ACADEMIC EASE MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

JOEL SUNDARSINGH A 220701110

KAILAASH B 220701115

ISHANT R 220701093

In partial fulfilment for the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI

An Autonomous Institute
Thandalam
Chennai - 602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this project report "**ACADEMIC EASE MANAGEMENT SYSTEM**" is the Bonafide work of **"JOEL SUNDARSINGH A (220701110), KAILAASH B (220701115), ISHANT R (220701093)"** who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.R.SABITHA
Professor and Academic Head,
Computer Science and Engineering,
Rajalakshmi Engineering College (Autonomous),
Thandalam, Chennai - 602 105.

SIGNATURE

Dr.G.Dharani Devi
Associate Professor ,
Computer Science and Engineering,
Rajalakshmi Engineering College (Autonomous),
Thandalam, Chennai - 602 105.

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

This report specifies the various processes and techniques used in gathering requirements, designing, implementing, and testing for the project on college management system. The problems regarding the current system in the college were analysed and noted. This project aims to solve some of those problems and thus, add more value to the current system. The requirements were gathered from all the stakeholders and based on that we created requirements, models and designed the software based on the based. The project was implemented in the form of a website using Django(python). Using the various resources and tools we gathered along way, we implemented the Academic Ease Management system using some features that solve the current problems in the system such as a provision to edit the attendance and marks before locking it at the end. The software was also tested using the various testing methods and results were positive. Thus, the results can be integrated in the current ERP system to improve It's working and solve some of the existing problems.

TABLE OF CONTENTS

CHAPTER 1
INTRODUCTION
1.1 INTRODUCTION

In the realm of educational institutions, the effective management of academic processes plays a pivotal role in ensuring smooth operations and enhancing student experiences. The Academic Ease Management System (AEMS) is a comprehensive solution designed to streamline various academic activities within an educational institution, leveraging the power of Database Management System (DBMS) technology.

The AEMS is crafted to address the multifaceted needs of academic administrators, faculty members, and students, providing them with a user-friendly platform to manage, monitor, and track various academic tasks efficiently. By harnessing the capabilities of a robust DBMS, AEMS facilitates seamless data storage, retrieval, and manipulation, ensuring the integrity and security of academic information.

This mini-project aims to delve into the development of a prototype AEMS, focusing on key functionalities such as student enrolment, course management, grading, and academic scheduling. Through the implementation of a well-structured database schema and intuitive user interfaces, the project aims to demonstrate how DBMS can be leveraged to enhance academic management processes.

The project will encompass the design and creation of a relational database schema, incorporating entities such as students, courses, instructors, academic departments, and administrative staff. By establishing relationships between these entities and implementing appropriate constraints, the database will serve as a centralized repository for academic data, facilitating efficient data management and retrieval.

Moreover, the project will involve the development of user interfaces tailored to the specific needs of administrators, faculty members, and students. These interfaces will provide intuitive functionalities for tasks such as course registration, grade entry, academic scheduling, and generating reports. Through seamless integration with the underlying database, these interfaces will enable users to interact with academic data effortlessly, thereby enhancing productivity and user satisfaction.

1.2 OBJECTIVES

- **Efficiency Enhancement**: Develop a software solution that improves the efficiency and flexibility of college record management by automating manual processes and providing easy access to up-to-date and accurate information.
- **Integrated Platform**: Create a common platform that enables teachers, Heads of Departments (HoDs), students, and administrators to access student information seamlessly, thereby facilitating effective communication and collaboration within the college ecosystem.
- **Module Management:** Design and implement various modules such as student, faculty, and admin, each interconnected to streamline operational tasks and reduce the time required for administrative activities.
- **Comprehensive Data Management**: Develop a system that comprehensively computerizes all college details, eliminating the need for manual maintenance and enabling a single user to manage reports and records efficiently.
- **Security Enhancement**: Provide customizable security features to ensure that access to sensitive information is restricted based on user roles and permissions, thereby safeguarding the confidentiality and integrity of college data.
- **User-Friendly Interface**: Design intuitive user interfaces that are easy to navigate and understand, promoting user adoption and minimizing the learning curve for teachers, HoDs, students, and administrators.
- **Real-Time Reporting:** Implement real-time reporting functionalities to generate accurate and timely reports on various aspects of college management, including student performance, attendance, and academic progress.
- **Scalability and Flexibility:** Develop a scalable and flexible system that can adapt to the evolving needs of the college, accommodating changes in academic programs, courses, and administrative processes over time.
- **Integration with Academic Processes**: Ensure seamless integration with academic processes such as Continuous Internal Evaluation (CIE) and Semester End Examination (SEE), providing support for conducting examinations, grading, and result processing.
- **User Training and Support**: Provide comprehensive user training and ongoing support to ensure that stakeholders are proficient in using the Academic Ease Management System, thereby maximizing its effectiveness and utility within the college community.

1.3 MODULES

The Academic Ease Management system has three main user classes. These include the students, teachers, and administrator. This section will explain in detail all the features and the working of those for each user class.

1.3.1 STUDENT

1.3.1.1 LOGIN

Each student in the college is assigned a unique username and password by the administrator. The username is the same as their USN and so is the password. They may change it later according to their wish.

1.3.1.2 HOMEPAGE

After successful login, the student is presented a homepage with their main sections, attendance, marks, and timetable. In the attendance section the student can view their attendance status which includes the total classes, attended classes and the attendance percentage for each of their courses. In the marks section, the student can view the marks for each of their courses out of 20 for 3 internal assessments, 2 events. Also, the semester end examination for 100 marks. Lastly, the timetable provides the classes assigned to that student and day and time of each in a tabular form.

1.3.1.3 ATTENDANCE

On the attendance page, there is a list of courses that is dependent on each student. For each course, the course id and name are display along with the attended classes, total classes, and the attendance percentage for that course. If the attendance percentage is below 75 for any course, it is displayed in red denoting shortage of attendance, otherwise it is green. If there is any shortage, it specifies the number of classes to attend to make up for it. If you click on each course, it takes you to the attendance detail page.

1.3.1.4 ATTENDANCE DETAILS

This page displays more details for the attendance in each course. For each the course, there is a list of classes conducted and each is marked with the date, day and whether the student was present or absent on that date.

1.3.1.5 MARKS

The Marks page is a table with an entry for each of their courses. The course id and name are specified along the marks obtained in each of the tests and exams. The tests 23 include 3 internal assessments with marks obtained out of a total of 20, 2 events such as project, assignment, quiz etc., with marks out of 20. Lastly, one semester end exam with marks out of 100.

1.3.1.6 TIMETABLE

This page is a table which lists the day and timings of each of the classes assigned to the student. The row headers are the days of the week and the column headers are the time slots. So, for each day, it specifies the classes in the time slots. The timetable is generated automatically from the assign table, which is a table containing the information of all the teachers assigned to a class with a course and the timings the classes.

1.3.2 TEACHER

1.3.2.1 LOGIN

Each teacher in the college is assigned a unique username and password by the administrator. The username is their teacher ID and the same for password. The teacher may change the password later.

1.3.2.2 HOMEPAGE

After successful login, the student is presented a homepage with their main sections, attendance, marks, timetable, and reports. In the attendance section, the teacher can enter the attendance of their respective students for the days on which classes were conducted. There is a provision to enter extra classes and view/edit the attendance of each individual student. In the marks section, the teacher may enter the marks for 3 internals, 2 events and 1 SEE for each student. They can also edit each of the entered marks. The timetable provides the classes assigned to the teacher with the day and timings in a tabular form. Lastly, the teacher can generate reports for each of their assigned class.

1.3.2.3 ATTENDANCE

There is a list of all the class assigned to teacher. So, for each class there are 3 actions available. They are,

1.3.2.4 ENTER ATTENDANCE

On this page, the classes scheduled or conducted is listed in the form of a list. Initially, all the scheduled classes will be listed from the start of the semester to the current date. Thus, if there is class scheduled for today, it will automatically appear on top of the list. If the attendance of any day is not marked it will be red, otherwise green if marked. Classes can also be cancelled which will make that date as yellow. While entering the attendance, the list of students in that class is listed and there are two options next to each. These options are in the form of a radio button for present and absent. All 24 the buttons are initially marked as present and the teacher just needs to change for the absent students.

1.3.2.5 EDIT ATTENDANCE

After entering attendance, the teacher can also edit it. It is like to screen for entering attendance, only the entered attendance is saved and display. The teacher can change the appropriate attendance and save it.

1.3.2.6 EXTRA CLASS

If a teacher has taken a class other than at the scheduled timings, they may enter the attendance for that as well. While entering the extra class, the teacher just needs to specify the date it was conducted and enter the attendance of each of the students. After submitting extra class, it will appear in the list of conducted classes and thus, it can be edited

1.3.2.7 STUDENT ATTENDANCE

For each assigned class, the teacher can view the attendance status of the list of students. The number of attended classes, total number of classes conducted and the attendance percentage is displayed. If the attendance percentage of any of the students is below 75, it will be displayed in red. Thus, the teacher may easily find the list of students not eligible to take a test.

1.3.2.8 STUDENT ATTENDANCE DETAILS

The teacher can view the attendance detail of all their assigned students individually. That is, for all the conducted classes, it will display whether that student was present or absent. The teacher can also edit the attendance of each student individually by changing the attendance status for each conducted class

1.3.2.9 MARKS

On this page, the list of classes assigned to the teacher are displayed along with two actions for each class. These actions are,

1.3.2.10 ENTER MARKS

On this page, the teacher can enter the marks for 3 internal assessments, 2 events and one semester end exam. Initially all of them are marked red to denote that the marks have not been entered yet. Once the marks for a test is entered, it turns green. While entering the marks for a particular test, the list of students in that class is listed and marks can be entered for all of them and submitted. Once, the marks are submitted, the students can view their respective marks. In Case, if there is a need to change the marks of any student, it is possible to edit the marks.

1.3.2.11 EDIT MARKS

Marks for a test can be edited. While editing, the list of students in that class is displayed along with already entered marks. The marks to be updated can be changed and submitted. The students can view this change immediately

1.3.2.12 STUDENT MARKS

For each assigned class, the teacher has access to the list of students and the marks they obtained in all the tests. This is displayed in a tabular form.

1.3.2.13 TIMETABLE

This page is a table which lists the day and timings of each of the classes assigned to the teacher. The row headers are the days of the week and the column headers are the time slots. So, for each day, it specifies the classes in the time slots. The timetable is generated automatically from the assign table, which is a table containing the information of all the teachers assigned to a class with a course and the timings the classes.

1.3.2.14 FREE TEACHERS

For each entry in the table, the list of free teachers can be generated. Free teachers are the teachers who assigned to the class and are free for that time slot on that day. This is very useful for the teachers particularly when they are on leave as it helps them find a suitable replacement are that class.

1.3.2.15 REPORTS

The last page for the teachers is used to generate reports for each class. The report specifies the list of students in that class and their respective CIE and attendance percentage. CIE is the average of the marks obtained from the tests, 3 internals and 2 events. The CIE is out of 50 and the students with CIE below 25 are marked in red and are not eligible to write the semester end exam. Also, the attendance percentage is displayed with students below 75% marked in red.

1.3.3 ADMINISTRATOR

The administrator is responsible for adding and maintaining all the departments, students, teachers, classes, and courses. All this data is stored in the database in their respective tables. The admin is also responsible for adding and maintaining the list of teachers assigned to class with a course and the timings. This information is stored in the Assign table. The admin also has access to the marks and attendance of each student and can modify them. There are several features in place to ensure that querying the database is quick and efficient for the administrator. As the database has the potential to become huge, there is a search feature for every table including student, teacher etc. The search has got a specific record based on name or id. Also, it can filter the record based on department, class etc.

1.4 PURPOSE

The purpose is to design software for college database which contains up to date or accurate information of the college. That should improve efficiency and flexibility of college record management and to provide a common and or simple platform for everyone to access the student's information.College Automation System consists of different modules such as student, faculty, admin etc. Our main purpose is to create a software which will manage the working of these different modules. The interconnectivity among modules reduces the time to perform different operational task.

1.5 SCOPE

College management is becoming a very essential component in education in this modern day age. With the help of College Automation System we can gather all the useful information needed to the management in few clicks. The College ERP system now computerizes all the details that are maintained manually. Once the details are fed into the system or computer there is no need for various persons to deal with separate sections. Only a person is enough to maintain all the reports and records. The security can also be given as per the user requirement.

1.6 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

• Department The educational sub bodies of the college, which can offer several Programmes and Courses. Each department is managed by a HoD (Head of the Department)

• Course The subject offered by a Department in a semester, which is compulsary for a Student to take in order to make him/her eligible for SEE, and subsequently, award of BE Degree.

• Semester The 5 Month (or 2 month, in case of supplementary) Duration in student is offered a set of courses by a department and the courses are conducted in a part time or full time fashion. Each academic year consists of 3 semesters, out of which 2 are regular and 1 is supplementary.

• CIE Continuous Internal Evaluation, series of examinations con- ducted throughout the semester to assess the academic per- formance of the student. CIE conducted in the form of events (Usually 5). Finally, CIE is reduced to a total of 50 marks

• SEE Semester End Examination, conducted at the end of each semester to asses the academic performance of the student. Conducted for 100 marks and reduced to 50 marks.

CHAPTER 2
SURVEY OF TECHNOLOGIES
2.1 SOFTWARE DESCRIPTION

ARCHITECTURE DESCRIPTION

Designing a detailed architecture for a College ERP system involves creating a multi-layered structure to ensure the system is scalable, maintainable, and secure. Here is a detailed description of the architecture consisting of presentation, application, data access, and database layers

Presentation Layer

Purpose: The presentation layer is responsible for displaying the user interface and handling user interactions. It is the front end of the ERP system that users interact with.

Components:

Web Interface: Built using HTML, CSS, JavaScript, and frameworks like React, Angular, or Vue.js. It provides a responsive and user-friendly interface for accessing the ERP system from web browsers.

Mobile Interface: Developed using native languages (Swift for iOS, Kotlin/Java for Android) or cross-platform frameworks (React Native, Flutter) to allow mobile access to the ERP system.

Desktop Interface:(Optional) For systems that require a desktop application, technologies like Electron can be used to create cross-platform desktop applications.

Responsibilities:

- ✓ Render user interfaces and present data to users.
- ✓ Capture user inputs and send them to the application layer.
- ✓ Ensure a consistent user experience across different devices and platforms.

Application Layer

Purpose: The application layer contains the business logic and workflows of the ERP system. It processes user requests, applies business rules, and interacts with the data access layer.

Components:

Web Application Server: Typically implemented using frameworks like Spring Boot (Java), Django (Python), ASP.NET (C#), or Node.js (JavaScript). It handles HTTP requests and coordinates the execution of business logic.

Business Logic Layer: Encapsulates the core business rules and operations of the ERP system. It ensures that all processes, such as student registration, attendance recording, and internal marks entry, adhere to defined business rules.

Authentication and Authorization: Manages user authentication (login, logout) and enforces authorization rules to ensure users can only access the resources they are permitted to.

Responsibilities:

- ✓ Process incoming requests from the presentation layer.
- ✓ Execute business logic and validate inputs.
- ✓ Manage user sessions and enforce security policies.

Data Access Layer

Purpose: The data access layer provides an abstraction for accessing the database. It ensures that business logic in the application layer does not need to directly interact with the database.

Components:

Data Access Objects (DAOs): Classes that encapsulate the logic for accessing the database. They provide methods for CRUD (Create, Read, Update, Delete) operations on the database.

ORM Framework: An Object-Relational Mapping (ORM) framework like Hibernate (Java), Entity Framework (C#), or Sequelize (Node.js) can be used to map database tables to objects, making it easier to work with the database using object-oriented programming languages.

Repositories: High-level abstractions that manage data access operations, often built on top of DAOs and ORM frameworks.

Responsibilities:

- ✓ Provide a clean and simple API for the application layer to perform database operations.
- ✓ Manage connections to the database and ensure efficient query execution.
- ✓ Handle transactions to ensure data integrity and consistency.

Database Layer

Purpose: The database layer stores the data used by the ERP system. It is responsible for ensuring data is stored securely, efficiently, and reliably.

Components:

RDBMS: A relational database management system like MySQL, PostgreSQL, Microsoft SQL Server, or Oracle Database. It manages the data storage, retrieval, and management.

Database Schema: The structure of the database, including tables, relationships, indexes, and constraints. Key tables might include Students, Teachers, Courses, Attendance, InternalMarks, Departments, etc.

Stored Procedures and Triggers: SQL code that is stored and executed in the database to enforce business rules and automate repetitive tasks.

Responsibilities:

- ✓ Store and manage data in a structured manner.
- ✓ Ensure data integrity through constraints and referential integrity.
- ✓ Provide backup and recovery solutions to protect data.

Integration and Communication

Purpose: Ensure seamless communication between different layers and components of the ERP system.

Components:

APIs and Web Services: RESTful APIs or GraphQL services to facilitate communication between the presentation layer and the application layer, as well as between different modules within the application layer.

Middleware: Tools like message brokers (RabbitMQ, Kafka) for asynchronous communication and integration between different parts of the system.

Security Protocols: SSL/TLS for secure data transmission, OAuth for secure API authentication, and encryption for sensitive data storage.

Example Workflow

1. User Interaction:

- A student logs into the ERP system via the web interface.
- The presentation layer sends the login credentials to the application layer.

2. Authentication:

- The application layer verifies the credentials using the authentication component.
- If authenticated, a session is created, and the user is granted access to the relevant modules.

3. Data Request:

- The student requests to view their attendance report.
- The application layer processes the request and interacts with the data access layer to retrieve the relevant data from the database.

4. Data Access:

- The data access layer uses DAOs and ORM to fetch the attendance data from the database.
- The data is returned to the application layer.

5. Response:

- The application layer applies any necessary business logic (e.g., calculating attendance percentages) and sends the response back to the presentation layer.
- The presentation layer displays the attendance report to the student.

MODEL-VIEW-CONTROLLER (MVC) DESCRIPTION

Model

Purpose: The Model component represents the data and the business logic of the application. It directly manages the data, logic, and rules of the application.

Components:

- **Entities:** Classes that represent the database tables (e.g., Student, Teacher, Course, Department, Attendance, InternalMarks).
- **Business Logic:** Methods and functions that enforce business rules (e.g., calculating attendance percentages, checking prerequisites for course enrollment).
- **Data Access Objects (DAOs) or Repositories:** Classes that handle communication with the database, often using an ORM framework to map between database tables and entity classes.

View

Purpose: The View component is responsible for displaying the data provided by the Model in a specific format. It presents data to the user and sends user commands to the Controller.

Components:

- **Templates:** HTML/CSS/JS files or other templating engines (e.g., Thymeleaf, JSP, EJS) that define the user interface.
- **UI Frameworks:** Libraries like Bootstrap for styling and making the interface responsive.
- **Client-side Scripts:** JavaScript or frameworks like React, Angular, or Vue.js for dynamic content and interaction.

Controller

Purpose: The Controller component acts as an intermediary between the Model and the View. It listens to the input from the View, processes it (often making calls to the Model), and returns the data to the View in the appropriate format.

Components:

- **Routing:** Defines the routes for handling requests (e.g., Spring MVC, Express.js, ASP.NET Core).
- **Action Methods**: Methods that handle user actions, process data, and determine the appropriate view to render.

MVC Flow in College ERP System:

1. **User Requests:**
   - A student accesses the ERP system via a web browser and requests to view their attendance records.
   - The request is sent to the Controller.
2. **Controller Handling:**
   - The Controller receives the request and determines the necessary action.
   - It calls the appropriate method in the Model (e.g.,studentService.getAttendanceRecords(studentID)).
3. **Model Processing:**
   - The Model retrieves the required data from the database using DAOs or Repositories.
   - It applies any business logic (e.g., calculating total attendance percentage).
4. **Data Returned to Controller:**
   - The Model returns the processed data to the Controller.
5. **Controller Updates View:**
   - The Controller passes the data to the View.
   - It determines the appropriate view template to render (e.g., studentAttendance.html).
6. **View Rendering:**
   - The View renders the data into a user-friendly format (HTML/CSS).
   - It may use client-side scripting to enhance interactivity (e.g., using Vue.js to dynamically update attendance records).
7. **User Interaction with View:**
   - The user interacts with the rendered view (e.g., viewing detailed attendance records).
   - Any further actions by the user (e.g., filtering attendance by date) are sent back to the Controller, and the cycle repeats.

COMPONENT DESCRIPTION

User Management

Purpose: Handles the creation, management, and authentication of users within the system.

Components:

- **User Registration:** Allows new users (students, teachers, administrators) to register with the system.
- **Authentication:** Manages login and logout processes, ensuring only authorized users can access the system.
- **Role-Based Access Control (RBAC**): Assigns roles to users (e.g., student, teacher, admin) and controls access to different system functionalities based on these roles.
- **User Profiles:** Stores and manages user-specific information, such as personal details, contact information, and preferences.

Academic Management

Purpose: Manages academic activities, including course management, class schedules, and curriculum planning.

Components:

- **Course Management:** Allows creation, modification, and deletion of courses, including details like course ID, name, description, and prerequisites.
- **Class Scheduling:** Manages the timetable for classes, including assigning teachers, rooms, and time slots.
- **Curriculum Planning:** Helps in designing and updating academic programs and curricula.

Student Information System (SIS)

Purpose: Manages student-related data and processes, from enrollment to graduation.

Components:

- **Student Enrollment:** Handles the admission process, including application submission, verification, and enrollment.
- **Student Records:** Maintains detailed records of students, including personal information, academic history, and performance.

- **Attendance Management:** Tracks and manages student attendance, including recording attendance, generating reports, and sending notifications for irregularities.

Examination and Grading

Purpose: Manages the examination process and grading of students.

Components:

- **Exam Scheduling:** Plans and schedules exams, including setting exam dates, times, and locations.
- **Question Bank:** Maintains a repository of exam questions, categorized by subject and difficulty level.
- **Grading System:** Handles the grading process, including grade entry, calculation of final grades, and generation of report cards.
- **Result Publication:** Manages the publication of exam results, allowing students to view their performance online.

Attendance Management

Purpose: Manages and tracks attendance for students and faculty.

Components:

- **Attendance Recording:** Allows teachers to record attendance for their classes.
- **Attendance Reports:** Generates reports on student attendance, including daily, weekly, and monthly summaries.
- **Notifications:** Sends alerts and notifications for irregular attendance to students, parents, and administrators.

Internal Marks Management

Purpose: Manages the recording and analysis of internal assessments and marks.

Components:

- **Marks Entry:** Allows teachers to enter marks for assignments, quizzes, and internal exams.
- **Marks Analysis:** Analyzes internal marks to provide insights into student performance and progress.Gradebook: Maintains a gradebook for each student, tracking all internal marks and assessments.

Library Management

Purpose: Manages library resources and operations.

Components:

- **Catalog Management:** Maintains a catalog of all library resources, including books, journals, and digital media.
- **Circulation:** Manages the lending process, including issuing and returning of books, and tracking due dates.
- **Inventory Management:** Handles the acquisition, cataloging, and maintenance of library resources.
- **Member Management:** Manages library memberships, including registration of new members and tracking of borrowing history.

Finance and Accounts

Purpose: Manages financial operations, including fee management and accounting.

Components:

- **Fee Management:** Manages the collection of fees, including tuition, hostel, and other fees.
- **Billing and Invoicing:** Generates bills and invoices for various financial transactions. Accounting Manages financial records, including income, expenses, and budgeting.
- **Payroll:** Handles payroll processes for staff, including salary calculations, deductions, and payments.

Human Resource Management (HRM)

Purpose: Manages HR activities, including staff recruitment, attendance, and payroll.

Components:

- **Staff Recruitment:** Manages the recruitment process, including job postings, applications, and hiring.
- **Staff Attendance:** Tracks and manages staff attendance, including leave management.
- **Payroll Management:** Manages payroll processes for staff, including salary calculations, deductions, and payments.

Communication and Collaboration

Purpose: Facilitates communication and collaboration among students, teachers, and administrators.

Components:

- **Messaging System:** Allows users to send and receive messages within the system.
- **Announcements:** Manages the creation and distribution of announcements and notifications.
- **Forums and Discussion Boards:** Provides a platform for users to engage in discussions and share information.
- **Calendar:** Maintains a calendar for scheduling events, meetings, and deadlines.

Reporting and Analytics

Purpose: Provides insights and data analysis to support decision-making.

Components:

- **Report Generation:** Generates various reports on academic performance, attendance, finance, and more.
- **Dashboards:** Provides visual representations of key metrics and performance indicators.
- **Data Analytics:** Analyzes data to provide insights into trends, patterns, and anomalies.

System Administration

Purpose: Manages the overall system configuration, maintenance, and security.

Components:

- **User Management:** Handles the creation and management of user accounts and roles.
- **System Configuration:** Manages system settings, preferences, and configurations.
- **Security Management:** Ensures the security of the system, including data encryption, access control, and regular security audits.
- **Backup and Recovery:** Manages data backup and recovery processes to ensure data integrity and availability.

Integration and Communication

Purpose: Ensures seamless communication and integration between different components and external systems.

Components:

- **APIs and Web Services:** RESTful APIs or SOAP services to facilitate communication between components and with external systems.
- **Middleware:** Message brokers (RabbitMQ, Kafka) for asynchronous communication and integration.
- **Data Integration:** Tools and processes for integrating data from various sources, ensuring data consistency and accuracy.

Example Workflow:

Student Enrollment

- **User Interaction:**

A prospective student visits the college ERP system website and fills out an application form. The form data is submitted to the application layer.

- **Application Layer Processing:**

The controller receives the application data and invokes the business logic for validating the application. The system checks for required fields, eligibility criteria, and other business rules.

- **Data Access Layer:**

The validated application data is saved to the database using the data access layer (DAOs or Repositories).The system generates an application ID and stores it with the student's details.

- **Notification:**

An email notification is sent to the student confirming the submission of the application. The admission team is notified of the new application for further processing.

2.2 LANGUAGES

Academic Ease Management system uses a variety of programming languages to tie together many different functions of a college or university into one system.

2.2.1 SQL

SQL (Structured Query Language) is a language specifically designed for interacting with relational databases. In Academic ease management system, the database would hold a vast amount of information about students, faculty, courses, finances, and more. SQL allows the system to:

- **Retrieve data:** Find and extract specific information from the database. For example, finding all students enrolled in a particular course or retrieving a faculty member's contact details.
- **Insert data:** Add new information to the database. This could involve adding new students, registering them for courses, or recording financial transactions.
- **Update data:** Modify existing information in the database. This might involve updating student grades, changing faculty contact information, or reflecting changes in course schedules.
- **Delete data:** Remove outdated or irrelevant information from the database. This could be old student records, archived courses, or past financial data (depending on retention policies).

2.2.2 PYTHON

Python is a general-purpose, high-level programming language commonly used in the back-end development of web applications. In Academic ease management system built with Python, it would likely handle tasks like:

- **Business logic:** Python code would define the core functionalities of the system, such as processing student registrations, calculating fees, managing course enrollment, and generating reports.
- **Data interaction:** Python libraries would connect to the database using SQL and execute queries to retrieve, insert, update, or delete data as needed by the various functionalities of the system.
- **User Interface (UI) interaction:** While Python isn't typically used for the core UI development (HTML, CSS, JavaScript), it might be used to handle interactions between the UI and the back-end logic. For instance, processing user input from forms or sending data to be displayed on the user interface.

CHAPTER 3
REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENT SPECIFICATION

USER STORY: STUDENT REGISTRATION

Description: As an administrator, I want to register new students into the system so that their information can be managed and accessed efficiently.

Acceptance Criteria:

- A form for entering student details (name, department, etc.)
- Unique StudentID generated for each new student
- Confirmation message upon successful registration
- Validation to prevent duplicate StudentIDs

Priority: High

USER STORY: TEACHER REGISTRATION

Description: As an administrator, I want to register new teachers into the system so that their information can be managed and accessed efficiently.

Acceptance Criteria:

- A form for entering teacher details (name, department, etc.)
- Unique TeacherID generated for each new teacher
- Confirmation message upon successful registration
- Validation to prevent duplicate TeacherIDs

Priority: High

USER STORY: COURSE MANAGEMENT

Description: As an administrator, I want to add, update, and delete courses so that the course information remains up-to-date.

Acceptance Criteria:

- Ability to add new courses with CourseID, CourseName, and DepartmentID

- Ability to update course details
- Ability to delete courses
- Confirmation messages for each action

Priority: Medium

USER STORY: ATTENDANCE RECORDING

Description: As a teacher, I want to record student attendance for my courses so that attendance data is maintained accurately.

Acceptance Criteria:

- Ability to select course and date
- List of students enrolled in the selected course
- Option to mark each student as present or absent
- Confirmation message upon saving attendance

Priority: High

USER STORY: VIEW ATTENDANCE REPORT

Description: As a student, I want to view my attendance report so that I can keep track of my attendance status.

Acceptance Criteria:

- Ability to view attendance by course and date range
- Display of total classes attended and percentage attendance
- Option to download or print the attendance report

Priority: Medium

USER STORY: INTERNAL MARKS ENTRY

Description: As a teacher, I want to enter internal marks for my students so that their academic performance is recorded accurately.

Acceptance Criteria:

- Ability to select course and list of enrolled students

- Option to enter marks for each student
- Validation to ensure marks are within the acceptable range
- Confirmation message upon saving marks

Priority: High

USER STORY: VIEW INTERNAL MARKS

Description: As a student, I want to view my internal marks for each course so that I can monitor my academic progress.

Acceptance Criteria:

- Ability to view marks by course
- Display of total marks and individual assessments
- Option to download or print the marks report

Priority: Medium

USER STORY: DEPARTMENT MANAGEMENT

Description: As an administrator, I want to manage department details so that the department information remains accurate and up-to-date.

Acceptance Criteria:

- Ability to add, update, and delete department details
- Unique DepartmentID for each department
- Confirmation messages for each action
- Validation to prevent duplicate DepartmentIDs

Priority: Medium

NON-FUNCTIONAL REQUIREMENTS

❖ **Safety requirements**

If there is extensive damage to a wide portion of the database due to catastrophic failure, such as a disk crash, the recovery method restores a past copy of the database that was backed up to archival storage (typically tape) and reconstructs a more current state by reapplying or redoing the operations of committed transactions from the backed-up log, up to the time of failure.

❖ **Security requirements**

The database contains sensitive information of all the students and staff. Therefore, optimal security measures must be taken to ensure data is safe from unauthorized users.

❖ **Software Quality Attributes**

- **Availability:** The users must always be able to view their information so that they can keep track regularly.
- **Correctness:** The information about attendance and marks must be correct to not feed wrong information to the users.
- **Portability:** The users access the ERP from various platforms such as desktops and mobile phones. The webapp must be portable to all platforms and the user experience must be optimal.

❖ **System Design:** Various Design concepts and processes were applied to this project. Following concepts like separation of concerns, the software is divided into individual modules that are functionally independent and incorporates information hiding. The software is divided into 3 modules which are students, teachers and administrators. We shall look at each module in detail.

STUDENT

Each student belongs to a class identified by semester and section. Each class belongs to a department and are assigned a set of courses. Therefore, these courses are common to all students of that class. The students are given a unique username and password to login. Each of them will have a different view. These views are described below.

• **Student information** Each student can view only their own personal information. This includes their personal details like name, phone no, address etc. Also, they can view the courses they are enrolled in and the attendance, marks of each of those.

• **Attendance information** Attendance for each course will be displayed. This includes the number of attended classes and the attendance percentage. If the attendance percentage if below a specified threshold, say 75%, It will be marked in red otherwise it be in green. There will also be a day wise attendance view for each course which shows the date and status. This will be presented in a calender format.

• **Marks information** There will be 5 events and 1 semester end examination for each course. The marks for each of these will be provided in the ERP system.

• **Notifications and events** This section is common to all students. Notification are messages from the admin such as declaration of holidays, test time-table etc. The events and their details are specified here

TEACHER

Each teacher belongs to a department and are assigned to classes with a course. Teachers will also have a username and password to login. The different views for teachers are described below.

• **Information** The teachers will have access to information regarding the courses and classes they are assigned to. Details of the courses include the credits, the syllabus plan. Details of the class include the department, semester, section and the list of students in each class. The teacher will also have access to information of students who belong to the same class as as the teacher.

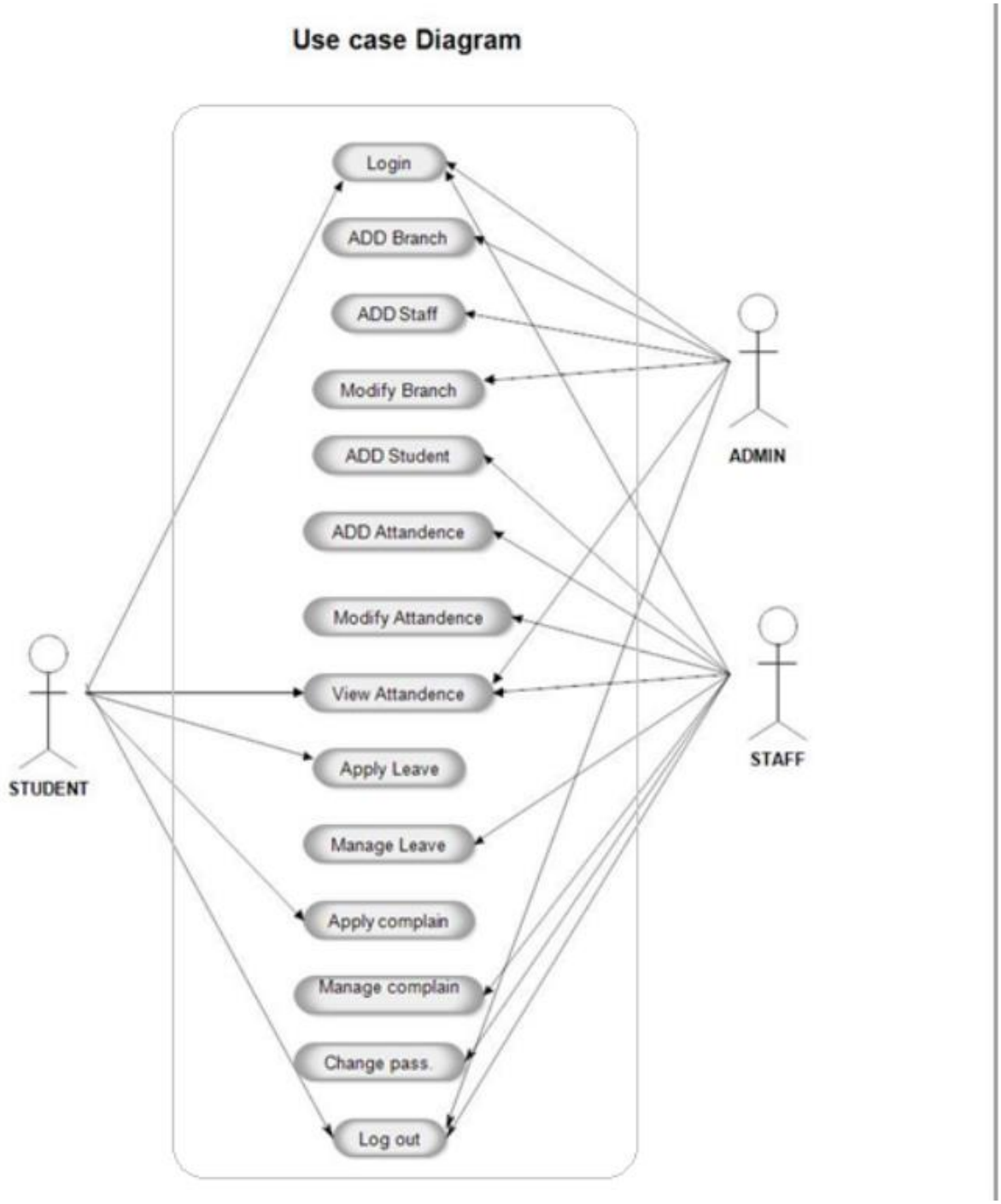
• **Attendance** The teacher has the ability to add also edit the attendance of each student. For entering the attendance, they will be given the list of students in each class and they can enter the attendance of the whole class on a day to day basis. There will be two radio buttons next to each student name, one for present and the other for absent. There will also be an option for extra classes. Teachers can edit the attendance of each student either for each student individually or for the whole class.

• **Marks** The teacher can enter the marks for the 5 events and 1 SEE for each course they are assigned. They also have the ability to edit the marks in case of any changes. Reports such as the report card including all the marks and CGPA of a student can be generated.

ADMINISTRATOR

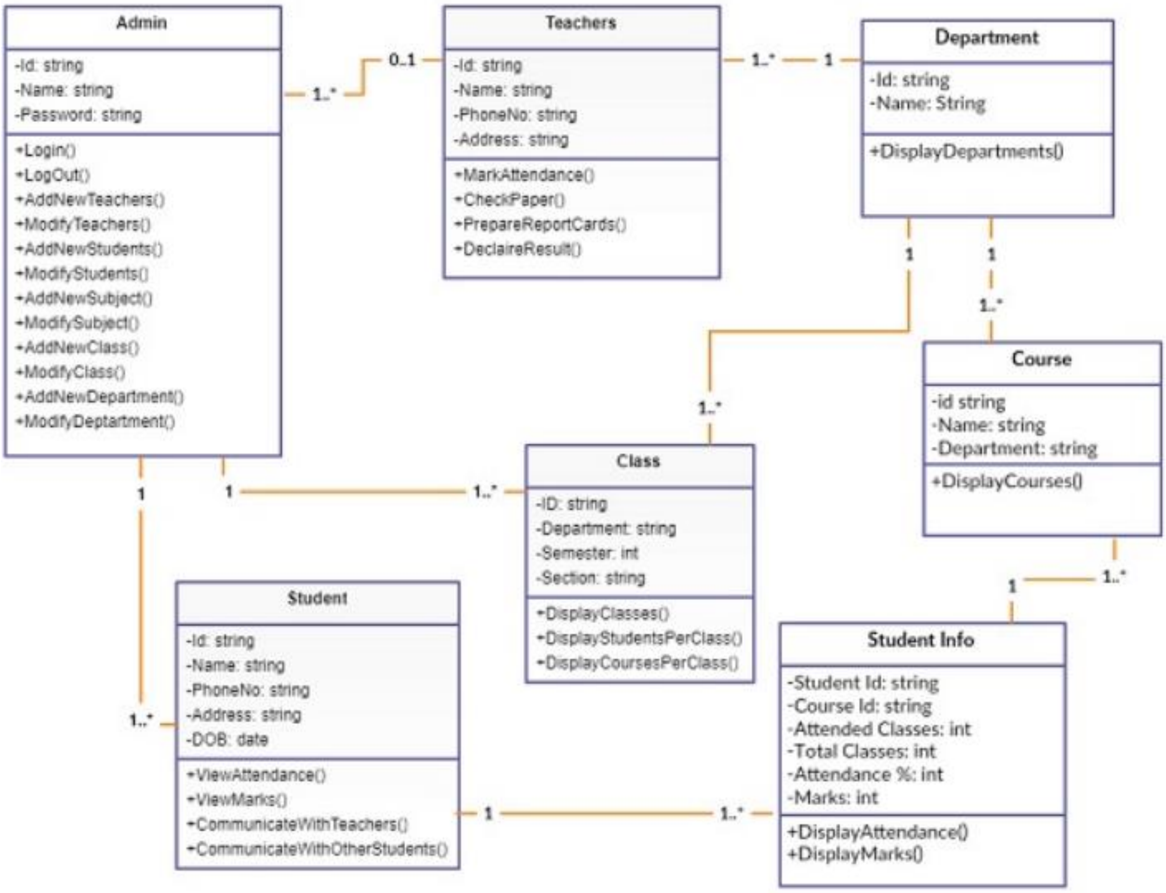
The administrator will have access to all the information in the different tables in the database. They will access to all the tables in a list form. They will be able to add a entry in any table and also edit them. The design of the view for the admin will provide a modular interface so that querying the tables will be optimized. They will be provided with search and filter features so that they can access data efficiently.

USE CASE DIAGRAM



CLASS DIAGRAM

The class diagram states the different classes involved in the software. For each class, a set of attributes and method are included. The relationship between the classes are also specified. For example, the teacher class has the attributes Id, name, phone no, address and methods such as marking attendance, declaring marks and preparing report cards. Each instance of the teacher class belongs to a department. This is specified by the relationship between Teacher and Department classes



MATHEMATICAL MODEL

Finite State Machine (FSM) for E-college ERP:

A change from one state to another when initiated by a triggering event or condition; this is called a transition. A particular FSM is defined by a list of its states, and the triggering condition for each transition.

As there are no mathematical calculations to be implemented in our project thus we have designed this FSM for our E- college. The Fig below shows the states and path which describes the flow E-College. It consists of M-set of tuples, Q-set of states, q0- Initial state, and F-final state.

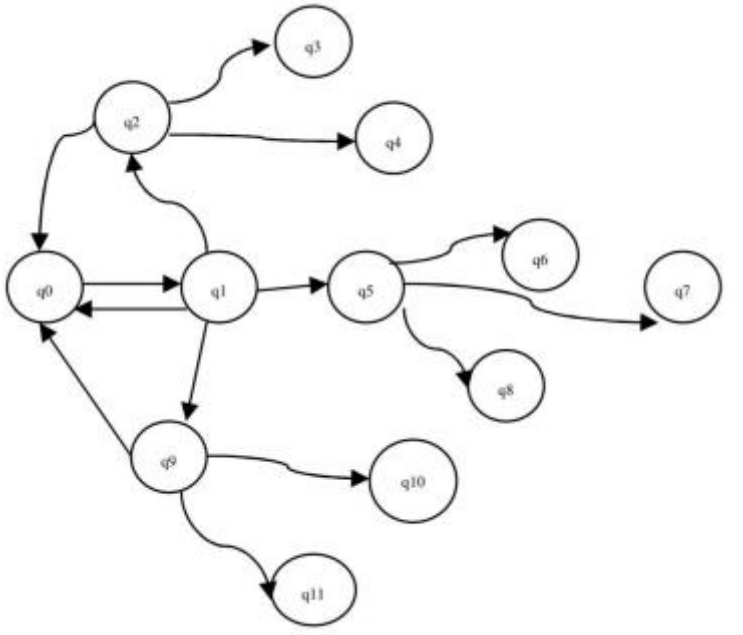
M= (Q, Σ, δ, q0, F)

Q:q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11

E: 1, 2, 3...16

q0: Homepage

F: Homepage



3.2 HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE REQUIREMENTS

1.Server

The application server handles the ERP system's operations, processing user requests, and running the application logic. A powerful processor and ample RAM ensure smooth performance, while SSD storage provides fast data access speeds. RAID configuration offers redundancy and data protection.

- **Processor:** Multi-core (minimum 4 cores), 3.0 GHz or higher
- **RAM:** 16 GB or higher
- **Storage:** SSD, minimum 1 TB with RAID configuration
- **Network Interface:** Gigabit Ethernet

2.Database Server

The database server stores and manages the ERP system's data. It requires a high-performance processor and significant RAM to handle large volumes of data and complex queries efficiently. SSD storage ensures quick data retrieval, and RAID configuration provides data redundancy.

- **Processor:** Multi-core (minimum 8 cores), 3.0 GHz or higher
- **RAM:** 32 GB or higher
- **Storage:** SSD, minimum 2 TB with RAID configuration
- **Network Interface**: Gigabit Ethernet

3. Network Infrastructure

- **Network Switches:**

Managed switches provide better control over the network, including traffic management, security features, and VLAN support, which are crucial for a large-scale ERP system.

- **Wireless Access Points:**

Reliable wireless connectivity is essential for mobile devices and laptops. Dual-band access points ensure better performance and reduced interference.

- **Router/Firewall:**

The router manages internet connectivity and network traffic. An integrated firewall provides security against external threats and ensures the safe operation of the ERP system.

- **Backup and Storage Solutions:**

Regular backups are crucial for data integrity and disaster recovery. NAS or SAN solutions provide scalable and reliable storage for backups and large data volumes.

4. Client Devices

- **Desktop Computers**

Desktop computers for students, teachers, and administrators should have sufficient processing power and memory to run the ERP client applications or web browsers smoothly.

  - Processor: Dual-core, 2.0 GHz or higher
  - RAM: 4 GB or higher
  - Storage: HDD/SSD, 250 GB or higher
  - Network Interface: Ethernet/Wi-Fi

- **Laptops:**

Laptops offer mobility for users who need to access the ERP system from different locations within the campus. They should have similar specifications to desktop computers.

  - Processor: Dual-core, 2.0 GHz or higher
  - RAM: 4 GB or higher
  - Storage: HDD/SSD, 250 GB or higher
  - Network Interface: Wi-Fi

- **Tablets/Smartphones:**

Tablets and smartphones provide additional flexibility for users to access the ERP system on the go. They should be capable of running the ERP mobile app or web interface efficiently.

  - Processor: Quad-core, 1.5 GHz or higher
  - RAM: 2 GB or higher
  - Storage: 16 GB or higher
  - Network Interface: Wi-Fi/4G

SOFTWARE REQUIREMENTS

1. Operating System

For the server: Linux distributions are often preferred for their stability, security, and cost-effectiveness, while Windows Server is chosen for its compatibility with certain enterprise applications.

For the Client devices: Client operating systems on desktops, laptops, and other devices used by students, teachers, and administrators should be compatible with the ERP client applications or web browsers.

2. Web Server

Apache and Nginx are popular open-source options known for their performance and flexibility, while IIS is integrated with Windows Server. Tomcat and JBoss are commonly used for Java-based applications, while .NET is used for applications developed in C# and other Microsoft technologies.

3. Programming Languages

Server-side scripting language such as PHP, Python which handles complex tasks and interacts with the database.
Client-side scripting languages like JavaScript for interactive features.

4. Database Management System (DBMS)

SQLite: The RDBMS stores and manages the ERP system's data, providing efficient data retrieval, transaction management, and security features. Choice of RDBMS depends on factors like licensing costs, scalability, and existing infrastructure.

5. Security Software

Firewalls protect the ERP system from unauthorized access and cyber threats by filtering incoming and outgoing network traffic based on security rules.

SSL/TLS certificates encrypt data transmitted between clients and servers, ensuring secure communication and protecting sensitive information.

Antivirus and anti-malware software protect the ERP system from malware infections and security breaches, ensuring the integrity and availability of the system.

6. Other Dependencies

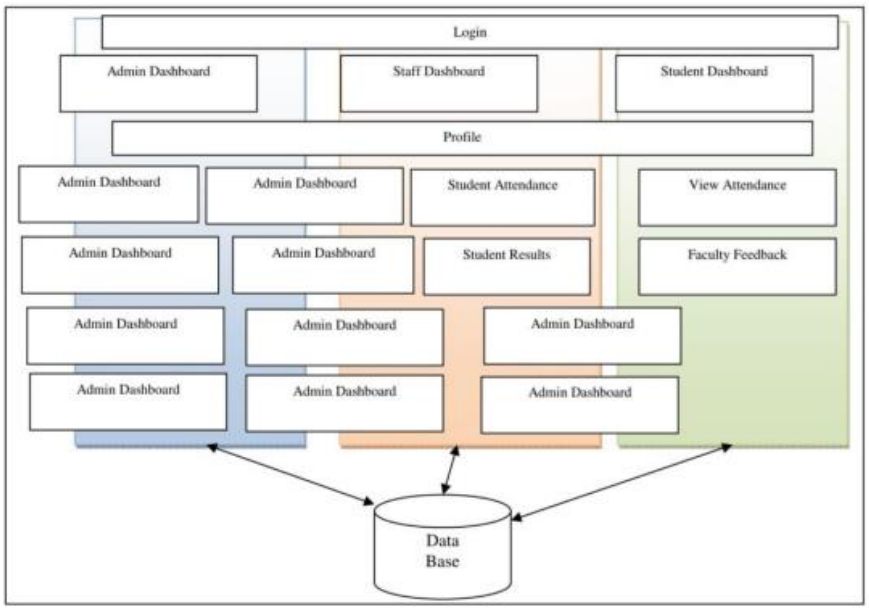
IDEs provide a comprehensive environment for developers to write, debug, and deploy ERP applications. They include code editors, debuggers, and build automation tools.

Version control systems manage source code changes, enabling collaboration among developers, tracking revisions, and maintaining code integrity. Git is widely used for its distributed nature and flexibility.

7. Monitoring and Management Tools

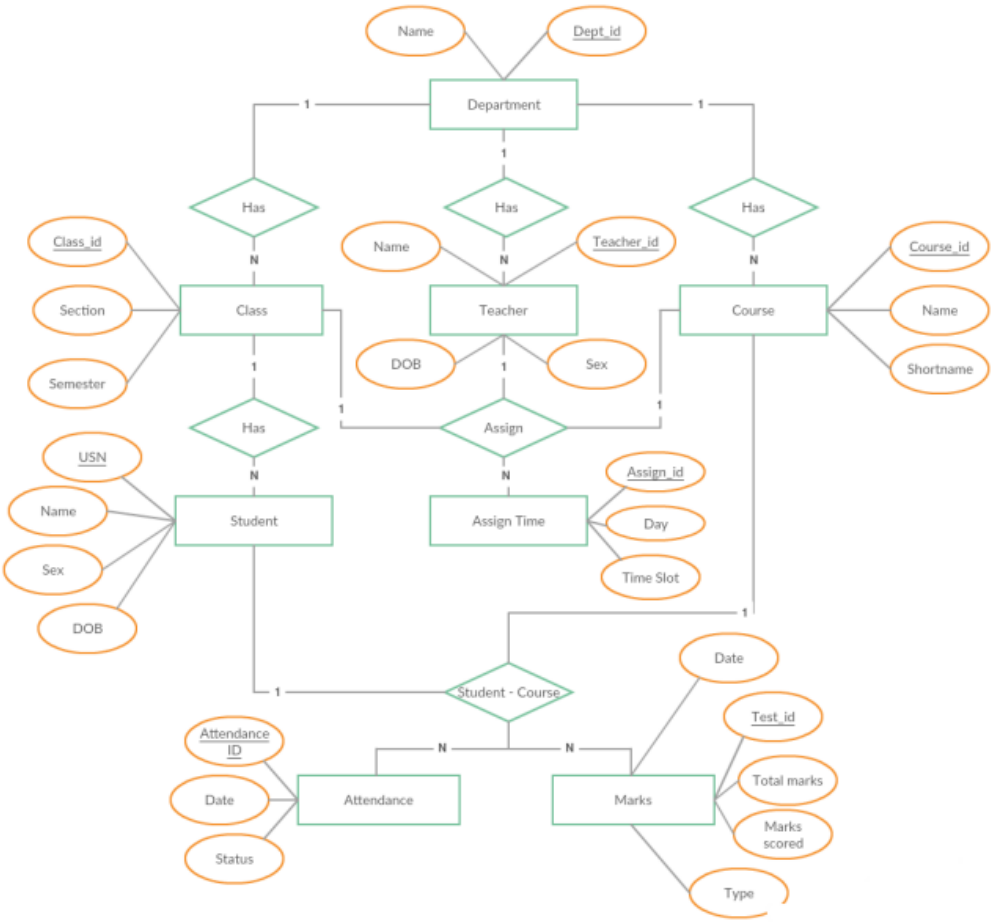
Monitoring tools like Nagios track the performance, availability, and health of the ERP system, providing alerts and reports to help administrators address issues promptly. Backup and recovery software like Veeam, Acronis, Bacula ensures regular backups of ERP data and system configurations, enabling quick restoration in case of data loss or system failure.

3.3 ARCHITECTURE DIAGRAM



Designing a detailed architecture for a College ERP system involves creating a multi-layered structure to ensure the system is scalable, maintainable, and secure. Here is a detailed description of the architecture consisting of presentation, application, data access, and database layers:

3.4 ER DIAGRAM



Entities and Attributes

- **Student:**
  - ✓ Attributes: StudentID (PK), Name, DepartmentID (FK)
  - ✓ Explanation: Represents the students enrolled in the college.
- **Teacher:**
  - ✓ Attributes: TeacherID (PK), Name, DepartmentID (FK)
  - ✓ Explanation: Represents the teachers who instruct courses.
- **Department:**
  - ✓ Attributes: DepartmentID (PK), DepartmentName
  - ✓ Explanation: Represents the various academic departments in the college.
- **Course:**
  - ✓ Attributes: CourseID (PK), CourseName, DepartmentID (FK)
  - ✓ Explanation: Represents the courses offered by the college.
- **Attendance:**
  - ✓ Attributes: AttendanceID (PK), StudentID (FK), CourseID (FK), Date, Status
  - ✓ Explanation: Represents the attendance records of students for specific courses.

- **InternalMarks:**
  - ✓ Attributes: MarksID (PK), StudentID (FK), CourseID (FK), Marks
  - ✓ Explanation: Represents the internal marks of students for specific courses.

Relationships

- **Enrollment:**
  - ✓ Entities Involved: Student, Course
  - ✓ Attributes: EnrollmentID (PK), StudentID (FK), CourseID (FK)
  - ✓ Explanation: Represents the enrollment of students in courses.
  - ✓ Cardinality: One student can enroll in multiple courses (1), and one course can have multiple students enrolled (N:1).
  - ✓ Participation: Total participation from the enrollment side (every enrollment must be linked to a student and a course).
- **Teaching:**
  - ✓ Entities Involved: Teacher, Course
  - ✓ Attributes: TeachingID (PK), TeacherID (FK), CourseID (FK)
  - ✓ Explanation: Represents the assignment of teachers to courses.
  - ✓ Cardinality: One teacher can teach multiple courses (1), and one course can be taught by multiple teachers (N:1).
  - ✓ Participation: Partial participation from the teacher side (not all teachers may teach) and total participation from the course side (every course must have a teacher).
- **Department-Course:**
  - ✓ Entities Involved: Department, Course
  - ✓ Explanation: Represents the relationship between departments and the courses they offer.
  - ✓ Cardinality: One department can offer multiple courses (1), but each course belongs to only one department (N:1).
  - ✓ Participation: Total participation from the course side (every course must belong to a department).
- **Department-Teacher:**
  - ✓ Entities Involved: Department, Teacher
  - ✓ Explanation: Represents the relationship between departments and their teachers.
  - ✓ Cardinality: One department can have multiple teachers (1), but each teacher belongs to only one department (N:1).
  - ✓ Participation: Total participation from the teacher side (every teacher must belong to a department).
- **Attendance Record:**
  - ✓ Entities Involved: Student, Course, Attendance
  - ✓ Explanation: Represents the attendance records of students for specific courses.

- ✓ Cardinality: One student can have multiple attendance records for different courses (1), and one course can have multiple attendance records for different students (N:1).
- ✓ Participation: Total participation from the attendance side (every attendance record must be linked to a student and a course).

- **Marks Record:**
  - ✓ Entities Involved: Student, Course, InternalMarks
  - ✓ Explanation: Represents the internal marks of students for specific courses.
  - ✓ Cardinality: One student can have multiple marks records for different courses (1), and one course can have multiple marks records for different students (N:1).
  - ✓ Participation: Total participation from the marks side (every marks record must be linked to a student and a course).

Explanation of Relationships, Cardinality, and Participation

- **Student - Enrollment:**
  - ✓ Cardinality: One student can enroll in multiple courses (1), and each enrollment entry must link to a student.
  - ✓ Participation: Total participation from the enrollment side.
- **Course - Enrollment:**
  - ✓ Cardinality: One course can have multiple students enrolled (N:1), and each enrollment entry must link to a course.
  - ✓ Participation: Total participation from the enrollment side.
- **Student - Attendance:**
  - ✓ Cardinality: One student can have multiple attendance records (1), and each attendance record must link to a student.
  - ✓ Participation: Total participation from the attendance side.
- **Course - Attendance:**
  - ✓ Cardinality: One course can have multiple attendance records (N:1), and each attendance record must link to a course.
  - ✓ Participation: Total participation from the attendance side.
- **Student - InternalMarks:**
  - ✓ Cardinality: One student can have multiple marks records (1), and each marks record must link to a student.
  - ✓ Participation: Total participation from the marks side.
- **Course - InternalMarks:**
  - ✓ Cardinality: One course can have multiple marks records (N:1), and each marks record must link to a course.
  - ✓ Participation: Total participation from the marks side.
- **Department - Course:**
  - ✓ Cardinality: One department can offer multiple courses (1), but each course belongs to only one department.
  - ✓ Participation: Total participation from the course side.

- **Department - Teacher:**
  - ✓ Cardinality: One department can have multiple teachers (1), but each teacher belongs to only one department.
  - ✓ Participation: Total participation from the teacher side.
- **Teacher - Teaching:**
  - ✓ Cardinality: One teacher can teach multiple courses (1), and each course can be taught by multiple teachers (N:1).
  - ✓ Participation: Partial participation from the teacher side, total participation from the course side.

3.5 NORMALISATION

1NF (First Normal Form)

In the first normal form, the table should have atomic values, and each column should contain only one value per record.

Teacher_id	Teacher_name	Student_id	Student_name	Student_marks
1	John	101	Alice	85
1	John	102	Bob	90
2	Jane	101	Alice	88
2	Jane	103	Charlie	92

2NF (Second Normal Form)

Eliminate partial dependencies. We break down the table further to ensure all non-key attributes depend on the entire primary key.

Teachers Table:

Teacher_id	Teacher_name
1	John
2	Jane

Students Table:

Student_id	Student_name
101	Alice
102	Bob
103	Charlie

Marks Table:

Teacher_id	Student_id	Student_marks
1	101	Alice
1	102	Bob
2	101	Alice
2	103	Charlie

3NF(Third Normal Form)

Eliminate transitive dependencies. Here, teacher_name depends only on teacher_id, student_name depends only on student_id, and marks depend on the combination of teacher_id, student_id, and subject.

The tables from 2NF already adhere to 3NF. We ensure there are no transitive dependencies in the design.

Teachers Table:

Teacher_id	Teacher_name
1	John
2	Jane

Students Table:

Student_id	Student_name
101	Alice
102	Bob
103	Charlie

Marks Table:

Teacher_id	Student_id	Student_marks
1	101	Alice
1	102	Bob
2	101	Alice
2	103	Charlie

CHAPTER 4

PROGRAM CODE

admin.py

```
from datetime import timedelta, datetime
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from django.http import HttpResponseRedirect
from django.urls import path
from .models import Dept, Class, Student, Attendance, Course, Teacher, Assign, AssignTime,
AttendanceClass
from .models import StudentCourse, Marks, User, AttendanceRange
# Register your models here.
days = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
    'Saturday': 6,
}
def daterange(start_date, end_date):
    for n in range(int((end_date - start_date).days)):
        yield start_date + timedelta(n)
class ClassInline(admin.TabularInline):
    model = Class
    extra = 0
class DeptAdmin(admin.ModelAdmin):
    inlines = [ClassInline]
    list_display = ('name', 'id')
    search_fields = ('name', 'id')
    ordering = ['name']
class StudentInline(admin.TabularInline):
```

```
    model = Student
    extra = 0
class ClassAdmin(admin.ModelAdmin):
    list_display = ('id', 'dept', 'sem', 'section')
    search_fields = ('id', 'dept__name', 'sem', 'section')
    ordering = ['dept__name', 'sem', 'section']
    inlines = [StudentInline]
class CourseAdmin(admin.ModelAdmin):
    list_display = ('id', 'name', 'dept')
    search_fields = ('id', 'name', 'dept__name')
    ordering = ['dept', 'id']
class AssignTimeInline(admin.TabularInline):
    model = AssignTime
    extra = 0
class AssignAdmin(admin.ModelAdmin):
    inlines = [AssignTimeInline]
    list_display = ('class_id', 'course', 'teacher')
    search_fields = ('class_id__dept__name', 'class_id__id', 'course__name', 'teacher__name',
'course__shortname')
    ordering = ['class_id__dept__name', 'class_id__id', 'course__id']
    raw_id_fields = ['class_id', 'course', 'teacher']
class MarksInline(admin.TabularInline):
    model = Marks
    extra = 0
class StudentCourseAdmin(admin.ModelAdmin):
    inlines = [MarksInline]
    list_display = ('student', 'course',)
    search_fields = ('student__name', 'course__name', 'student__class_id__id',
'student__class_id__dept__name')
    ordering = ('student__class_id__dept__name', 'student__class_id__id', 'student__USN')
class StudentAdmin(admin.ModelAdmin):
    list_display = ('USN', 'name', 'class_id')
    search_fields = ('USN', 'name', 'class_id__id', 'class_id__dept__name')
```

```
    ordering = ['class_id__dept__name', 'class_id__id', 'USN']
class TeacherAdmin(admin.ModelAdmin):
    list_display = ('name', 'dept')
    search_fields = ('name', 'dept__name')
    ordering = ['dept__name', 'name']
class AttendanceClassAdmin(admin.ModelAdmin):
    list_display = ('assign', 'date', 'status')
    ordering = ['assign', 'date']
    change_list_template = 'admin/attendance/attendance_change_list.html'
    def get_urls(self):
        urls = super().get_urls()
        my_urls = [
            path('reset_attd/', self.reset_attd, name='reset_attd'),
        ]
        return my_urls + urls
    def reset_attd(self, request):
        start_date = datetime.strptime(request.POST['startdate'], '%Y-%m-%d').date()
        end_date = datetime.strptime(request.POST['enddate'], '%Y-%m-%d').date()
        try:
            a = AttendanceRange.objects.all()[:1].get()
            a.start_date = start_date
            a.end_date = end_date
            a.save()
        except AttendanceRange.DoesNotExist:
            a = AttendanceRange(start_date=start_date, end_date=end_date)
            a.save()
        Attendance.objects.all().delete()
        AttendanceClass.objects.all().delete()
        for asst in AssignTime.objects.all():
            for single_date in daterange(start_date, end_date):
                if single_date.isoweekday() == days[asst.day]:
                    try:
```

```
                AttendanceClass.objects.get(date=single_date.strftime("%Y-%m-%d"),
assign=asst.assign)
            except AttendanceClass.DoesNotExist:
                a = AttendanceClass(date=single_date.strftime("%Y-%m-%d"), assign=asst.assign)
                a.save()
        self.message_user(request, "Attendance Dates reset successfully!")
        return HttpResponseRedirect("../")
admin.site.register(User, UserAdmin)
admin.site.register(Dept, DeptAdmin)
admin.site.register(Class, ClassAdmin)
admin.site.register(Student, StudentAdmin)
admin.site.register(Course, CourseAdmin)
admin.site.register(Teacher, TeacherAdmin)
admin.site.register(Assign, AssignAdmin)
admin.site.register(StudentCourse, StudentCourseAdmin)
admin.site.register(AttendanceClass, AttendanceClassAdmin)
```

apps.py

```
from django.apps import AppConfig
class InfoConfig(AppConfig):
    name = 'info'
```

models.py

```
from django.db import models
import math
from django.core.validators import MinValueValidator, MaxValueValidator
from django.contrib.auth.models import AbstractUser
from django.db.models.signals import post_save, post_delete
from datetime import timedelta
# Create your models here.
sex_choice = (
    ('Male', 'Male'),
```

```
    ('Female', 'Female')
)
time_slots = (
    ('7:30 - 8:30', '7:30 - 8:30'),
    ('8:30 - 9:30', '8:30 - 9:30'),
    ('9:30 - 10:30', '9:30 - 10:30'),
    ('11:00 - 11:50', '11:00 - 11:50'),
    ('11:50 - 12:40', '11:50 - 12:40'),
    ('12:40 - 1:30', '12:40 - 1:30'),
    ('2:30 - 3:30', '2:30 - 3:30'),
    ('3:30 - 4:30', '3:30 - 4:30'),
    ('4:30 - 5:30', '4:30 - 5:30'),
)
DAYS_OF_WEEK = (
    ('Monday', 'Monday'),
    ('Tuesday', 'Tuesday'),
    ('Wednesday', 'Wednesday'),
    ('Thursday', 'Thursday'),
    ('Friday', 'Friday'),
    ('Saturday', 'Saturday'),
)
test_name = (
    ('Internal test 1', 'Internal test 1'),
    ('Internal test 2', 'Internal test 2'),
    ('Internal test 3', 'Internal test 3'),
    ('Event 1', 'Event 1'),
    ('Event 2', 'Event 2'),
    ('Semester End Exam', 'Semester End Exam'),
)
class User(AbstractUser):
    @property
    def is_student(self):
```

```
        if hasattr(self, 'student'):
            return True
        return False
    @property
    def is_teacher(self):
        if hasattr(self, 'teacher'):
            return True
        return False
class Dept(models.Model):
    id = models.CharField(primary_key='True', max_length=100)
    name = models.CharField(max_length=200)
    def __str__(self):
        return self.name
class Course(models.Model):
    dept = models.ForeignKey(Dept, on_delete=models.CASCADE)
    id = models.CharField(primary_key='True', max_length=50)
    name = models.CharField(max_length=50)
    shortname = models.CharField(max_length=50, default='X')
    def __str__(self):
        return self.name
class Class(models.Model):
    # courses = models.ManyToManyField(Course, default=1)
    id = models.CharField(primary_key='True', max_length=100)
    dept = models.ForeignKey(Dept, on_delete=models.CASCADE)
    section = models.CharField(max_length=100)
    sem = models.IntegerField()
    class Meta:
        verbose_name_plural = 'classes'
    def __str__(self):
        d = Dept.objects.get(name=self.dept)
        return '%s : %d %s' % (d.name, self.sem, self.section)
class Student(models.Model):
```

```
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
    class_id = models.ForeignKey(Class, on_delete=models.CASCADE, default=1)
    USN = models.CharField(primary_key='True', max_length=100)
    name = models.CharField(max_length=200)
    sex = models.CharField(max_length=50, choices=sex_choice, default='Male')
    DOB = models.DateField(default='1998-01-01')
    def __str__(self):
        return self.name
class Teacher(models.Model):
    user = models.OneToOneField(User, on_delete=models.CASCADE, null=True)
    id = models.CharField(primary_key=True, max_length=100)
    dept = models.ForeignKey(Dept, on_delete=models.CASCADE, default=1)
    name = models.CharField(max_length=100)
    sex = models.CharField(max_length=50, choices=sex_choice, default='Male')
    DOB = models.DateField(default='1980-01-01')
    def __str__(self):
        return self.name
class Assign(models.Model):
    class_id = models.ForeignKey(Class, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    teacher = models.ForeignKey(Teacher, on_delete=models.CASCADE)
    class Meta:
        unique_together = (('course', 'class_id', 'teacher'),)
    def __str__(self):
        cl = Class.objects.get(id=self.class_id_id)
        cr = Course.objects.get(id=self.course_id)
        te = Teacher.objects.get(id=self.teacher_id)
        return '%s : %s : %s' % (te.name, cr.shortname, cl)
class AssignTime(models.Model):
    assign = models.ForeignKey(Assign, on_delete=models.CASCADE)
    period = models.CharField(max_length=50, choices=time_slots, default='11:00 - 11:50')
    day = models.CharField(max_length=15, choices=DAYS_OF_WEEK)
```

```
class AttendanceClass(models.Model):
    assign = models.ForeignKey(Assign, on_delete=models.CASCADE)
    date = models.DateField()
    status = models.IntegerField(default=0)
    class Meta:
        verbose_name = 'Attendance'
        verbose_name_plural = 'Attendance'
class Attendance(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    attendanceclass = models.ForeignKey(AttendanceClass, on_delete=models.CASCADE, default=1)
    date = models.DateField(default='2018-10-23')
    status = models.BooleanField(default='True')
   def __str__(self):
        sname = Student.objects.get(name=self.student)
        cname = Course.objects.get(name=self.course)
        return '%s : %s' % (sname.name, cname.shortname)
class AttendanceTotal(models.Model):
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    class Meta:
        unique_together = (('student', 'course'),)
     @property
    def att_class(self):
        stud = Student.objects.get(name=self.student)
        cr = Course.objects.get(name=self.course)
        att_class = Attendance.objects.filter(course=cr, student=stud, status='True').count()
        return att_class
    @property
    def total_class(self):
        stud = Student.objects.get(name=self.student)
        cr = Course.objects.get(name=self.course)
```

```
        total_class = Attendance.objects.filter(course=cr, student=stud).count()
        return total_class
    @property
    def attendance(self):
        stud = Student.objects.get(name=self.student)
        cr = Course.objects.get(name=self.course)
        total_class = Attendance.objects.filter(course=cr, student=stud).count()
        att_class = Attendance.objects.filter(course=cr, student=stud, status='True').count()
        if total_class == 0:
            attendance = 0
        else:
            attendance = round(att_class / total_class * 100, 2)
        return attendance
    @property
    def classes_to_attend(self):
        stud = Student.objects.get(name=self.student)
        cr = Course.objects.get(name=self.course)
        total_class = Attendance.objects.filter(course=cr, student=stud).count()
        att_class = Attendance.objects.filter(course=cr, student=stud, status='True').count()
        cta = math.ceil((0.75 * total_class - att_class) / 0.25)
        if cta < 0:
            return 0
        return cta
class StudentCourse(models.Model):
    student = models.ForeignKey(Student, on_delete=models.CASCADE)
    course = models.ForeignKey(Course, on_delete=models.CASCADE)
    class Meta:
        unique_together = (('student', 'course'),)
        verbose_name_plural = 'Marks'
    def __str__(self):
        sname = Student.objects.get(name=self.student)
        cname = Course.objects.get(name=self.course)
```

```
        return '%s : %s' % (sname.name, cname.shortname)

    def get_cie(self):
        marks_list = self.marks_set.all()
        m = []
        for mk in marks_list:
            m.append(mk.marks1)
        cie = math.ceil(sum(m[:5]) / 2)
        return cie

    def get_attendance(self):
        a = AttendanceTotal.objects.get(student=self.student, course=self.course)
        return a.attendance


class Marks(models.Model):
    studentcourse = models.ForeignKey(StudentCourse, on_delete=models.CASCADE)
    name = models.CharField(max_length=50, choices=test_name, default='Internal test 1')
    marks1=models.IntegerField(default=0,validators=[MinValueValidator(0),MaxValueValidator(100)])

    class Meta:
        unique_together = (('studentcourse', 'name'),)

    @property
    def total_marks(self):
        if self.name == 'Semester End Exam':
            return 100
        return 20


class MarksClass(models.Model):
    assign = models.ForeignKey(Assign, on_delete=models.CASCADE)
```

```
    name = models.CharField(max_length=50, choices=test_name, default='Internal test 1')
    status = models.BooleanField(default='False')

    class Meta:
        unique_together = (('assign', 'name'),)

    @property
    def total_marks(self):
        if self.name == 'Semester End Exam':
            return 100
        return 20


class AttendanceRange(models.Model):
    start_date = models.DateField()
    end_date = models.DateField()


# Triggers


def daterange(start_date, end_date):
    for n in range(int((end_date - start_date).days)):
        yield start_date + timedelta(n)


days = {
    'Monday': 1,
    'Tuesday': 2,
    'Wednesday': 3,
    'Thursday': 4,
    'Friday': 5,
```

```
    'Saturday': 6,
}


def create_attendance(sender, instance, **kwargs):
    if kwargs['created']:
        start_date = AttendanceRange.objects.all()[:1].get().start_date
        end_date = AttendanceRange.objects.all()[:1].get().end_date
        for single_date in daterange(start_date, end_date):
            if single_date.isoweekday() == days[instance.day]:
                try:
                    AttendanceClass.objects.get(date=single_date.strftime("%Y-%m-%d"),
assign=instance.assign)
                except AttendanceClass.DoesNotExist:
                    a = AttendanceClass(date=single_date.strftime("%Y-%m-%d"), assign=instance.assign)
                    a.save()


def create_marks(sender, instance, **kwargs):
    if kwargs['created']:
        if hasattr(instance, 'name'):
            ass_list = instance.class_id.assign_set.all()
            for ass in ass_list:
                try:
                    StudentCourse.objects.get(student=instance, course=ass.course)
                except StudentCourse.DoesNotExist:
                    sc = StudentCourse(student=instance, course=ass.course)
                    sc.save()
                    sc.marks_set.create(name='Internal test 1')
                    sc.marks_set.create(name='Internal test 2')
                    sc.marks_set.create(name='Internal test 3')
                    sc.marks_set.create(name='Event 1')
                    sc.marks_set.create(name='Event 2')
```

```
                sc.marks_set.create(name='Semester End Exam')
        elif hasattr(instance, 'course'):
            stud_list = instance.class_id.student_set.all()
            cr = instance.course
            for s in stud_list:
                try:
                    StudentCourse.objects.get(student=s, course=cr)
                except StudentCourse.DoesNotExist:
                    sc = StudentCourse(student=s, course=cr)
                    sc.save()
                    sc.marks_set.create(name='Internal test 1')
                    sc.marks_set.create(name='Internal test 2')
                    sc.marks_set.create(name='Internal test 3')
                    sc.marks_set.create(name='Event 1')
                    sc.marks_set.create(name='Event 2')
                    sc.marks_set.create(name='Semester End Exam')


def create_marks_class(sender, instance, **kwargs):
    if kwargs['created']:
        for name in test_name:
            try:
                MarksClass.objects.get(assign=instance, name=name[0])
            except MarksClass.DoesNotExist:
                m = MarksClass(assign=instance, name=name[0])
                m.save()


def delete_marks(sender, instance, **kwargs):
    stud_list = instance.class_id.student_set.all()
    StudentCourse.objects.filter(course=instance.course, student__in=stud_list).delete()
```

```
post_save.connect(create_marks, sender=Student)
post_save.connect(create_marks, sender=Assign)
post_save.connect(create_marks_class, sender=Assign)
post_save.connect(create_attendance, sender=AssignTime)
post_delete.connect(delete_marks, sender=Assign)
```

tests.py

```
from django.test import TestCase
from info.models import Dept, Class, Course, User, Student, Teacher, Assign, AssignTime, AttendanceTotal, Attendance, StudentCourse, Marks, MarksClass
from django.urls import reverse
from django.test.client import Client


# Create your tests here.


class InfoTest(TestCase):

    def create_user(self, username='testuser', password='project123'):
        self.client = Client()
        return User.objects.create(username=username, password=password)

    def test_user_creation(self):
        us = self.create_user()
        ut = self.create_user(username='teacher')
        s = Student(user=us, USN='CS01', name='test')
        s.save()
        t = Teacher(user=ut, id='CS01', name='test')
        t.save()
        self.assertTrue(isinstance(us, User))
```

```
        self.assertEqual(us.is_student, hasattr(us, 'student'))
        self.assertEqual(ut.is_teacher, hasattr(ut, 'teacher'))

    def create_dept(self, id='CS', name='CS'):
        return Dept.objects.create(id=id, name=name)

    def test_dept_creation(self):
        d = self.create_dept()
        self.assertTrue(isinstance(d, Dept))
        self.assertEqual(d.__str__(), d.name)

    def create_class(self, id='CS5A', sem=5, section='A'):
        dept = self.create_dept()
        return Class.objects.create(id=id, dept=dept, sem=sem, section=section)

    def test_class_creation(self):
        c = self.create_class()
        self.assertTrue(isinstance(c, Class))
        self.assertEqual(c.__str__(), "%s : %d %s" % (c.dept.name, c.sem, c.section))

    def create_course(self, id='CS510', name='Data Struct', shortname='DS'):
        dept = self.create_dept(id='CS2')
        return Course.objects.create(id=id, dept=dept, name=name, shortname=shortname)

    def test_course_creation(self):
        c = self.create_course()
        self.assertTrue(isinstance(c, Course))
        self.assertEqual(c.__str__(), c.name)

    def create_student(self, usn='CS01', name='samarth'):
        cl = self.create_class()
        u = self.create_user()
```

```
    return Student.objects.create(user=u, class_id=cl, USN=usn, name=name)

def test_student_creation(self):
    s = self.create_student()
    self.assertTrue(isinstance(s, Student))
    self.assertEqual(s.__str__(), s.name)

def create_teacher(self, id='CS01', name='teacher'):
    dept = self.create_dept(id='CS3')
    return Teacher.objects.create(id=id, name=name, dept=dept)

def test_teacher_creation(self):
    s = self.create_teacher()
    self.assertTrue(isinstance(s, Teacher))
    self.assertEqual(s.__str__(), s.name)

def create_assign(self):
    cl = self.create_class()
    cr = self.create_course()
    t = self.create_teacher()
    return Assign.objects.create(class_id=cl, course=cr, teacher=t)

def test_assign_creation(self):
    a = self.create_assign()
    self.assertTrue(isinstance(a, Assign))

# views
def setUp(self):
    self.client = Client()
    self.user = User.objects.create_user('test_user', 'test@test.com', 'test_password')

def test_index_admin(self):
```

```
        self.client.login(username='test_user', password='test_password')
        response = self.client.get(reverse('index'))
        self.assertContains(response, "you have been logged out")
        self.assertEqual(response.status_code, 200)

    def test_index_student(self):
        self.client.login(username='test_user', password='test_password')
        s = Student.objects.create(user=User.objects.first(), USN='test', name='test_name')
        response = self.client.get(reverse('index'))
        self.assertContains(response, s.name)
        self.assertEqual(response.status_code, 200)

    def test_index_teacher(self):
        self.client.login(username='test_user', password='test_password')
        s = Teacher.objects.create(user=User.objects.first(), id='test', name='test_name')
        response = self.client.get(reverse('index'))
        self.assertContains(response, s.name)
        self.assertEqual(response.status_code, 200)

    def test_no_attendance(self):
        s = self.create_student()
        self.client.login(username='test_user', password='test_password')
        response = self.client.get(reverse('attendance', args=(s.USN,)))
        self.assertContains(response, "student has no courses")
        self.assertEqual(response.status_code, 200)

    def test_attendance_view(self):
        s = self.create_student()
        self.client.login(username='test_user', password='test_password')
        Assign.objects.create(class_id=s.class_id, course=self.create_course(),
teacher=self.create_teacher())
        response = self.client.get(reverse('attendance', args=(s.USN,)))
        self.assertEqual(response.status_code, 200)
```

```
        self.assertQuerysetEqual(response.context['att_list'], ['<AttendanceTotal: AttendanceTotal object (1)>'])


    def test_no_attendance__detail(self):
        s = self.create_student()
        cr = self.create_course()
        self.client.login(username='test_user', password='test_password')
        resp = self.client.get(reverse('attendance_detail', args=(s.USN, cr.id)))
        self.assertEqual(resp.status_code, 200)
        self.assertContains(resp, "student has no attendance")


    def test_attendance__detail(self):
        s = self.create_student()
        cr = self.create_course()
        Attendance.objects.create(student=s, course=cr)
        self.client.login(username='test_user', password='test_password')
        resp = self.client.get(reverse('attendance_detail', args=(s.USN, cr.id)))
        self.assertEqual(resp.status_code, 200)
        self.assertQuerysetEqual(resp.context['att_list'], ['<Attendance: ' + s.name + ' : ' + cr.shortname + '>'])
```

manage.py

```
#!/usr/bin/env python
import os
import sys


if __name__ == '__main__':
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'CollegeERP.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
```

```
        "available on your PYTHONPATH environment variable? Did you "
        "forget to activate a virtual environment?"
    ) from exc
execute_from_command_line(sys.argv)
```

CHAPTER 5

RESULTS AND DISCUSSION

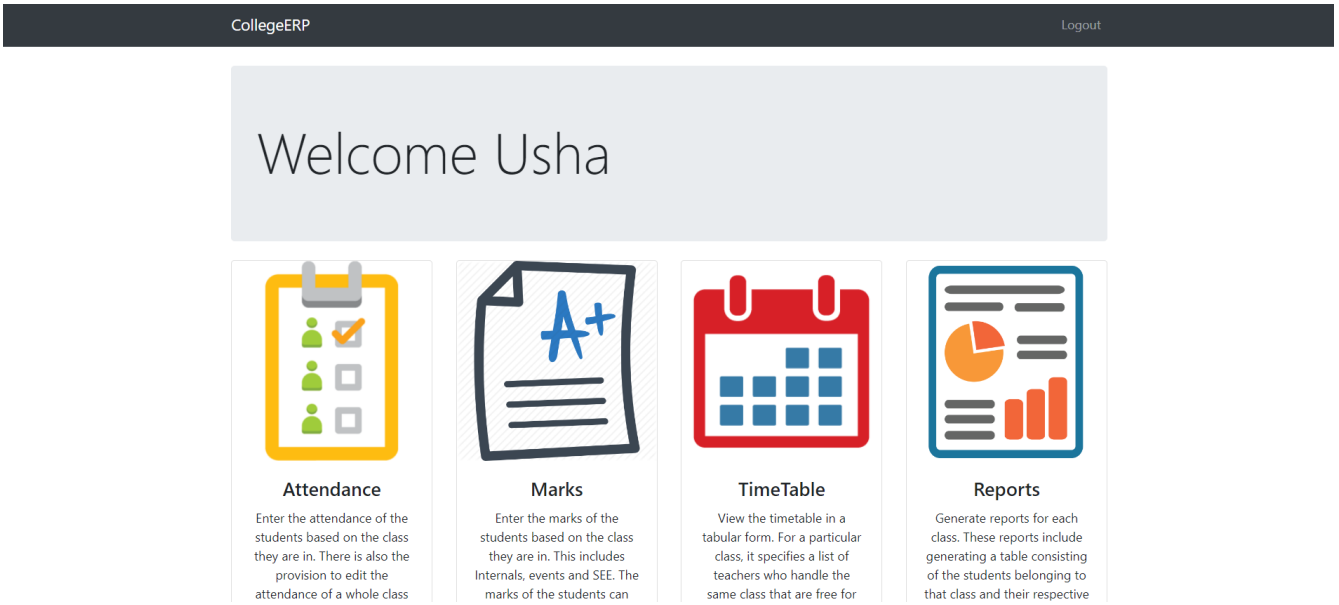


FIG 1. STAFF LOGIN

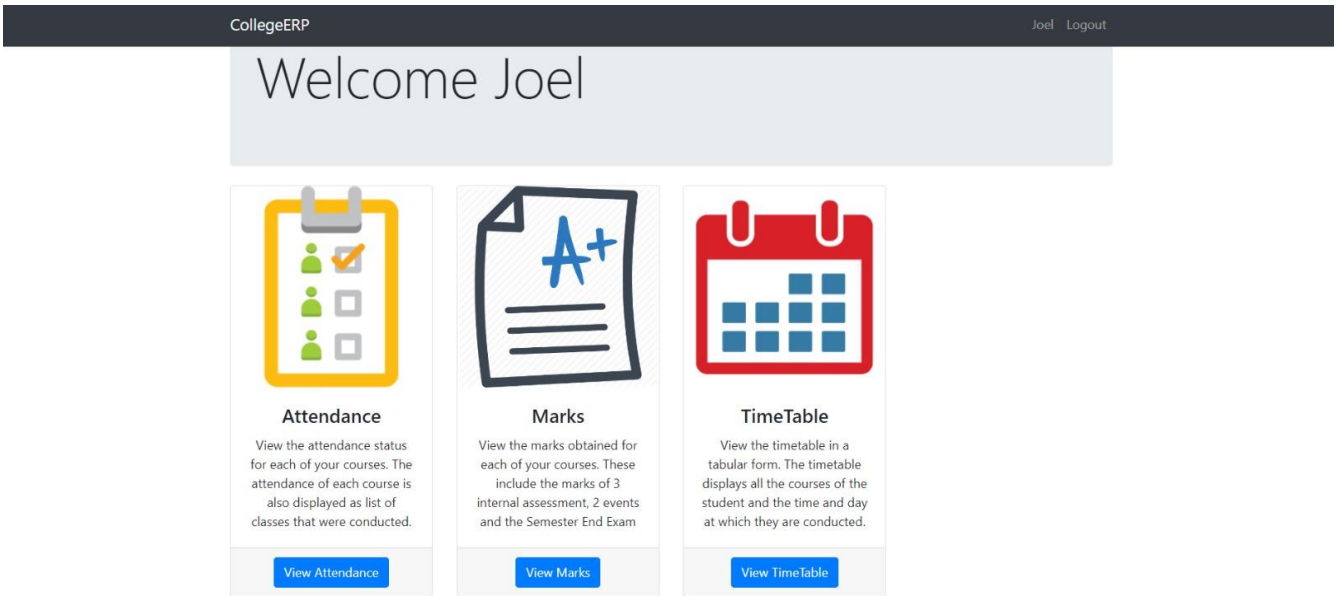


FIG 2. STUDENT LOGIN

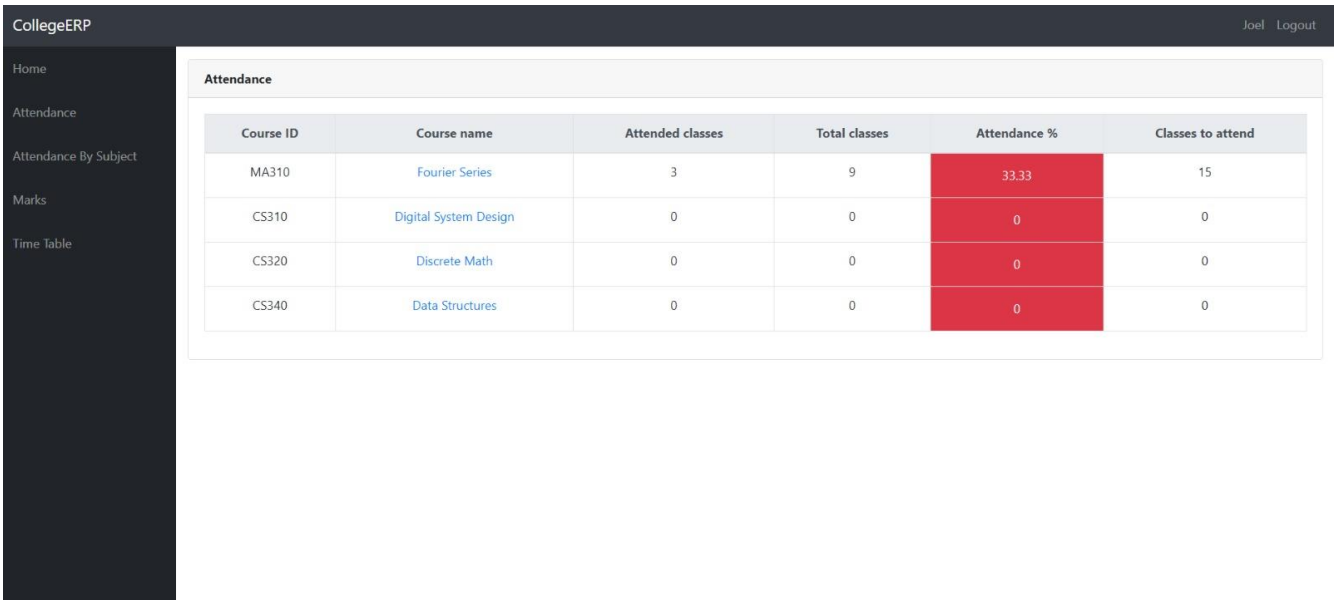


FIG 3. STUDENT ATTENDANCE VIEWING

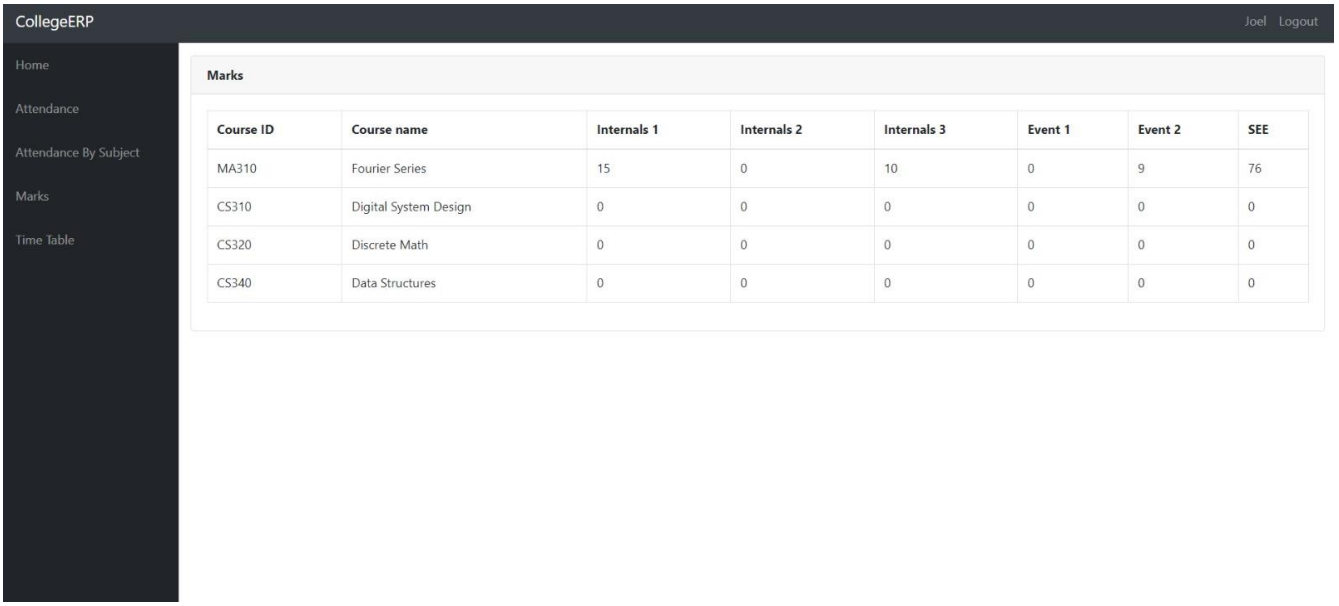


FIG 4. STUDENT MARK VIEWING

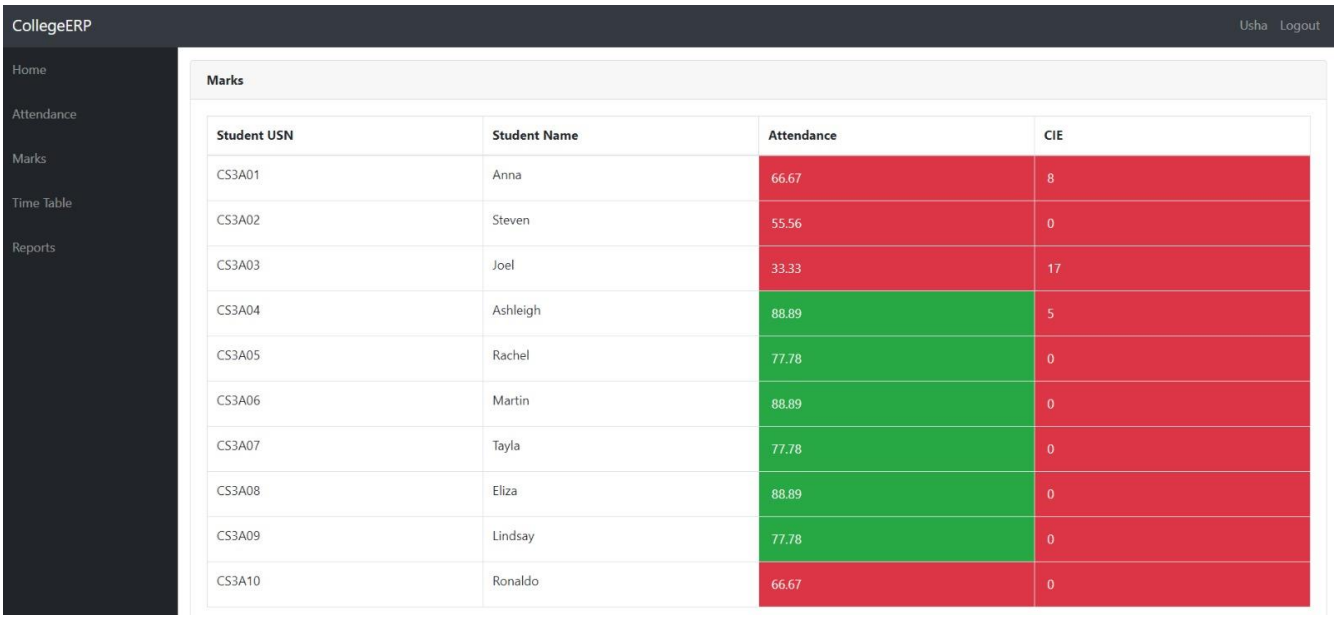


FIG 5.REPORT GENERATION

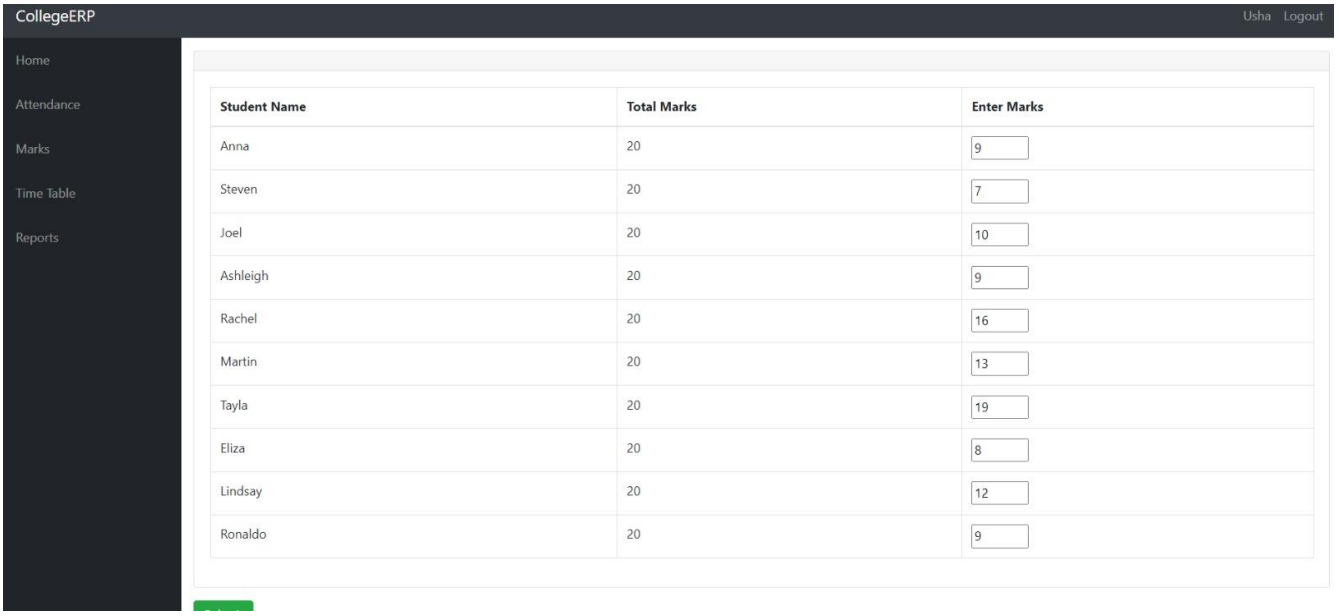


FIG 6. STAFF ENTERING INTERNAL MARKS

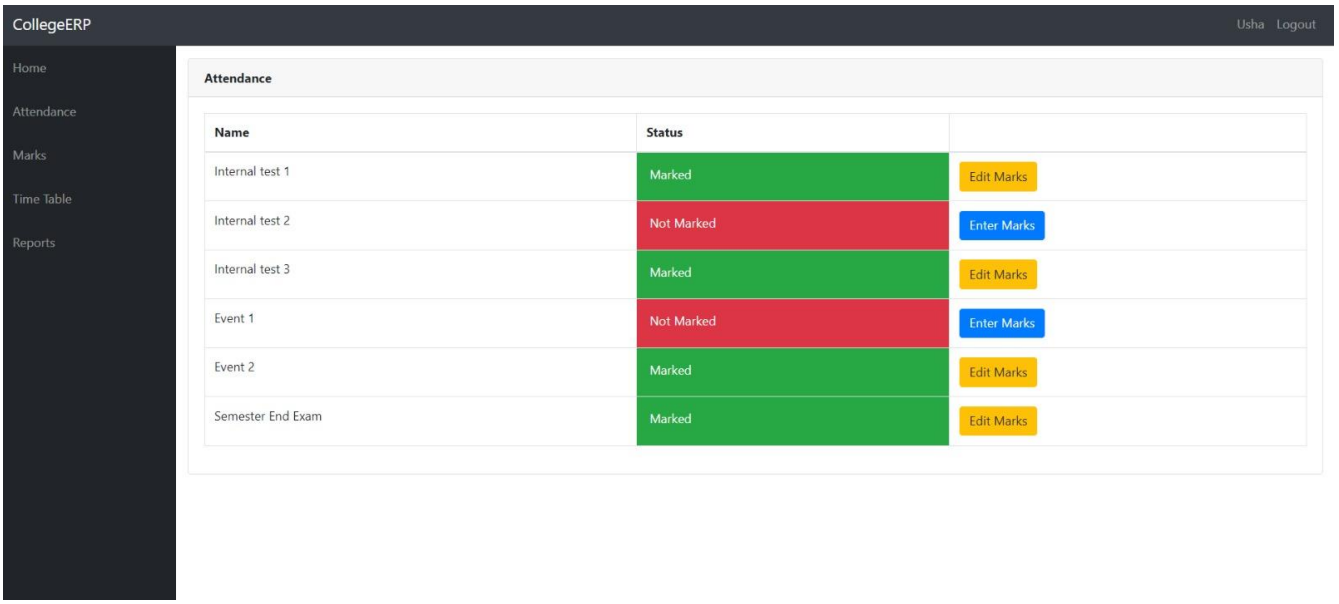


FIG 7.SUBMISSION AFTER INTERNAL MARK ENTERING

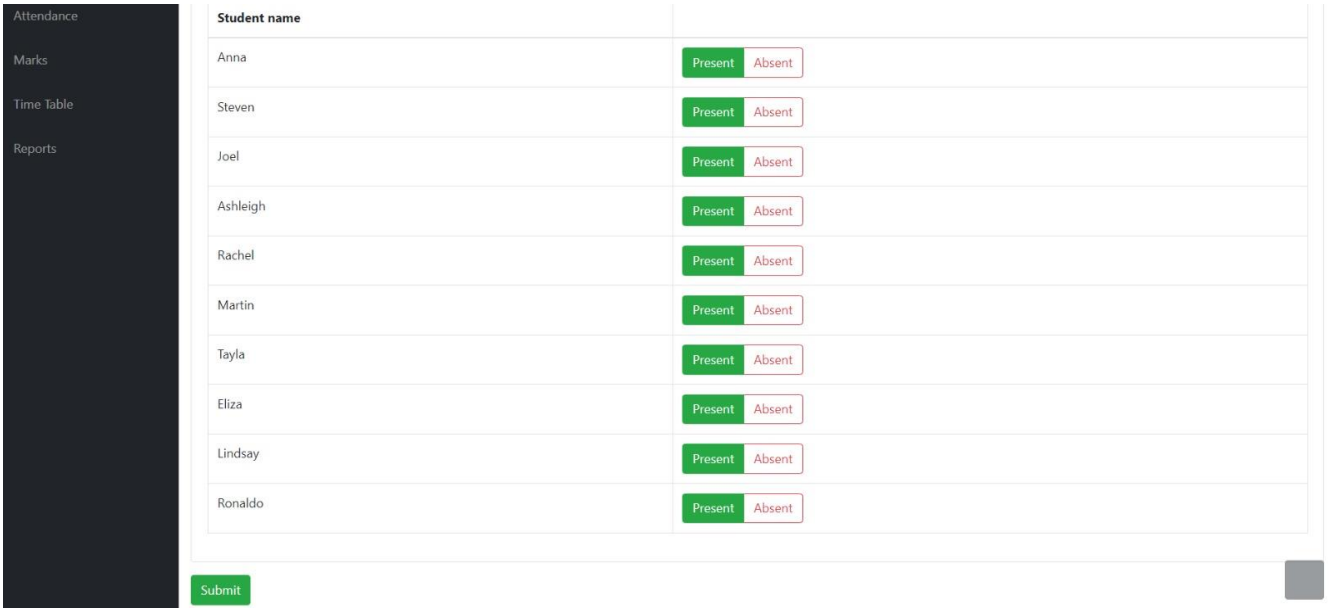


FIG 8. STAFF ENTERING ATTENDANCE

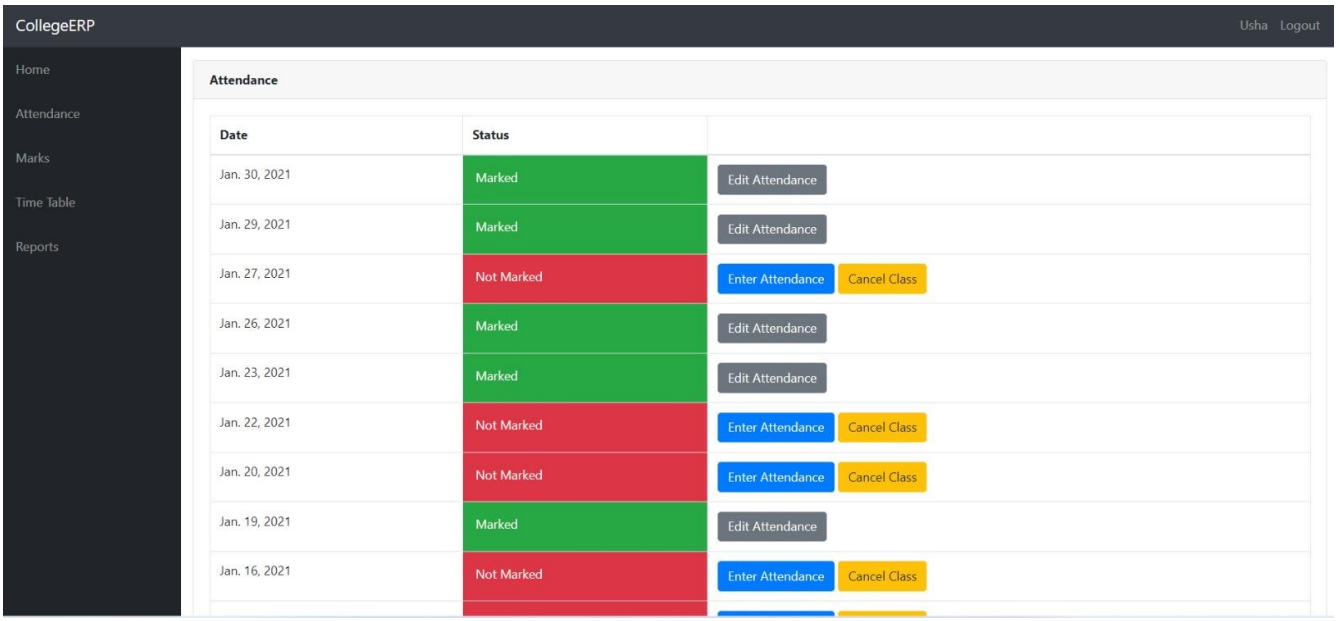


FIG 9. SUBMISSION AFTER ENTERING ATTENDANCE

The implementation of the Academic Ease Management System has resulted in significant improvements in efficiency, collaboration, and data management within the college community. Through automation of manual processes and integration of various modules, the system has streamlined administrative tasks and facilitated seamless communication among stakeholders. Comprehensive data management features have ensured accuracy and security of college records, while user-friendly interfaces have promoted widespread adoption and satisfaction. Real-time reporting functionalities have enhanced decision-making processes, while scalability and flexibility have enabled the system to adapt to evolving needs over time. Effective user training and support have further contributed to the success of the project, positioning the college for enhanced competitiveness and performance in the educational landscape.

CHAPTER 6

CONCLUSION

By using Existing System accessing information from files is a difficult task and there is no quick and easy way to keep the records of students and staff. Lack of automation is also there in the Existing System. The aim of Our System is to reduce the workload and to save significant staff time.

Title of the project as Academic Ease Management System is the system that deals with the issues related to a particular institution. It is the very useful to the student as well as the faculties to easy access to finding the details. The system provides appropriate information to users based on their profiles and role in the system. This project is designed keeping in view the day-to-day problems faced by a college system.

The fundamental problem in maintaining and managing the work by the administrator is hence overcome. Prior to this it was a bit difficult for maintaining the time table and keeping track of the daily schedule. But by developing this web-based application the administrator can enjoy the task, doing it ease and by saving the valuable time. The amount of time consumption is reduced and the manual calculations are omitted, the reports can be obtained regularly and whenever on demand by the user. The effective utilization of the work, by proper sharing it and by providing the accurate results. The storage facility will ease the job of the operator. Thus, the system developed will be helpful to the administrator by easing his/her task.

This System provide the automate admissions no manual processing is required. This is a paperless work. It can be monitored and controlled remotely. It reduces the manpower required. It provides accurate information always. All years together gathered information can be saved and can be accessed at any time. The data which is stored in the repository helps in taking intelligent decisions by the management providing the accurate results. The storage facility will ease the job of the operator. Thus, the system developed will be helpful to the administrator by easing his/her task providing the accurate results. The storage facility will ease the job of the operator.

This project is successfully implemented with all the features and modules of the college management system as per requirement

CHAPTER 8

REFERENCES

1. Elmasri and Navathe: Fundamentals of Database Systems, 7th Edition, Pearson Education, 2016, is a comprehensive textbook that covers the principles of database design, modeling, and implementation

2. Ian Sommerville: Software Engineering, 10th edition, Person Education Ltd, 2015, provides a comprehensive overview of software engineering principles and practices, emphasizing both theoretical foundations and practical applications.

3. Roger S Pressman: Software Engineering- A Practitioners approach,8th edition, McGraw-Hill Publication, 2015, provides a comprehensive guide to software engineering principles and practices.

4. https://en.wikipedia.org/wiki/Requirements-engineering provides information about management to ensure that the final system meets the stakeholders' needs and expectations.

5. https://web.cs.dal.ca/ hawkey/3130/srs-template-ieee.doc provides knowledge on development of an application.

6.http://www.ntu.edu.sg/home/cfcavallaro/Reports/Report%20writing.htmTop elucidate on report generating

7. https://en.wikipedia.org/wiki/Class diagram/ gives knowledge on construction of class diagram

8. https://www.djangoproject.com/ Django is a Python web framework known for its rapid development and clean, pragmatic design.

9. https://getbootstrap.com/ is a popular front-end framework for building responsive and mobile-first websites and web applications.

10. https://www.tutorialspoint.com/ is an online platform offering tutorials and resources covering a wide range of topics, including programming languages, web development, software testing, and more.

11. https://creately.com/ is a web-based diagramming tool that allows users to create various types of diagrams, including flowcharts, wireframes, UML diagrams, and mind maps, collaboratively and easily.

12. https://www.overleaf.com/project , is an online collaborative LaTeX editor used for creating and editing documents in LaTeX.