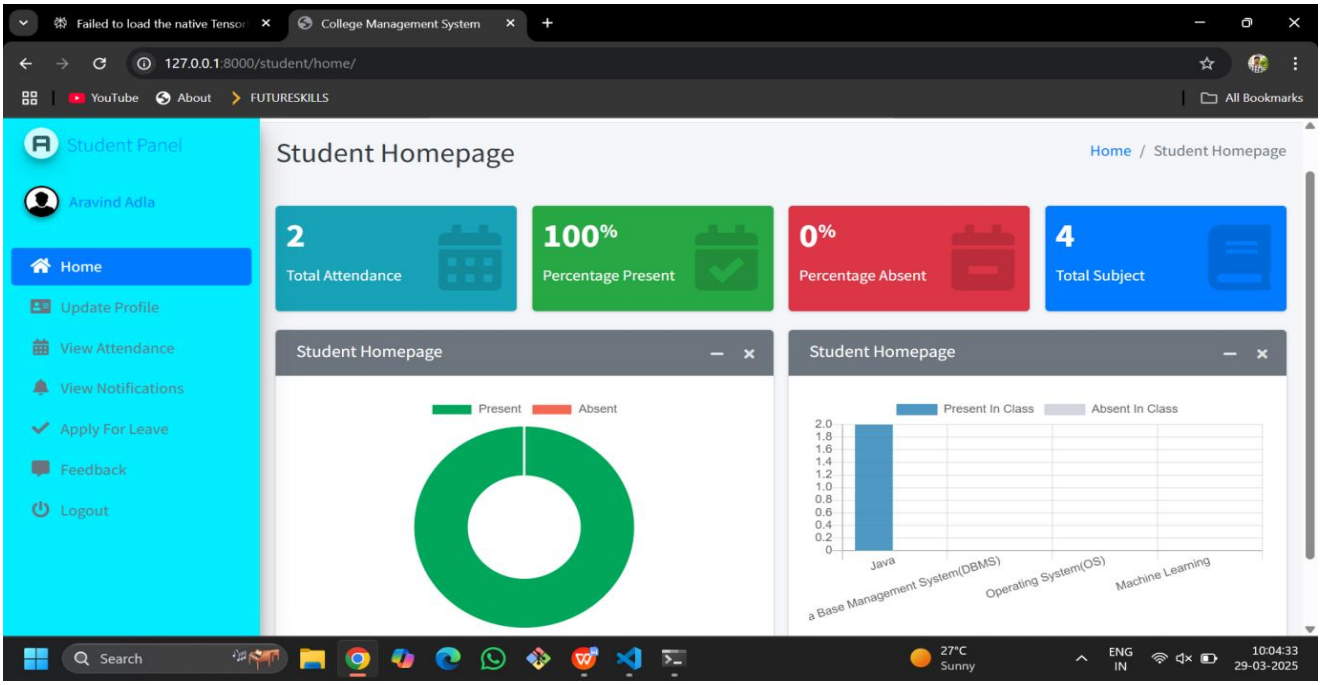

Fig 10.5 Student Dashboard

11. FUTURE ENHANCEMENTS

Future enhancement of this project can be done by using advanced technologies like Artificial Intelligence and Machine Learning to implement predictive analytics for student performance. Integrate fingerprint or face recognition for more accurate and efficient attendance tracking. We can also develop a mobile app for easy access to the system on phones. We can also implement real time notifications for important updates, deadlines, and events. Additionally we can add built-in video conferencing capabilities for online classes and meetings. And including a module for job postings, internship opportunities, and career counselings. We can be implement more comprehensive data visualization tools for better user interface and decision making. Additional security measures can be implemented to protect sensitive data. These enhancements will transform the system into a robust platform that will meet the requirements of the administrators, staff, and as well as students.

12. CONCLUSION

In conclusion, College Management System that was developed with Django addresses the inefficiencies of traditional college institutions and is comprised of a student panel for student to manage their academic responsibilities, attendance, grades, schedules, and leave requests, staff panel to mange the inefficient processes of administrative staff so they can put more focus on teaching and improving student experience,and the Admin panel to manage the centralized skills of attendance tracking, result management, timetabling, and leave request processing. This system reduces drawbacks of manual systems for various reasons, including data loss, delays, and workload increase, improves communication between college community members, and leads to increased efficiency throughout the college. Finally it lays the foundation for future potential improvements in technology - including mobile application ability, insights from artificial intelligence on student experience. Overall, this College Management System represents a real step forward in creating organizational efficiencies of academic instituitions.

13. REFERENCES

[1] Tang, Y.F., & Zhang, Y.S. (2009, August). design and implementation

Of college student information management system based on web Services. IT in Medicine & Education, 2009. IEEE International Symposium on (Vol.1, pp. 1044-1048), IEEE.

[2] Hashim, N. M. Z., & Mohamed, S. N. K. S. (2013). Development of student information system. International Journal of Science and Research (IJSR),2,256-260

[3] Patnaik, S., Kumari Singh, K., Ranjan,R., & kumari, N. (2016). College Management System. International Research Journal of Engineering and Technology (IRJET), 3(5), 659-661.

[4] D B Heras, D. Otero, and F. Arguello, "An eco feedback system for improving the sustainability Performance of universities, " in Proc. 2011 IEEE International Conference on Virtual Environment Human-Computer Interfaces and Measurement Systems, Ottawa, OB 2011, pp. 1-6

[5] Y Wang, B Y Sun, and F Cheng, "Electronic document- based process model for image archives in universities, " in Proc. 2011 International Conference on Information Technology, Computer Engineering, and Management sciences, Nanjing, Jiangsu, pp. 57-60.

[6] X.X.Xin, R.M.Wu, and H.H.Li,"A framework model of the e-campus management system based on SOA," in Proc.2009 International Conference on computational Intelligence and software Engineering Wuhan, 2009,pp.

[7] H. M. Weiand L.J. He, "Constructing the comprehensive academic affairs management system based on SOA,"in Proc. 2009 1st International Conference on Information Science and Engineering, Nanjing, jiangsu, pp.3261- 3264.

[8]S. Jeyalatha, B. VijayKumar, and G.S. Wadhwa, "Design and implementation of web based application for relational data maintenance in n university environment," in Proc.2011 International Conference and Workshop on current Trends in Information Technology, Dubai, pp. 105-112

[9] Bharmagoudar, S. R., Geeta, R. B., & Totad, S.G.(2013). Web based student information management system. International Journal of Advnced Research in Computer and Communication Engineering,2(6), 2342-2348.

[10] Norasiah, M.A., & Norhayati, A. (2003, January). Intelligent student information system. 4th National Conference on Telecommunication Technology, NCTT 2003, Proceedings. (pp. 212-215), IEEE.

[11] Rajmane, S. S., Mathpati, S. R., & Dawlw, J.K. (2016). Digitalization of Management System for College and Student Information. Research Journal of Science and Technology, 8(4), 179-180.

[12] A. Sharma and P. Mehta, "College ERP system for effective resource management," International Journal of Computer Applications, vol. 90, no. 6, pp. 1–6, Mar. 2014.

[13] R. K. Tripathi and S. Pandey, "A web-based college management system," International Journal of Engineering Research & Technology (IJERT), vol. 2, no. 4, pp. 850–854, Apr. 2013.

[14] A. Gupta, R. Jain, and P. Sinha, "Design and development of student information management system," in Proc. 2012 IEEE International Conference on Information Systems Design and Intelligent Applications, Visakhapatnam, India, pp. 215–220.

[15] N. Patel and M. Desai, "An integrated approach to college management systems using PHP and MySQL," International Journal of Scientific Research in Computer Science, Engineering and Information Technology, vol. 3, no. 2, pp. 345–350, 2018.

[16] S. Rao, M. Reddy, and R. Joshi, "Cloud-based architecture for university management systems," in Proc. 2015 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM), Bangalore, India, pp. 51–56.