Acknowledgment

I take this opportunity to express my deep sense of gratitude to all those who have contributed significantly by sharing their knowledge and experience in the completion of this project work.

My first word of gratitude is to **Dr. Sushma Bahuguna,** for providing this kind of opportunity and guidance throughout the project.

My heartfull thanks to My Project Guide **Dr. Sushma Bahuguna** his continuation helps the project would not have been materialized in the present form. I am thankful to him for providing me with necessary insights and valuable suggestions helped me at every step. We would like to take this opportunity to extend our warm thoughts to those who helped me in making this project a wonderful experience.

Last but not least; we would also like to thank our family and friends for their support an encouragement.

Name: Rajnish Kumar

Enroll No.: 2401877374

Date: _____

Index

ABSTRACT

This report specifies the colorful processes and ways used in gathering conditions, designing, enforcing and testing for the design on council operation system. The problems regarding the current system in the council was anatomized and noted. This design aims to break some of those problems and therefore, add further value to the current system. The conditions were gathered from all the stakeholders and grounded on that we created conditions models and designed the software grounded on the grounded. The design was enforced in the form of a website using Django (python). Using the colorful coffers and tools we gathered along the way, we enforced the council ERP system using some features that break the current problems in the system similar as a provision to edit the attendance and marks before locking it at the end. The software was also tested using the colorful testing styles and results were positive.

Therefore, the results can be integrated in the current ERP system to ameliorate its working and break some of the being problems.

3. INTRODUCTION

In today's fast-paced world, managing a college's resources and data can be an arduous task. Fortunately, the Next Generation College ERP System is here to revolutionize the way colleges operate. This system is a comprehensive solution that streamlines administrative tasks, enhances communication among students and faculty, and provides a seamless user experience. This project aims to design, develop, and implement this system in a college setting to enhance efficiency, productivity, and overall performance.

The College Information Management System aims to simplify the management and maintenance of personal student data. It provides an efficient means for administrators to edit and access the personal details of students, while allowing students to keep their profiles up-to-date. The system maintains records of students' information, such as their ID, name, mailing address, phone number, and date of birth, making it easy to retrieve information in just a few seconds.

This comprehensive project is designed to fulfill the requirements of colleges and streamline daily operations, from attendance management to communication among students and teachers. It is an integrated information management system that ensures data accuracy and keeps information up-to-date throughout the college. With a simple and intuitive user interface, the system is easy to learn and use, increasing users' productivity. Additionally, the system features efficient security measures that ensure data privacy.

A College ERP System is an integrated software solution designed to streamline and automate various administrative and academic processes within a college or university. It helps to manage and monitor different departments of an institution such as student admissions, course management, fee collection, library management, student performance tracking, faculty management, and many more.

As we know that, a college consists of different departments, such as course departments, fees management, library, event management etc. Nowadays applications and uses of information technologies is increased as compared to before, each of these individual departments has its own computer system to do their own functionalities. By having one main system they can interact with each other from their respected system by having valid user id and password.

3.1. OBJECTIVE

The objective of College Information Management System is to allow the administrator of any organization the ability to edit and find out the personal details of a student and allows the student to keep up to date his profile. It'll also facilitate keeping all the records of students, such as their id, name, mailing address, phone number, DOB etc. So, all the information about a student will be available in a few seconds. Overall, it'll make Student Information an easier job for the administrator and the student of any organization.

The primary aim of a College ERP System is to create a user-friendly web platform accessible to students, faculty, and staff. This system should streamline administrative tasks and enhance overall efficiency by centralizing college data and automating key processes. For students, the objective is to provide easy access to academic information, simplify enrollment and fee payment, and improve communication with the college. For faculty, the system should facilitate efficient management of courses, student records, and communication. Ultimately, the ERP system aims to empower administrators with comprehensive tools for managing all aspects of college operations, including student information, academics, finances, and resources, leading to better decision-making and a more effective educational environment.

The main purpose of this project is to illustrate the requirements of the project College Information Management System and is intended to help any organization to maintain and manage personal data. It is a comprehensive project developed from the ground up to fulfill the needs of colleges as they guide their students. This integrated information management system connects daily operations in the college environment ranging from Attendance management to communicational means among students and teachers. This reduces data error and ensures that information is always up-to-date throughout the college. It provides a single source of data repository for streamlining our processes and for all reporting purposes. It has a simple user interface and is intuitive. This ensures that the users spend less time in learning the system and hence, increase their productivity. Efficient security features provide data privacy and hence, increase their productivity.

4. SYSTEM ANALYSIS

4.1. Technology Analysis Study

The College ERP System effectively addresses the challenges of managing diverse institutional processes by integrating functions such as admissions, student records, attendance, fee management, and communication onto a centralized platform. It automates data entry and retrieval, reducing manual errors and administrative burden for staff. The system's secure, role-based access protects sensitive information while keeping data current and easily accessible. Enhanced collaboration among departments streamlines operations and improves productivity. The intuitive interface also ensures rapid user adoption, maximizing efficiency. Overall, the system offers colleges a scalable solution for optimizing resources and performance in today's technology-driven educational landscape.

Difference with previous technology current technology

The previous technology used in college ERP systems was often based on legacy software applications that were designed to perform specific tasks. These systems were often not designed with flexibility in mind, and they lacked the ability to adapt to changing needs and new technologies. Additionally, these legacy systems were often limited in terms of scalability, making it difficult to expand the system as the college grew.

The new technology used in college ERP systems is much more flexible and adaptable. It is designed to be scalable, which means it can easily accommodate an expanding college. This technology is often based on cloud computing, which allows for easy and quick updates to the system. It also provides greater security, as the data is stored in a centralized location with strict access controls. Additionally, the new technology is designed with mobility in mind, allowing users to access the system from anywhere, at any time, on any device.

Overall, the new technology used in college ERP systems is much more powerful and flexible than previous technologies. It allows for greater efficiency and productivity, as well as greater scalability and mobility.

Concern technology in market

In recent years, there has been a surge in the development of advanced technologies, including artificial intelligence, machine learning, big data, and cloud computing. These technologies have revolutionized the way institutions manage their operations, resulting in improved efficiency, accuracy, and speed. Cloud-based ERP systems have gained popularity due to their flexibility and ease of use.

They enable users to access and manage data from anywhere, at any time, using any device with an internet connection. Additionally, AI and machine learning algorithms can provide insights and recommendations for data-driven decision-making.

Application area

The application area for college ERP systems is vast and covers nearly every aspect of college management. This includes student enrollment and registration, financial aid management, academic scheduling, grading and transcript management, faculty and staff management, facilities management, and alumni relations.

One of the most important application areas of a college ERP system is student enrollment and registration. The system allows students to enroll in classes, register for courses, and manage their schedules online. It also allows administrators to manage student information, track student progress, and monitor student attendance.

Financial aid management is another critical application area for college ERP systems. The system allows administrators to manage student financial aid, including scholarships, grants, and loans. It also allows students to view their financial aid information and track their loan balances.

Academic scheduling is another important application area for college ERP systems. The system allows administrators to create schedules for classes, manage course offerings, and schedule faculty and staff. It also allows students to view class schedules and register for courses.

Overall, the application area for college ERP systems is vast and critical to the efficient and effective management of a college. The system provides a centralized location for managing critical college functions and ensures that information is accurate, up-to-date, and easily accessible.

4.2. Identification of Need

Why do we need ERP?

Currently, in seminaries and sodalities, it's veritably delicate to manage each and everything manually. Supervising and maintaining the whole database of an academy or council can be time- consuming and challenging especially if it's done on a regular base. So, we need to handle and manage everything dashingly.

To break this problem ERP (Enterprise Resource Planning) is used. ERP software takes it easy to track the progress of every department of academy and automate different functions. With ERP every- thing can be seen on a single dashboard. The director can manage the council from anywhere. The possibilities of maintaining the whole database of a council with ERP software are endless.

Some of the prominent places of ERP are: -

- Manages the office and automates different functions.
- Helps in long- term operation and planning of all departments of council.
- Eliminates the need for having multiple operation software for each department.
- Diurnal conditioning like attendance can be digitalized and automated.
- Leave module for preceptors can be automated.

Commencement

Commencement is a process of establishing an introductory understanding of the problem and the nature of the result. This includes the need for this software, identification of stakeholders and defining multiple shoes.

What's the purpose of this design?

There’s presently an ERP system in our council. But not everyone is happy with the system. While it's a step towards automating the council conditioning, it comes with its own set of problems. This design is designed to apply a council ERP system to annihilate some of these problems and add some features of our own that would add value to system.

Identification of stakeholders

Enterprise Resource planning perpetration is a delicate and complex decision where it involves people issues further than technological issues. Identification of stakeholders is a crucial step during the process of ERP perpetration, because if done inaptly, it'll lead to failure of the perpetration design. The stakeholders are listed below: -

Teachers

Teachers are the crucial stakeholders of the council ERP. Because they're the bone who manages, edit, modernize the contents of the database of scholars similar as attendance, internal marks, CGPA etc. It also helps them to assign their class to other preceptors when they're on leave. This makes it easier to identify who among them are free to take the class at that time. So, this software helps them reduce their outflow and make their tasks easier and simple.

Students

Scholars are end druggies of ERP software. The attendance, internals marks uploaded by the preceptors is viewed by scholars. It helps them track their attendance status. It also helps them to communicate with preceptors and their classmates. So, scholars make up another set of stakeholders of this software.

Administrator

Director council director is responsible for maintaining the database of the council. They will have the honor to modify the database i.e., to add remove scholars' preceptors' staff, update information regarding each of these. It's their responsibility to maintain the database of scholars who pass out from the council and who lately get admission to the council. So, the director plays a major part in the ERP.

Viewpoints

Teacher's viewpoint

Shoes preceptors' standpoint for a school teacher, this software must be easy to use. It should be easy to find different modules like attend- cotillion, leave module, internals marks, affect etc. Teachers are the bone who modernizes the contents of the database, so it should be update save modify it.

Student's viewpoint

Scholar's standpoint a pupil can only view the information about himself, other than that everything will be hidden from them. They won't have the option to edit anything. So, the graphical stoner interface must be good. They anticipate it to be functional.

Administrator's viewpoint

Director's standpoint director will have the honor to view all the information about the council. They will have the option to track pretensions like, Average marks of all the scholars in a subject, Average attendance of all the scholars of a class etc.

Elicitation

When we started the design, we decided to collect the information from a couple of stakeholders like preceptors, directors, scholars and parents. They stated their part in the ERP system, their problems, likes and dislikes problems they're facing with the software and how it's enforced.

Preceptors

We had an occasion to meet our council Computer Science Department. They gave us an idea about how our council ERP was working and explained about their part in the ERP. We asked the following questions:

Can we explain the attendance entry process in detail?

Generally, just like the scholars, indeed preceptors have their own stoner ID and word for the login purpose. There will be a column reserved for attendance purpose in a hierarchical manner. First there will be two columns class and subjects. Under the class column there will be a list of all the classes distributed to the faculty. On the other column there's subjects which is further divided into theoretical subjects who are of 4 credits and integrated subjects which are of 5 credits for the university batch scholars. Since there are independent batch scholars who are yet to complete their degree there are separate columns reserved for them since their pattern is different from the university syllabus. They will be having theoretical subjects of 4 credits each and they will also be having separate lab sessions of1.5 credits each. Since the credits of independent subjects vary from those of the subjects of the university subjects there must be changes in terms of attendance and the credits allocated for each subject.

Can we explain how operation for leave is managed in the ERP system?

Also, there will be a column for the type of class. In this there will be further two types. Regular classes and alternate classes. Regular classes are those which the faculty handles for the allocated class as specified in the time- table. Alternate classes are those which the faculty handles in the absence of another faculty. When the faculty is on leave, it must be informed in the ERP similar that the communication goes to the professors and at the same time another schoolteacher who's free. However, also they should inform the scholars in the forum about the redundant class and they can handle it.

If the faculty wants to take redundant classes due to the incompletion of the courses. Generally, for the preceptors there are principally 4 types of leave.

1. Earned leave
2. Confined leave
3. Casual leave
4. Sick leave

What are the problems that we face with ERP system?

The problem with the ERP software is if the faculty applies for leave and wants to allocate the class to any other faculty, also the request goes to all the faculties of all the departments. This shouldn't be because other department faculty cannot handle the class for any other department i.e. However, it must be transferred to the faculties of the Computer Science department only and not for any other department like Civil if the faculty of Computer Science department applies for the leave and if the request assent., Mechanical, E&E, and so on.

When the faculty is fitting the attendance into the system, there must be a separate space for the faculty to fill what motifs they've covered in the class. It'll be time consuming for the faculty to enter the content every time. So, for this purpose the software must be designed in such a way that it inserts the content automatically. Originally, all the motifs and the duration for the faculty in which the faculty must cover must be mentioned. And also, the faculty must probe it and cover the syllabus according to the plan. This can also keep a track of the speaker what they're teaching. However, also that would lead to deficit of time to cover the syllabus, if the dubieties are raised by the scholars. So, for this purpose the faculty can have the freedom to extend the duration to cover those motifs by handling redundant classes when the scholars are free. For taking the redundant class, the faculty must block in the time table and it must be visible to all the faculties of that class so that there would be no collision in handling the redundant class. Once if the faculty enters the attendance and if they press cinch option, also there won’t be any option to change the attendance of the scholars. Incipiently, the preceptors would like it if they could enter the attendance in the class itself. This would minimize the paper work and they could modernize the details at any place and at any time.

What do we anticipate from the module the lets you enter the marks of the scholars?

There will be another section to enter the CIE of all the scholars. The internals will be for 20 marks and when the faculty enters it into the ERP, it must automatically convert it into 10 marks. Generally, there will be 5 events. There will be 3 internals, followed by two events similar as quiz, project. However, also there must be a warning communication transferred to the pupil to score further marks in the forthcoming internals, if the pupil scores below 50 of the allocated marks in the subject.

At the end of all the events if the pupil couldn't mark the 50 marks, also there will be a make- up test conducted by the faculty so that the pupil would be having another chance to come up to the mark of 50. This make- up test marks must be altered with the minimum marks of the CIE scored. And the final CIE marks should be displayed and be stated that the pupil is eligible or not eligible to take up the Semester End Examination. However, also it must be brought into the notice of the speaker and the leave can be profited, if the pupil isn't suitable to take up the CIE due to particular reasons or if he's representing the council in any form of the activity. However, also the medical instrument must be attested, and a letter must be transferred to the Tank to take pure-test, if the pupil is ill. After the faculty enters the CIE there must be an option to save the CIE marks. When the CIE marks are saved also the scholars won't be suitable to see the marks in their marks. They can view their CIE only when the marks are locked by the faculty. However, also there would not be any chance to change the CIE, if the faculty locks the CIE. The CIE must be locked after attesting the marks with the scholars only.

Student

As a pupil, what are some problems you're facing with the current ERP system? The ERP status wasn't streamlined regularly, and they couldn't track their attendance status as the app would crash. The GUI that's used in the interface isn't over to the mark. It's delicate to keep the track of the attendance and the CIE. It would be easy if the attendance would be shown in a timetable like format so that it would be accessible and can also keep a track of the status of the attendance. There should also be forums where the schoolteacher and the scholars are active. This will help the scholars in numerous ways similar as studies, assignments, systems and so on. There should be commerce with the pupil- pupil and pupil- schoolteacher so that the scholars can clear their dubieties with any schoolteacher as well as any pupil at any point of time. The forum will also help the scholars in conveying the information to all the scholars at a faster rate.

For the scholars who were in supplementary batch, they couldn't attend the first many weeks of class as they had examinations. But, in the ERP they were marked as absent which made their attendance drastically low.

When the scholars are into council conditioning similar as LCC sessions, IEEE sessions, representing our council in sports or any other conditioning also scholars are pronounced absent. There must be another way to handle these problems so that there will be justice for the scholars for their hard work.

Director

What are your conditions from the ERP system as an admin? As a director, they deal with large quantum of data and functions. The system must be modular with a simple interface. The admin performs numerous functions on the database. These include searching for a record, add, modernize and cancel a record. Therefore, their interface needs to be quick and searching for records in the huge database must be optimized.

Elaboration

For the College ERP design, there are numerous classes of end druggies. These include the council staff, scholars and admin. As mentioned in the elicitation section, we talked with several stakeholders of different classes and collected their conditions. The conditions of the different classes were different. Some of them were in accord and some were in conflict. Therefore, elaboration and latterly concession is needed.

College staff

College staff is crucial stakeholders and use the ERP system the most. Therefore, it's essential to feed to their requirements first. Among the staff there are several different places. For each part, The ERP system will have a different view grounded on the conditions of that group.

Tutoring staff

Teacher staff makes up utmost of the staff. A schoolteacher expects the ERP system to be easy to use, dependable and reduce the work cargo. Each schoolteacher belongs to department and is assigned to a class of scholars with a course. So, the schoolteacher should only be suitable to view and manipulate the data of the scholars that they're assigned to.

The preceptors' involvement In the ERP System is to enter the attendance, the internal marks, the semester end examination marks. They will also have other features which include serving leave and managing a lecture plan for each course.

For Attendance operation, the preceptors anticipate a compact and functional interface. An interface where preceptors use minimum trouble to manage the attendance status of the scholars. The features anticipated for the attendance are to capability to enter the attendance to the entire class at formerly, edit the attendance of each individual pupil. Also, in the event of leave, they should be given an option of assigning the class to another schoolteacher, who takes a course for the same class.

In the event of entering internal marks and semester end examination marks, the schoolteacher enters the marks for each individual pupil. This is original a draft and can be edited. The scholars review the marks and verify. However, the pupil notifies the schoolteacher and the miscalculations are corrected, if there are any miscalculations. After certain quantum of time, when all the marks are verified, the marks are 'locked'. i.e., after locking, the marks can not to change.

When a school teacher applies for a leave, there are numerous options for different orders of leave. The first order is casual leave; this is for general purposes. Confined leave can be profited only on specific days given by the council. Also, Sick leave is for when the schoolteacher is ill. Incipiently, earned leave is an option given to each schoolteacher for a period of 15 days.

4.3. Project Planning and Project Scheduling

Gantt chart

Test	3 Days	5 Days	20 Days	42 Days	4 Days	3 Days
Feasibility						
Requirement Analysis						
Design						
Coding						
Testing						
Implementation						

Task	Start Date	Durations
Feasibility	7/1/2025	3
Requirement Analysis	7/5/2025	5
Design	7/11/2025	20
Coding	8/1/2025	42
Testing	8/19/2025	4
Implementation	9/25/2025	5

4.4. Software Requirement Specification (SRS)

Purpose

The purpose of this design is to develop a College Management System that helps the preceptors and scholars in easier operation of college conditioning similar as attendance, marks.

Intended followership and Reading Suggestions

This design is intended for staff and scholars of JSS Science and Technology University. This document has been made under the guidance of council professors. This document has been organized into Overall description followed by the features and also the functional and non-functional conditions. The document may be read to desire of the anthology.

Design compass

The design is designed to help the preceptors and scholars manage their council conditioning. It consists of relational databases of scholars, departments, faculty, and courses of the entire university. Using these databases, colorful functions that include Attendance operation, marks operation and leave management are handed. Within attendance operation, a schoolteacher can enter the attendance status of each pupil for each course with their separate dates. Analogous to attendance, Internal and Semester end marks can also be entered for each pupil.

References

- Software Engineering- A interpreter's approach by Roger S Pressman
- Fundamentals of database systems by Ramez Elmarsi and Shamkant Navathe

Overall Description

Product Perspective

This design is modeled grounded on the current ERP system in the council. Scholars and preceptors face several problems while using the system. Thus, we wanted to make a system that has lower number of features than the current system but, has further functionality.

Product Features

- Each schoolteacher will be suitable to enter attendance and marks for their separate scholars.
- Each pupil will be suitable to view the attendance status for their separate courses.
- The preceptors will be suitable to apply for colorful types of leave directly through the system.
- The scholars will be suitable to Communicate and give feedback to their preceptors.
- The scholars will have access to a forum runner where they're communicating will each other.

Characteristics

There are several types of end druggies for the council ERP system. They're astronomically divided as scholars, Staff and the director. Each of these classes has their own set of features.

The pupil should have the following features:

- View the Attendance status of the courses to which they're enrolled.
- View the Marks of the courses to which they're enrolled.
- View the announcement from the council director.
- Communicate or give feedback to their separate preceptors.
- Communicate with other scholars of the same university.

The staff should have the following features:

- Access to the information of all scholars that attend their courses.
- Add and edit the Attendance status of those scholars.
- Add and edit the test marks of those scholars.
- Avail the different types of leave.
- Exchange classes with other preceptors who educate for the same class.

The director should have the following features:

- Add and modernize scholars, preceptors and courses.
- Assign preceptors and scholars to courses.

The operating Environment for College ERP system is listed below:

- Operating System: Windows 10
- Database: MySQL database
- Front end: HTML/CSS/Bootstrap
- Back end: Python Django

Anticipated demand Pupil and staff information

Description and precedence Information regarding scholars, preceptors and courses are stored in the database. Every stoner can view only certain information grounded on their stoner class. For illustration, a schoolteacher can view pupil and course information that they're handling. This point is of high precedence as the information must be viewed by only the authorized druggies.

Functional conditions

- Each stoner shall be suitable to view information in the database grounded on their stoner class.
- The director shall be suitable to view all the information in the database.

Normal demand Attendance and marks entry

Description and precedence Attendance and marks entry is the main point of the College ERP system. Hence, the precedence is high. Preceptors modernize the attendance and marks of the scholars who are part of her class. Scholars can view their separate Attendance and marks of the courses they've taken.

Functional conditions

- Preceptors shall be suitable to view, modernize and edit the attendance and marks of the scholars, part of their class.
- School teacher shall be suitable to take redundant classes, switch classes with other preceptors.

Instigative demand Communication among scholars and preceptors.

Description and precedence scholars and schoolteacher will be suitable to communicate with each other directly using the ERP system. Scholars may give their queries and feedback to a schoolteacher and they may respond consequently. The precedence of this point is low as cost of perpetration could be veritably high. A simple interpretation of this point is to be enforced.

Functional conditions

- Scholars shall be suitable to communicate with their preceptors by sends particular dispatches.
- Scholars shall be suitable to communicate with other scholars through a forum section.

External Interface Conditions

Stoner Interfaces

The stoner interface is made using Bootstrap. Originally, there will be a simple login runner separate for scholars and preceptors. Each pupil and schoolteacher will have a unique interface. There will be a fixed sidebar with links to all the modules. The preceptors will be suitable to view their separate scholars and modernize their attendance and marks using a royal interface.

Tackle Interfaces

Since neither the mobile operation nor the web gate has any designated tackle, it doesn't have any direct tackle interfaces. Any cyber surfer can be used to pierce the webapp.

Software Interfaces

The following is a list of software used in timber of the design:

- Operating System we've chosen Windows operating system for its stylish support and stoner-benevolence.
- Django We've chosen to use Django for the reverse- end of the website as Django is a simple python frame and is suitable for newcomers.
- Database we're using SQLite database, which comes as dereliction with Django.

Dispatches Interfaces

- This design is to be stationed an online website.
- All the druggies can connect to the database garçon from anywhere and have access to their information.

Non-functional conditions

Safety conditions

Still, similar as a fragment crash, the recovery system restores an once dupe of the database that was backed up to archival storehouse (generally tape recording) and reconstructs a more current state by reapplying or redoing the operations of married deals from the backed- up log, If there's expansive damage to a wide portion of the database due to disastrous failure.

Security conditions

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

Software Quality Attributes

Availability: The druggies must always be suitable 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 in- conformation to the druggies.

Portability: The druggies pierce the ERP from colorful platforms similar as desktops and mobile phones. The webapp must be movable to all platforms and the stoner experience must be optimal.

4.5. Software Engineering Paradigm

Software paradigm refers to method and steps, which are taken while designing the software. Programming paradigm is a subset of software design paradigm which is further a subset of software development paradigm. Software is considered to be a collection of executable programming code, associated libraries, and documentation. Software development paradigm is also known as software engineering, all the engineering concepts pertaining to developments software applied. It consists of the following parts as Requirement Gathering, Software design, Programming, etc. The software design paradigm is a part of software development. It includes design, maintenance, programming.

Software paradigm is a theoretical framework that serves as a guide for the development and structure of a software system. There are several software paradigms, including:

- **Imperative paradigm:** This is the most common paradigm and is based on the idea that a program is a set of instructions that tell a computer what to do. It is often used in languages such as C, C++, Java and Python.
- **Object-oriented paradigm:** This paradigm is based on the idea of objects, which are self-contained units that contain both data and behavior. It is often used in languages such as Java, C#, and Python.
- **Functional paradigm:** This paradigm is based on the idea that a program is a set of mathematical functions that transform inputs into outputs. It is often used in languages such as Haskell, Lisp, and ML.
- **Logic paradigm:** This paradigm is based on the idea that a program is a set of logical statements that can be used to infer new information. It is often used in languages such as Prolog and Mercury.

The Software Development Life Cycle (SDLC) is a process that software developers use to plan, design, develop, test, deploy, and maintain software systems. The most common SDLC models include:

- **Waterfall model:** This model is based on the idea that software development is a linear process, with each phase building on the previous one.
- **Agile model:** This model is based on the idea that software development is an iterative process, with small.

Adopted Model in this Project

The **waterfall model** is a sequential development approach, in which development is seen as flowing steadily downwards (like a waterfall), through several phases, typically:

- Requirements analysis resulting in a software requirements specification.
- Software design
- Implementation
- Testing
- Integration, if there are multiple subsystems
- Deployment or installation
- Maintenance

The first formal description of the method is often cited as an article published by Winston in 1870 although Royce did not use the term “waterfall” in this article the basic principles are:

- Project is divided into sequential phases, with some overlap and splash back acceptable between phases.
- Emphasis is on planning, time schedules, target dates, budgets and implementation of an entire system at one time.

Why it fits ERPs

While less suited for the entire ERP project, specific, well-defined phases, such as initial requirements gathering or final deployment, can benefit from the structured, sequential nature of the Waterfall model. It provides clear milestones for predictable parts of the project.

4.6. Data Models

ER - Diagram

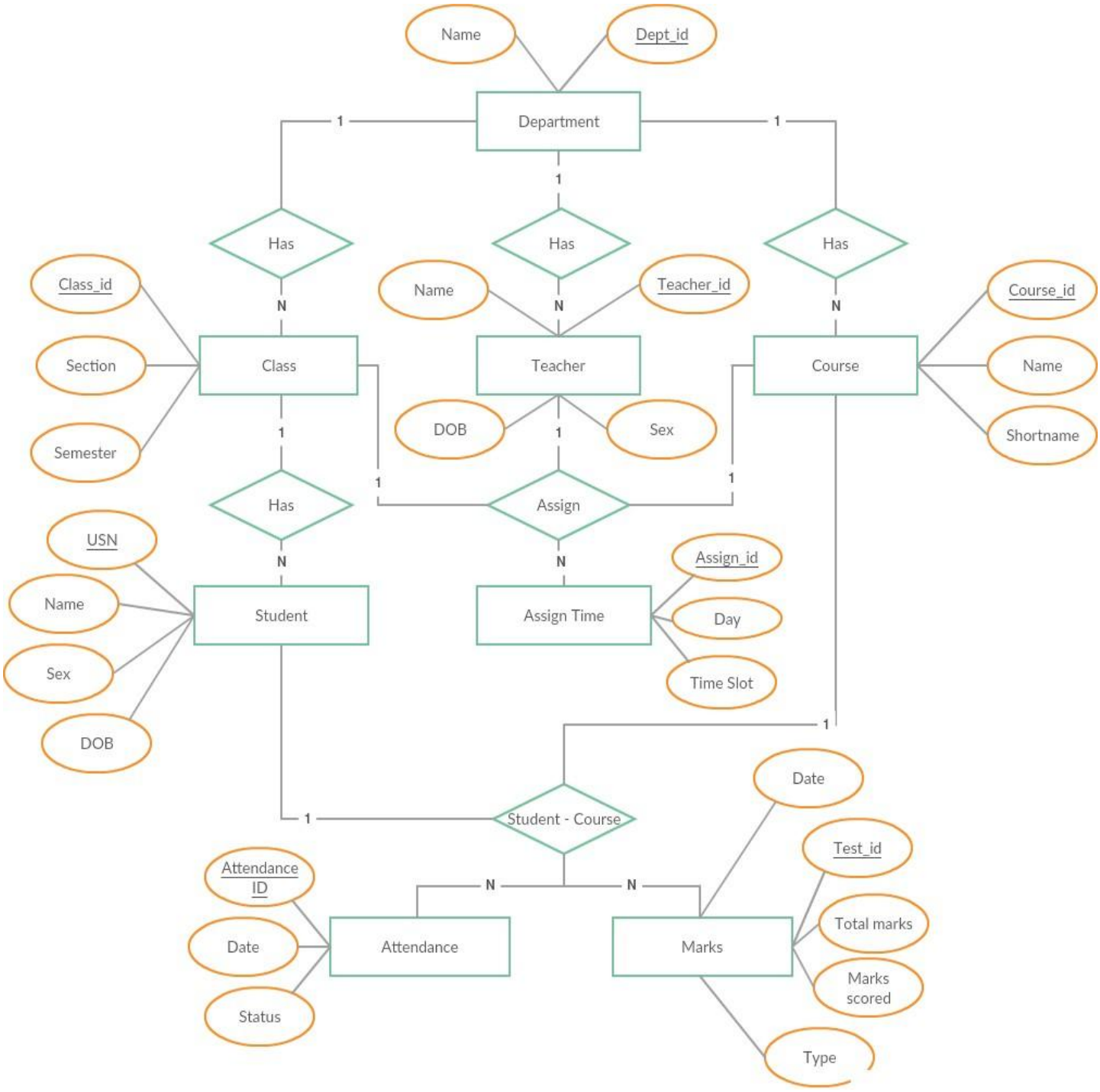


Fig: Entity Relationship diagram of college ERP

Data Flow Diagram

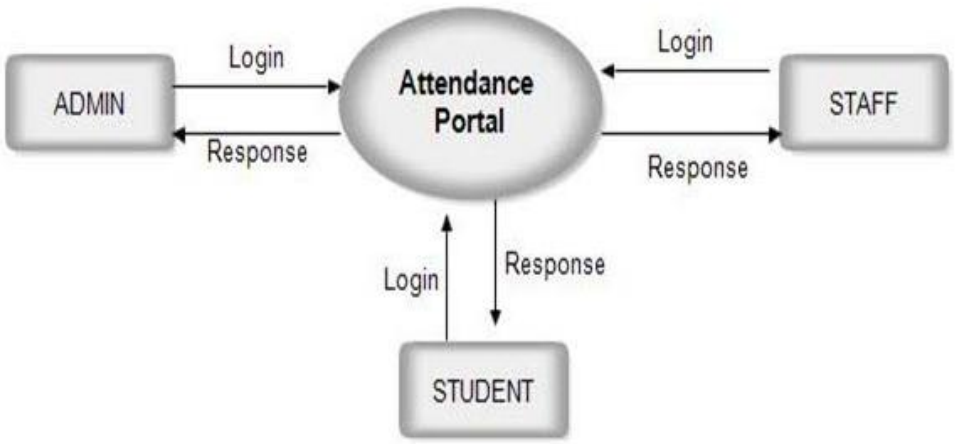


Fig: Level – 0

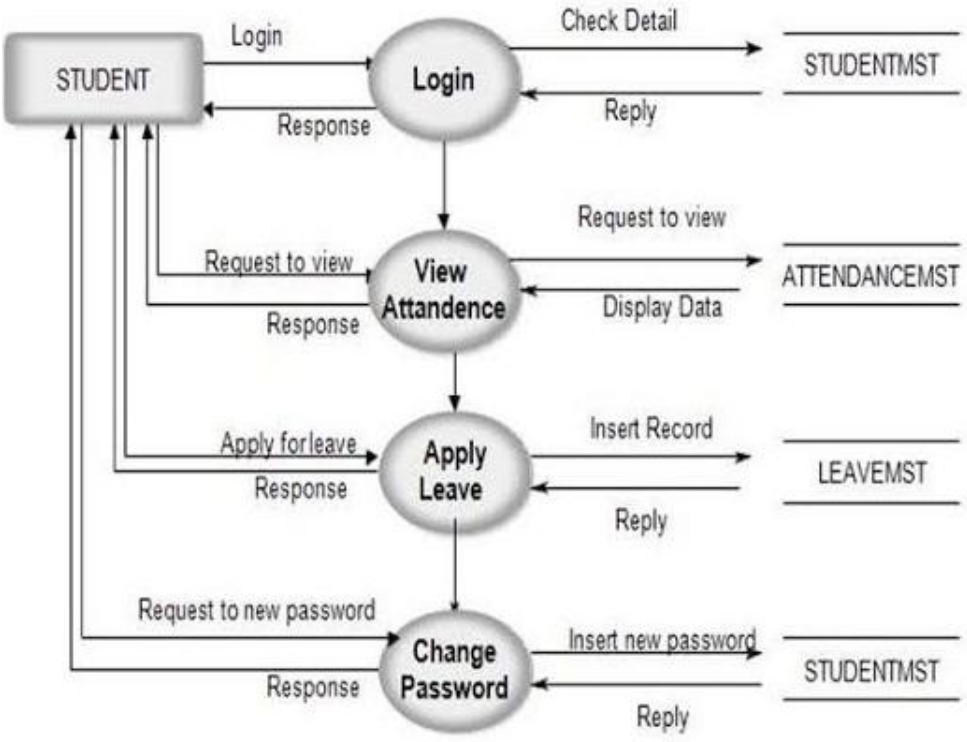


Fig: Level -1 Student

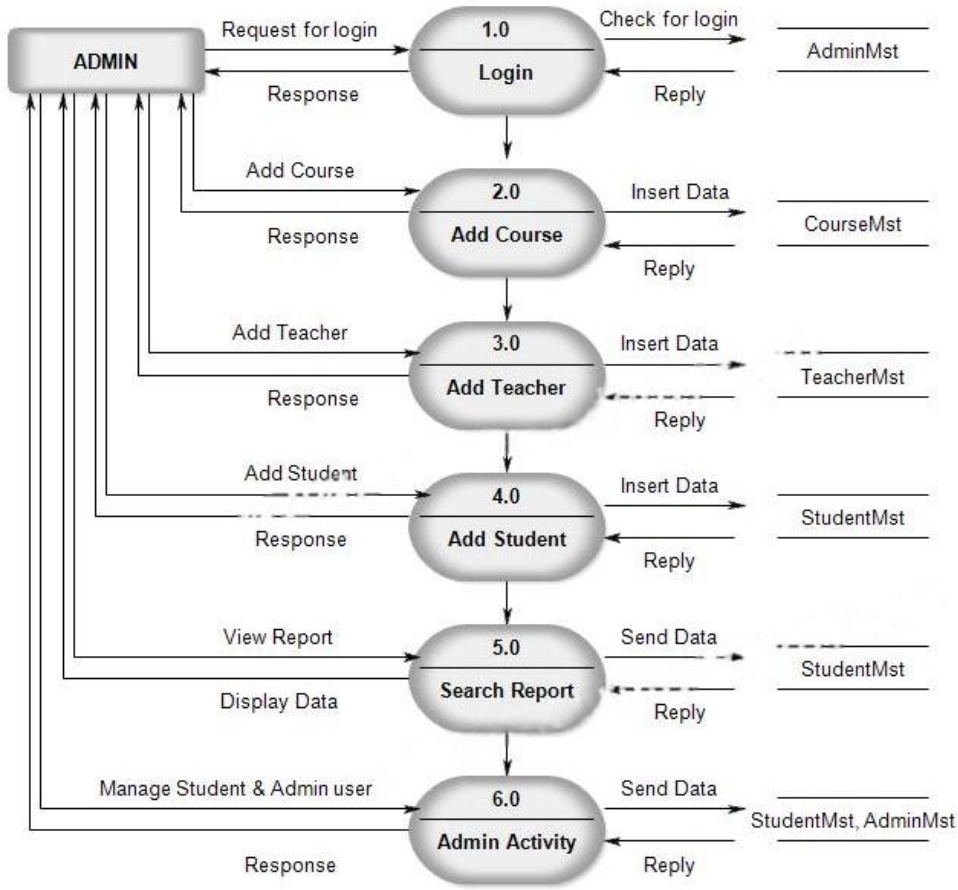


Fig: Level -1 Admin / Teacher

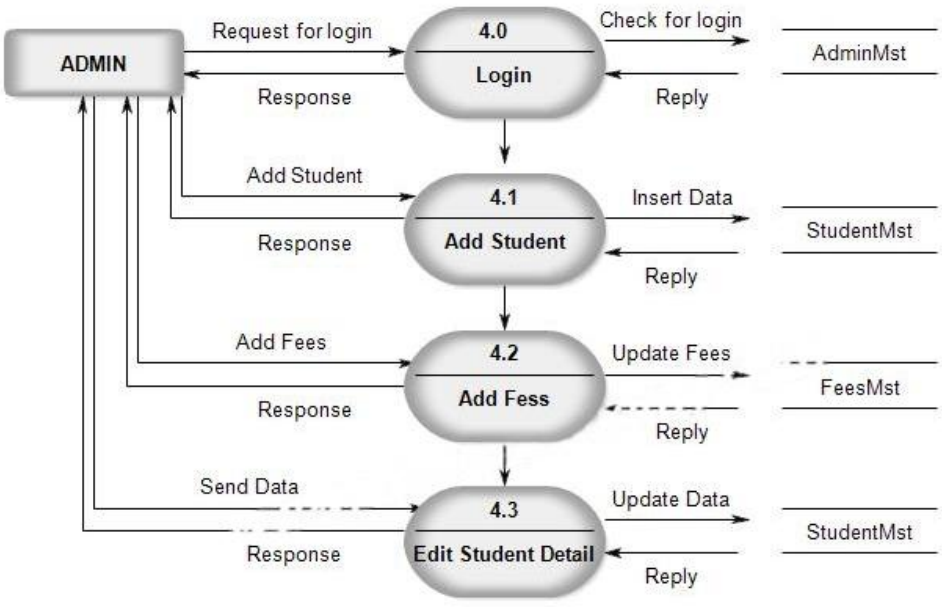


Fig: Level -2 Admin

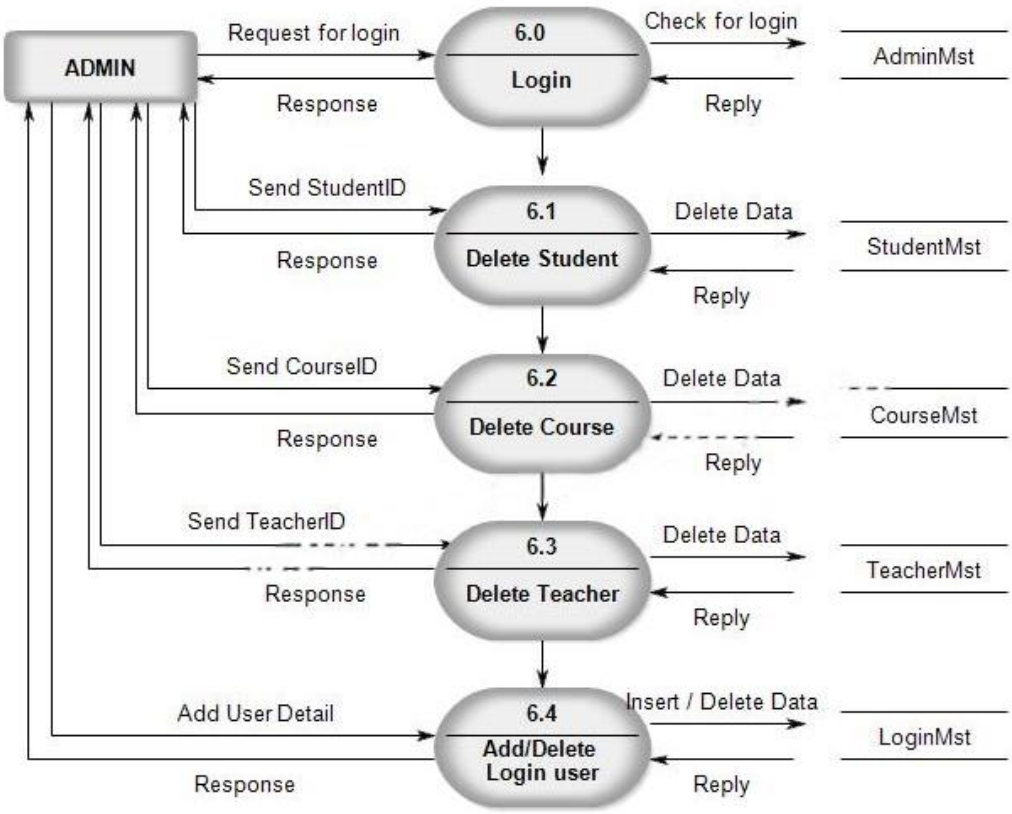


Fig: Level -2 Admin / Teacher

Activity chart

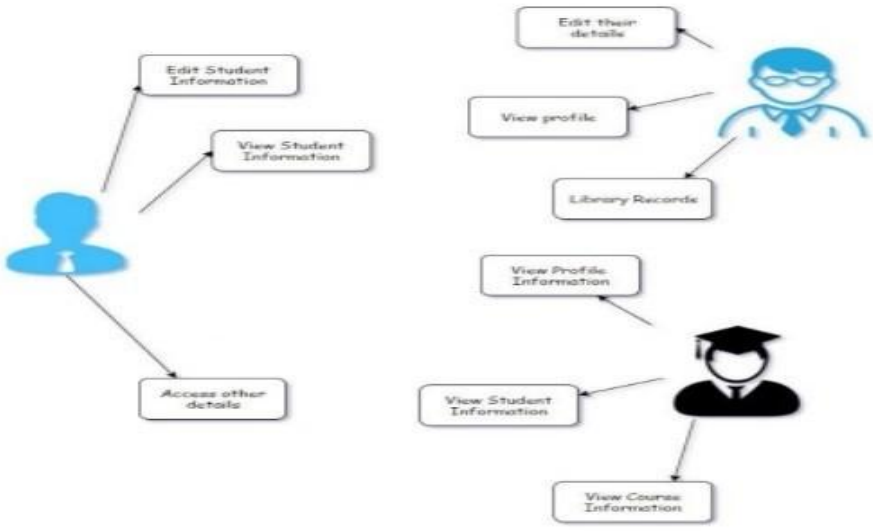


Fig: Activity diagram of college ERP

Class Diagram

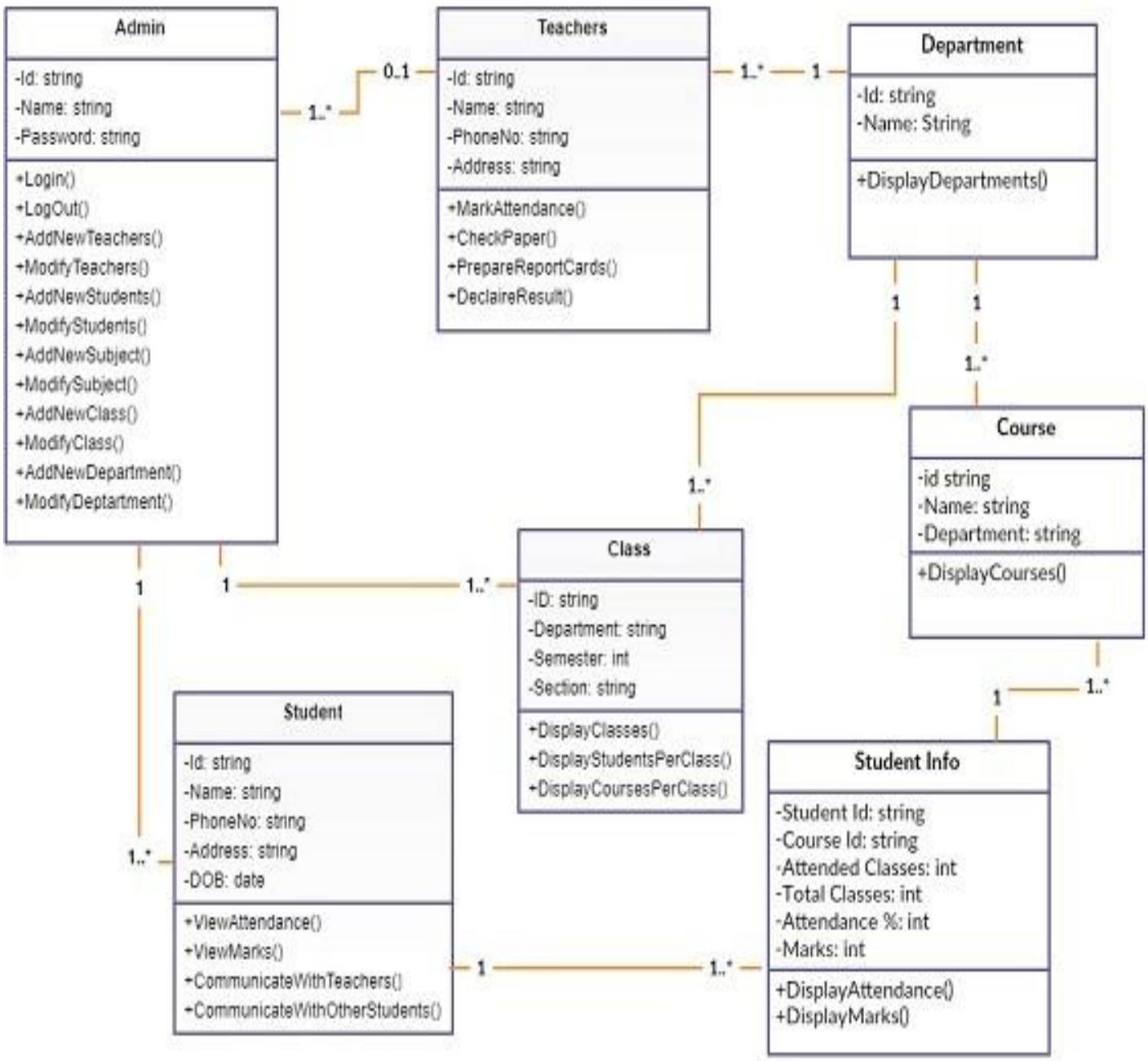


Fig: Class diagram of college ERP

UseCaseDiagram

A use case diagram at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved. A use case diagram can identify the different types of users of a system and the different use cases and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

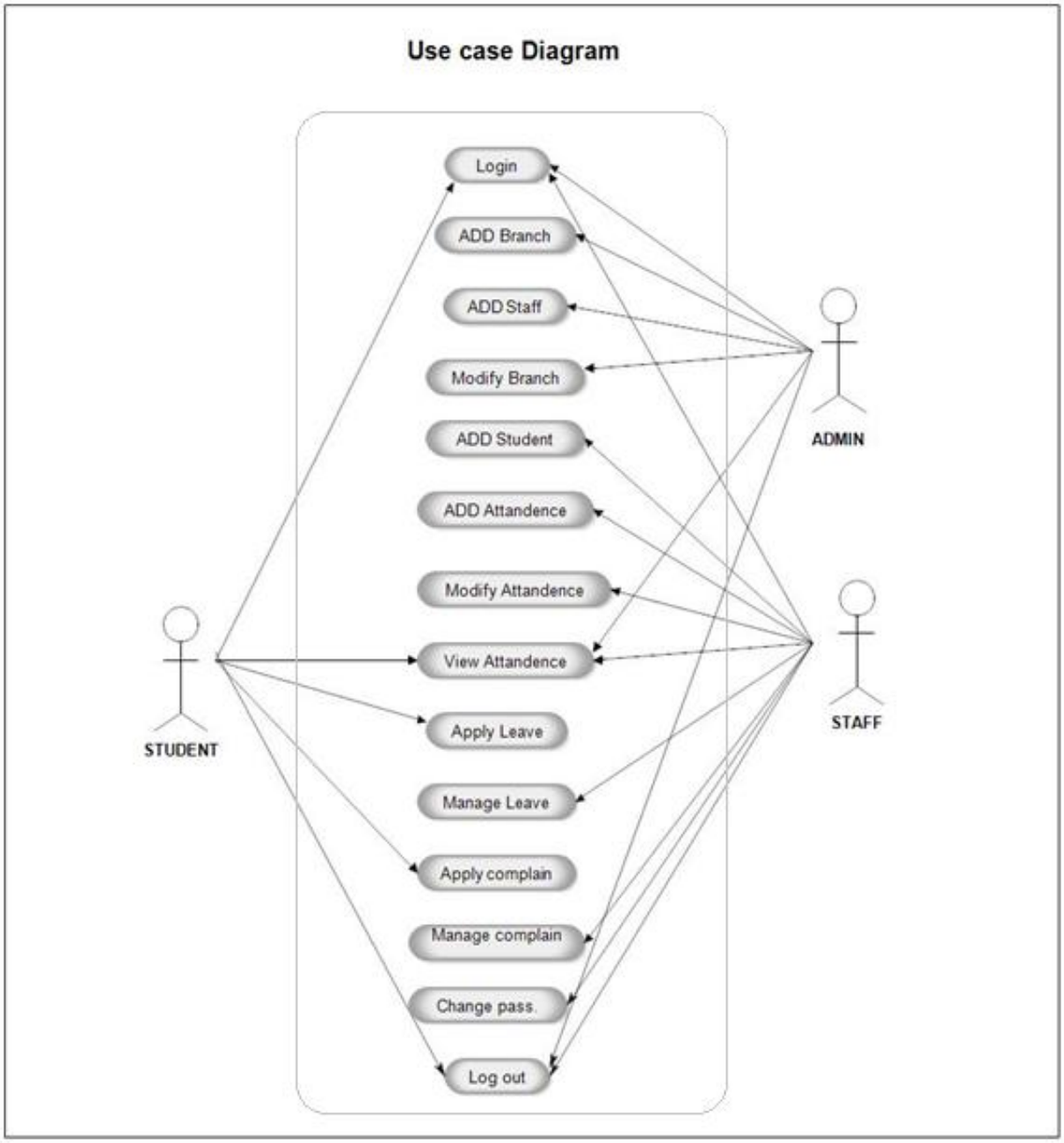


Fig: Use Case Diagram

5. SYSTEM DESIGN

5.1. Project Modularization details

1. ASGI (pip install asgiref)

ASGI is a standard for Python asynchronous web apps and waiters to communicate with each other, and deposited as an asynchronous successor to WSGI.

- This package includes ASGI base libraries, similar as
- Sync- to- async and async- to- sync function wrappers, asgiref.sync
- Garçon base classes, asgiref.server
- A WSGI- to- ASGI appendage, inasgiref.wsgi

Function wrappers

These allow you to wrap or embellish sync or sync functions to call them from the other style(so you can call sync functions from a coetaneous thread, or vice-versa).

AsyncToSync lets a coetaneous sub thread stop and stay while the sync function is called on the main thread's event circle, and also control is returned to the thread when the sync function is finished.

Sync To Async lets a sync law call a coetaneous function, which is run in a thread pool and control returned to the async co routine when the coetaneous function completes.

The idea is to make it easier to call coetaneous APIs from a sync law and asynchronous APIs from coetaneous law so it's easier to transition law from one style to the other. In the case of Channels, we wrap the(coetaneous) Django view system with SyncToAsync to allow it to run inside the(asynchronous) ASGI garçon.

2. **Django**

Django is a high- position web frame for erecting web operations in Python. It follows the model- template- view (MTV) architectural pattern and provides a set of tools and libraries that help inventors make scalable and justifiable web operations snappily.

Django includes features similar as an object- relational mapper (ORM) for database access, an important URL routing system, erected- in authentication and security features, an admin panel for managing operation data, and a templating machine for structure stoner interfaces.

With Django, inventors can concentrate on writing operation sense rather of fussing about low- position details of web development, similar as handling HTTP requests, managing sessions, or dealing with database connections.

Django is open- source software and has a large and active community of inventors, which means that it's constantly being bettered and streamlined with new features and security fixes.

3. Django REST framework

djangorestframework

Django REST frame (DRF) is an important toolkit for erecting Web APIs in Django. It's erected on top of Django and provides a set of tools and serviceability that make it easy to make peaceful APIs.

DRF provides a range of features similar as serializers, authentication and warrants, views, and routers. Serializers allow you to convert complex data types, similar as Django models, into JSON or other content types that can be transmitted over the web. DRF also supports a wide range of authentication and authorization schemes, allowing you to secure your API endpoints in a variety of ways.

Views in DRF are analogous to Django views, but they're designed specifically for handling web API requests. DRF also provides a important router system that allows you to define URL patterns for your API endpoints and automatically generates views for those endpoints.

DRF also includes support for pagination, filtering, and searching, making it easy to make APIs that can handle large quantities of data.

Overall, Django REST frame is a important and flexible tool for erecting Web APIs in Django, and it's extensively used by inventors around the world.

4. **Djoser**

pip install djoser

Djoser is a Django package that provides a set of views and serializers for handling stoner authentication and enrollment in Django REST frame (DRF) APIs. It's designed to be easy to use and largely customizable, and provides a range of features similar as word reset and dispatch evidence.

Djoser supports token- grounded authentication, which is a common system for securing DRF APIs. It also provides a range of serializers for handling stoner enrollment, login, and other affiliated tasks.

One of the main advantages of Djoser is that it's largely configurable, which means that inventors can fluently customize the authentication and enrollment workflows to fit the requirements of their specific operations. For illustration, Djoser allows inventors to customize the dispatch templates that are transferred to druggies during enrollment or word reset workflows.

Overall, Djoser is a useful tool for inventors who are erecting Django REST frame APIs that bear stoner authentication and enrollment. It simplifies the development process and provides a range of features that make it easy to make secure and scalable APIs.

5. Python pytz

pip install pytz

“pytz” is a Python package that provides functionality for working with time zones. It is used to localize datetimes, perform time zone conversions, and perform arithmetic with localized times.

pytz is particularly useful in situations where applications need to handle datetimes from different time zones or where it is important to maintain accurate and consistent time measurements across different locations.

One of the main advantages of pytz is that it provides a comprehensive database of time zones that is regularly updated. This database includes over 500 time zones, including historical time zones that are no longer in use. pytz also supports the Internationalization (I18n) features of Python's datetime module, which makes it easy to perform localized formatting of dates and times.

Using pytz is relatively simple. Once the package is installed, you can import it and use its functions to localize and convert datetimes. For example, you can use pytz.timezone() to create a timezone object, which can then be used to localize or convert datetimes.

Overall, pytz is a powerful tool for handling time zones in Python, and it is widely used by developers around the world.

6. **SQLPARSE**

pip install sqlparse

“sqlparse” is a Python library that provides functionality for parsing and formatting SQL statements. It can be used to dissect, manipulate, and induce SQL law in a variety of ways.

sqlparse is particularly useful for working with complex SQL statements that include multiple subqueries, join statements, and other advanced features. It can be used to format SQL law for readability, highlight syntax crimes, and prize individual rudiments of SQL statements, similar as table and column names.

One of the main advantages of sqlparse is that it's largely customizable. It provides a range of configuration options that allow inventors to fine- tune the geste of the library to match their specific requirements. For illustration, inventors can customize the indentation style of formatted SQL law, enable or disable certain parsing features, and customize the affair format of parsed SQL law.

Using sqlparse is fairly simple. Once the package is installed, you can import it and use its functions to parse and format SQL statements. For illustration, you can usesqlparse.parse() to parse a SQL statement into a list of individual commemoratives, or you can usesqlparse.format() to format a SQL statement for readability.

Overall, sqlparse is a important tool for working with SQL law in Python, and it's extensively used by inventors around the world.

5.2. Project Design

Colorful Design generalities and processes were applied to this design. Following generalities like separation of enterprises, the software is divided into individual modules that are functionally independent and incorporates information caching. The software is divided into 3 modules which are scholars, preceptors and directors. We shall look at each module in detail.

Pupil

Each pupil belongs to a class linked by semester and section. Each class belongs to a department and is assigned a set of courses. Thus, these courses are common to all scholars of that class. The scholars are given a unique username and word to login. Each of them will have a different view. These views are described below.

Pupil information

Each pupil can view only their own particular information. This includes their particular details like name, phone no, address etc. Also, they can view the courses they're 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. However, say 75, It'll be marked in red else it is in green, If the attendance chance if below a specified threshold. There will also be a day wise attendance view for each course which shows the date and status. This will be presented in a calendar format.

Marks information

There will be 5 events and 1 semester end examination for each course. The marks for each of these will be handed in the ERP system.

Announcements and events

This section is common to all scholars. Announcement is dispatches from the admin similar as declaration of leaves, test time- table etc. The events and their details are specified then.

Schoolteacher

Each schoolteacher belongs to a department and is assigned to classes with a course. Preceptors will also have a username and word to login. The different views for preceptors are described below.

Information

The preceptors will have access to information regarding the courses and classes they're 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 scholars in each class. The schoolteacher will also have access to information of scholars who belong to the same class as the schoolteacher.

Attendance

The schoolteacher has the capability to add and also edit the attendance of each pupil. For entering the attendance, they will be given the list of scholars in each class and they can enter the attendance of the whole class on a day-to-day base. There will be two radio buttons next to each pupil name, one for present and the other for absent. There will also be an option for redundant classes. Preceptors can edit the attendance of each pupil either for each pupil collectively or for the whole class.

Marks

The schoolteacher can enter the marks for the 5 events and 1 SEE for each course they're assigned. They also have the capability to edit the marks in case of any changes. Reports similar as the report card including all the marks and CGPA of a pupil can be generated.

Director

The director will have access to all the information in the different tables in the database. They will pierce to all the tables in a list form. They will be suitable to add an entry in any table and also edit them. The design of the view for the admin will give a modular interface so that querying the tables will be optimized. They will be handed with hunt and sludge features so that they can pierce data efficiently.

In conclusion, the project design outlines a modular, role-based College ERP System built on the principle of separation of concerns. It effectively segregates functionality into three distinct portals for students, teachers, and directors, ensuring data security and a tailored user experience. Students gain a personalized view of their academic progress, including attendance and marks. Teachers are equipped with tools to manage academic data for their classes, from recording attendance to entering marks and generating reports. The director enjoys comprehensive administrative control with full access to all database tables for efficient data management. Overall, the design promises a structured, efficient, and intuitive platform for managing core academic operations within an educational institution.

5.3. Data integrity and constraints

Add User

S. No.	Field Name	Data Type	Constraints	Description
1	Name	String	Not null	Name of User (F&L)
2	Email	String	Not null	Email of User
3	Password	String	Not null	User Password
4	Id	String	Primary key	Unique Id
5	Role	String	Not null	Teacher Or Student

Department

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Name	String	Not null	Name

Course

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Unique Id
2	Dept.	String	Foreign key	Name
3	Name	String	Not null	Course Name
4	Shortname	String	Not null	Short name

Classes

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Dept.	String	Foreign key	Name
3	Section	String	Not null	Section
4	Sem.	String	Not null	Semester

Marks

S. No.	Field Name	Data Type	Constraints	Description
1	Student	String	Foreign key	Name & ID
2	Course	String	Foreign key	Name & ID

Assignments

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Course	String	Foreign key	Name & Id
3	Teacher	String	Foreign Key	Name & Id

5.4. Database Design

Architectural design

The ERP software requires the architectural design to represent the design of the software. Then we define a collection of tackle and software factors and their interfaces to establish the frame for the development of this software.

There exists number of factors of the system which are integrated to form a system. The set of connectors will help in collaboration, communication, and cooperation between the factors. The ERP software is erected for computer- grounded system. It exhibits the data centric style of armature.

Architectural style

In the council ERP software, the database stores the data of all the scholars and faculties and the stored data is streamlined, added, deleted or modified. So, it exhibits the data centric architectural style.

In this armature different factors communicate with the participated data depository. The factors pierce a participated data structure and are fairly independent.

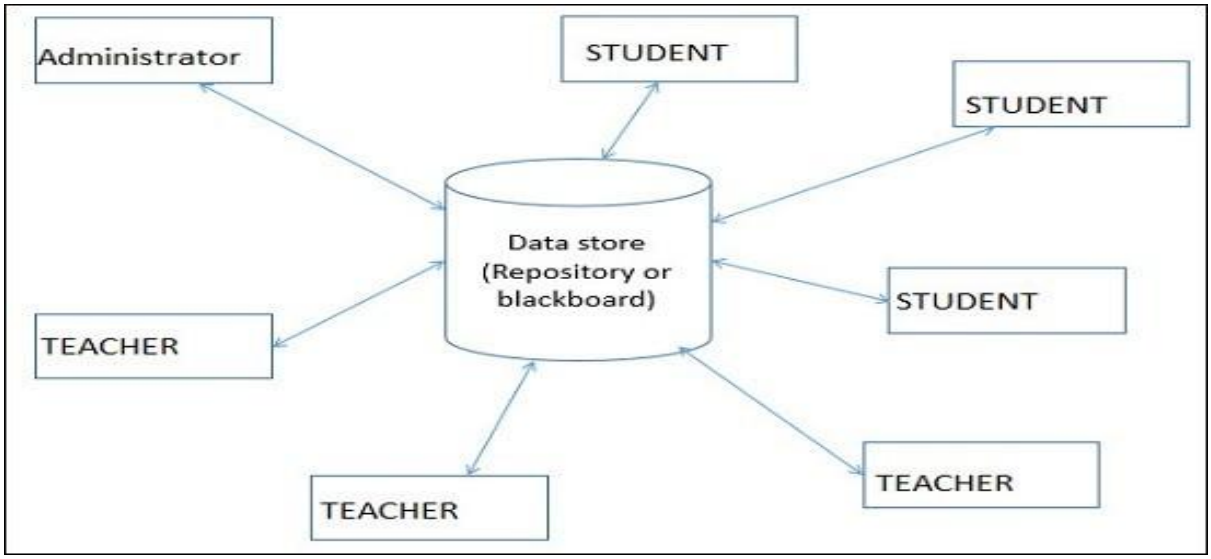


Fig: Data Centric architectural style

Central Data

Also known as data store or data depository, which is responsible for furnishing endless data storehouse? It represents the current state. It stores the information of scholars, attendance of scholars and faculties of each day, payment of all the facultiesetc.

This is the heart of the system, often called a repository or blackboard. It provides **persistent data storage** and acts as the single source of truth for all information. In this case, it stores all the data related to the college, such as student records, faculty details, attendance, and financial information. Components don't communicate directly with each other; instead, they interact with this central data store to read, write, or modify data.

Data Assessors

Data assessors one of the factors, they're also called as guests. A data accessor operates on the central data store, performs calculations, and might put back the results. Which includes scholars, faculties and director? Scholars request to pierce the data from the depository and get the request serviced. Faculty members modify the data in the depository. Director can add or cancel the guests.

These are the clients or actors that interact with the central data store. They are fairly independent and perform specific operations. The text identifies three types of data accessors:

- **Students:** They primarily request to access data from the repository, such as checking their grades or attendance.
- **Faculty Members:** They can modify data in the repository, for example, updating student grades or attendance records.
- **Administrators:** They have the highest level of access and can add or delete other users (students, faculty) and perform other administrative tasks.

5.5. User Interface Design

Based on the detailed requirements, here is a proposed interface design for the College Staff (Teaching) module.

College Staff (Teaching) Dashboard Interface

The interface is designed to be clean, modular, and task-oriented, reducing clicks and streamlining the workflow for teachers.

1. Main Dashboard View:

Upon login, the teacher is greeted with a dashboard that provides a high-level overview and quick access to key functions.

Navigation Sidebar: A collapsible sidebar on the left with clear icons and text for each module:

- Dashboard (Home icon)
- My Classes (List icon) - Shows assigned classes (e.g., CSE - 5th Sem - Section A).
- Attendance (Checkmark icon)
- Marks / CIE (Bar Chart icon)
- Leave Management (Calendar icon)
- Syllabus Planner (Clipboard icon)

Central Information Panel:

Welcome & Notifications: A "Good Morning, [Teacher Name]" message and a section for urgent announcements from admin.

Quick Stats: Cards showing:

Classes Today: A list of immediate upcoming classes with times and a quick link to take attendance.

Pending Actions: Number of leave requests awaiting approval (for HODs), unsubmitted marks, etc.

Recent Activity: A log of the teacher's recent actions (e.g., "Attendance updated for CSE-5A on [Date]").

2. Attendance Management Interface:

Clicking "Attendance" from the sidebar leads to a dedicated, streamlined interface.

Class Selection: A dropdown at the top to select the Department -> Semester -> Section -> Course.

Date Selection: A clear calendar widget to choose the date. The system automatically suggests the current date and checks for timetable conflicts.

Attendance Entry Panel:

Class Type: Radio buttons to select Regular Class or Alternate Class (if covering for another teacher).

Attendance Taker: A simple, scrollable list of all students in the selected class.

Each student row has:

Student Name & USN/ID.

Two large, clearly labeled radio buttons: Present | Absent. (The default could be "Present" to minimize clicks).

A small notes icon for adding a reason for absence (e.g., "University Sport", "Medical"), addressing the student concern about justified absences.

Bulk Actions: Buttons at the top: "Mark All Present", "Mark All Absent" (with a confirmation prompt).

Topic Covered (Optional but Prominent): A text field auto-populated with the next topic from the predefined syllabus plan. The teacher can easily edit it if the plan changed. This addresses the teacher's requirement directly.

Action Buttons: "Save Draft" (allows later changes) and a prominent "Submit & Lock" button (which triggers a confirmation warning: "You will not be able to edit after locking. Continue?").

3. Marks/CIE Management Interface:

This interface is designed for clear data entry and review.

Hierarchical Selection: The teacher first selects Class -> Course -> Assessment Event (e.g., "Internal 1", "Quiz 2", "Make-up Test").

Marks Entry Table: A clean table listing all students.

Columns: Student ID, Name, Marks (editable text field), Max Marks (displayed as reference, e.g., 20).

Auto-Calculation & Warnings: As marks are entered, the system automatically calculates the equivalent value (e.g., converting 17/20 to 8.5/10). If a value below 50% is entered, the cell is highlighted in yellow as an immediate warning to the teacher.

Workflow Buttons:

Save: Saves a draft. Students cannot see draft marks.

Notify Students for Review: This pushes the marks to the students' view for verification. This is the "draft" state mentioned in the requirements.

Lock Final Marks: After the review period, this button becomes active. Clicking it locks the marks permanently (requires a second authentication step or password confirmation). A log is generated for audit purposes.

4. Leave Management Interface:

A form-based interface to simplify the leave application process.

Leave Application Form:

Leave Type: Dropdown menu (Casual Leave, Sick Leave, Earned Leave, Confined Leave).

Date Range: From [Date] To [Date] (with a calendar picker).

Reason: Text area for description.

Intelligent Class Assignment:

Upon selecting dates, the system automatically lists the classes that will be missed.

A checkbox for each class with the option: "Request alternate faculty assignment."

When checked, the system uses the timetable to intelligently suggest available teachers from the same department, addressing a major pain point. The teacher can select a suggested name or choose another.

Status Tracker: A section showing the status of the application (e.g., "Pending with HOD", "Approved") and any comments.

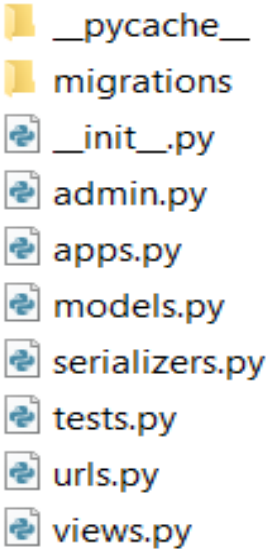
This interface design prioritizes the teachers' needs for efficiency, clarity, and control, directly incorporating the feedback gathered during the elicitation phase.

6. CODING

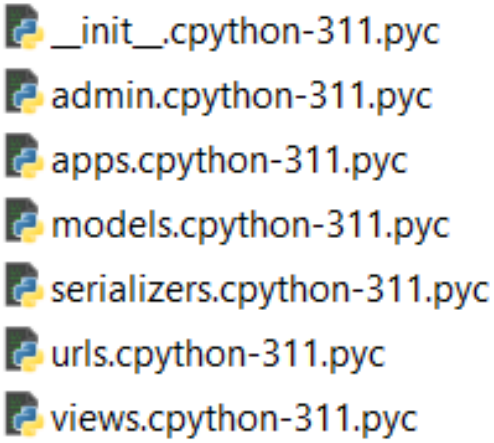
IMPLIMENTATION

Source Code

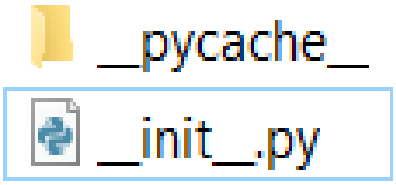
1. Apis



2. __pycache__



3. migrations



==========

__init__.py

==========

admin.py

```
from django.contrib import admin

# Register your models here.
```

==========

apps.py

```
from django.apps import AppConfig


class ApisConfig(AppConfig):
    name = 'apis'
```

==========

models.py

```
from django.db import models

# Create your models here.
```

==========

serializers.py

```
from rest_framework import serializers

from info.models import *


class DetailSerializer(serializers.ModelSerializer):
    class Meta:
        model = Student
        fields = '__all__'


class AttendanceSerializer(serializers.ModelSerializer):
    class Meta:
        model = AttendanceTotal
        fields = '__all__'


class MarksSerializer(serializers.ModelSerializer):
    class Meta:
        model = Marks
        fields = '__all__'
```

```
class TimeTableSerializer(serializers.ModelSerializer):
    class Meta:
        model = AssignTime
        fields = '__all__'
```

==========

tests.py

```
from django.test import TestCase

# Create your tests here.
```

==========

urls.py

```
from django.urls import path, include
import apis.views as api_view
from django.contrib import admin


urlpatterns = [
    path('details/', api_view.DetailView.as_view()),
    path('attendance/', api_view.AttendanceView.as_view()),
    path('marks/', api_view.MarksView.as_view()),
    path('timetable/', api_view.TimetableView.as_view()),
]
```

views.py

```
from django.shortcuts import render
from info.models import *
from rest_framework.response import Response
from rest_framework.views import APIView
from rest_framework.authtoken.models import Token
from rest_framework.permissions import IsAuthenticated, AllowAny
from rest_framework.pagination import PageNumberPagination
from itertools import chain
from rest_framework import serializers, status
from rest_framework.generics import ListAPIView
from django.db.models.signals import post_save
from rest_framework.generics import get_object_or_404
from rest_framework import generics
from rest_framework import mixins
from rest_framework import status
from django.db.models import Sum, Count
from django.conf import settings
import apis.serializers as api_ser


class DetailView(APIView):
    """
    Returns user's info.
    """
    permission_classes = [IsAuthenticated, ]

    def get(self, request):
        try:
            # fetching token sent in request header by the user.
            us = Token.objects.filter(user=request.user)
```

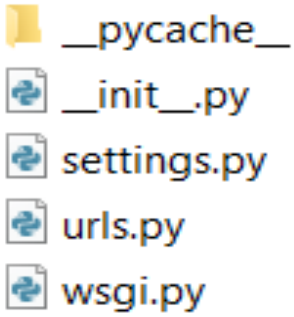
```
            if(us):         # checking for authentication using token authentication.

                # getting user from in-built user model class.
                user = User.objects.filter(auth_token=us[0]).first()
                # getting student from student model by filtering based on user that we got.
                details = Student.objects.get(user=user)
                serializer = api_ser.DetailSerializer(
                    details, context={'request': request})       # Serializing the data into Json format.
                return Response({'data': serializer.data, }, status=status.HTTP_200_OK)
            else:
                return Response({'message': 'User not authenticated'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response(str(e), status=status.HTTP_400_BAD_REQUEST)


class AttendanceView(APIView):
    """
    This view is used to return user's attendance
    that is to check user's attendance.
    """
    permission_classes = [IsAuthenticated, ]

    def get(self, request):
        try:
            token = Token.objects.filter(user=request.user).first()
            if(token):  # checking for authentication using token authentication.
                # getting user from in-built user model class.
                user = User.objects.get(auth_token=token)
                # getting student from student model by filtering based on user that we got.
                stud = Student.objects.get(user=user)
                # using ass_list and att_list we get the classes assigned to that user
                ass_list = Assign.objects.filter(class_id_id=stud.class_id)
```

```
                # and respectively their attendance
                att_list = []
                for ass in ass_list:
                    try:
                        a = AttendanceTotal.objects.get(
                            student=stud, course=ass.course)
                    except AttendanceTotal.DoesNotExist:
                        a = AttendanceTotal(student=stud, course=ass.course)
                        a.save()
                    att_list.append(a)
                serializer = api_ser.AttendanceSerializer(
                    att_list, many=True, context={'request': request})     # Serializing the data into Json
format.
                return Response({'user_attendance': serializer.data, }, status=status.HTTP_200_OK)
            else:
                # returning not authenticated message when user isn't authenticated with status code 400.
                return Response({'message': 'User not authenticated'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response(str(e), status=status.HTTP_400_BAD_REQUEST)


class MarksView(APIView):
    """
    This view is used to return user's marks
    that is to check user's marks in different subjects as given by the teacher.
    """
    permission_classes = [IsAuthenticated, ]

    def get(self, request):
        try:
            token = Token.objects.filter(user=request.user).first()
            if(token):  # checking for authentication using token authentication.
```

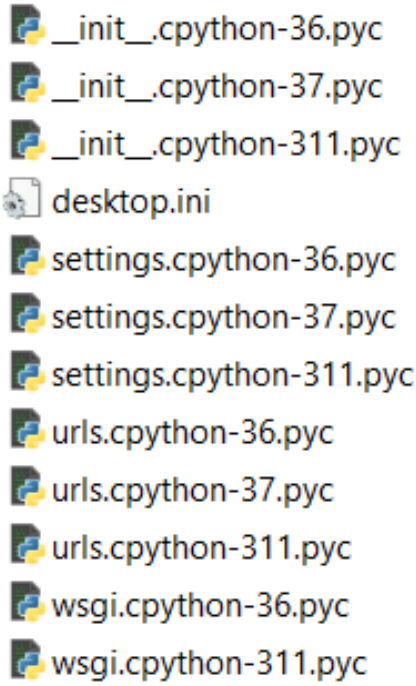
```
                user = User.objects.get(auth_token=token)
                stud = Student.objects.get(user=user)

                # using ass_list and sc_list we retrieve all the subjects assigned
                ass_list = Assign.objects.filter(class_id_id=stud.class_id)
                # and then their respective marks. Store them in a dictionary and return it to the user.
                sc_list = []
                for ass in ass_list:
                    sc = StudentCourse.objects.get(
                        student=stud, course=ass.course)
                    sc_list.append(sc)
                sc_total = {}
                for sc in sc_list:
                    for m in sc.marks_set.all():
                        sc_total[m.studentcourse.course.name] = m.marks1
                return Response({'user_marks': sc_total, }, status=status.HTTP_200_OK)
            else:
                return Response({'message': 'User not authenticated'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response(str(e), status=status.HTTP_400_BAD_REQUEST)


class TimetableView(APIView):
    """
    This view is used to check user's class timetable
    It returns the respective class' timetable to which the user is assigned.
    """

    permission_classes = [IsAuthenticated, ]

    def get(self, request):
        try:
```

```
            token = Token.objects.filter(user=request.user).first()
            if(token):  # checking for authentication using token authentication.
                user = User.objects.get(auth_token=token)
                stud = Student.objects.get(user=user)
                asst = AssignTime.objects.filter(
                    assign__class_id=stud.class_id)
                serializer = api_ser.TimeTableSerializer(
                    asst, many=True, context={'request': request})     # Serializing the data into Json format.
                return Response({'user_marks': serializer.data, }, status=status.HTTP_200_OK)
            else:
                return Response({'message': 'User not authenticated'},
status=status.HTTP_400_BAD_REQUEST)
        except Exception as e:
            return Response(str(e), status=status.HTTP_400_BAD_REQUEST)
```

4. CollegeERP



5. __pycache__



__init__.py

==========

Settings.py

```
"""
Django settings for erptest project.

Generated by 'django-admin startproject' using Django 2.1.2.

For more information on this file, see
https://docs.djangoproject.com/en/2.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/2.1/ref/settings/
"""

import os

# Build paths inside the project like this: os.path.join(BASE_DIR, ...)
BASE_DIR = os.path.dirname(os.path.dirname(os.path.abspath(__file__)))



# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/2.1/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'jy8c-n9y=pf##!2^jae-l_5iafq6q%wfq8gdb6c0r5d52su+9y'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True
```

```
ALLOWED_HOSTS = ['*']

AUTH_USER_MODEL = 'info.User'


# Application definition

INSTALLED_APPS = [
    'info.apps.InfoConfig',
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'djoser',
    'rest_framework.authtoken',
    'apis',

]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]
```

```
ROOT_URLCONF = 'CollegeERP.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [os.path.join(BASE_DIR, 'templates')],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'CollegeERP.wsgi.application'


# Database
# https://docs.djangoproject.com/en/2.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': os.path.join(BASE_DIR, 'db.sqlite3'),
    }
}
```

```
# Password validation
# https://docs.djangoproject.com/en/2.1/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]
```

```
# Internationalization
# https://docs.djangoproject.com/en/2.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_L10N = True

USE_TZ = True



# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/2.1/howto/static-files/

STATIC_URL = '/static/'

LOGIN_REDIRECT_URL = '/'

REST_FRAMEWORK = {
   'DEFAULT_PERMISSION_CLASSES': (
      'rest_framework.permissions.IsAuthenticated',
   ),
   'DEFAULT_AUTHENTICATION_CLASSES': (
      'rest_framework.authentication.TokenAuthentication',
      'rest_framework.authentication.SessionAuthentication',
   ),          }
```

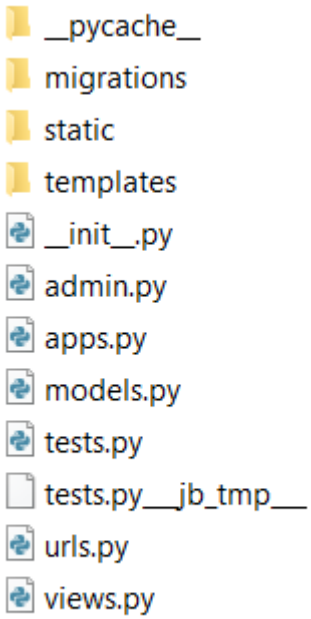
Urls.py

```
from django.contrib import admin
from django.urls import path, include
from django.contrib.auth import views as auth_views

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('info.urls')),
    path('info/', include('info.urls')),
    path('api/', include('apis.urls')),
    path('accounts/login/',
        auth_views.LoginView.as_view(template_name='info/login.html'), name='login'),
    path('accounts/logout/',
        auth_views.LogoutView.as_view(template_name='info/logout.html'), name='logout'),
]
```

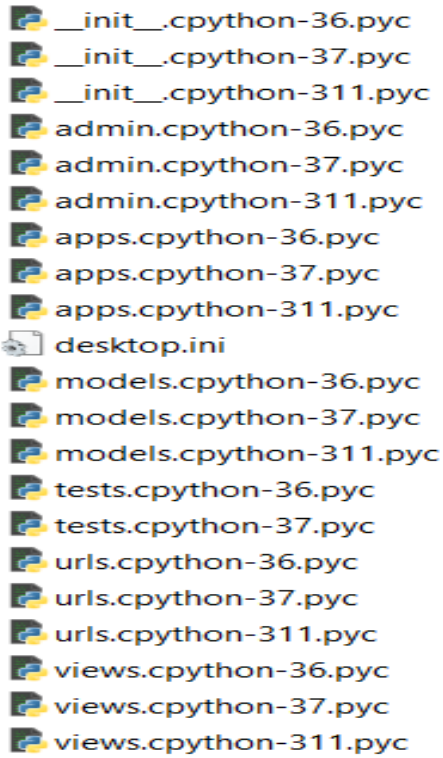
Wsgi.py

```
"""
WSGI config for CollegeERP project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/2.1/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'CollegeERP.settings')

application = get_wsgi_application()
```

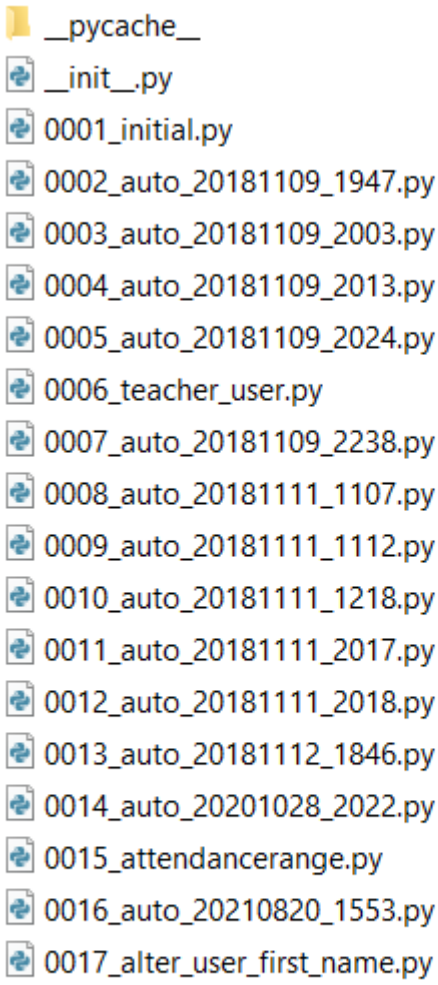
6. info



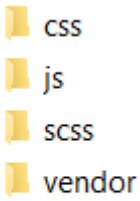
7. __pycache__



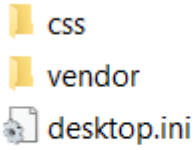
8. Migrations



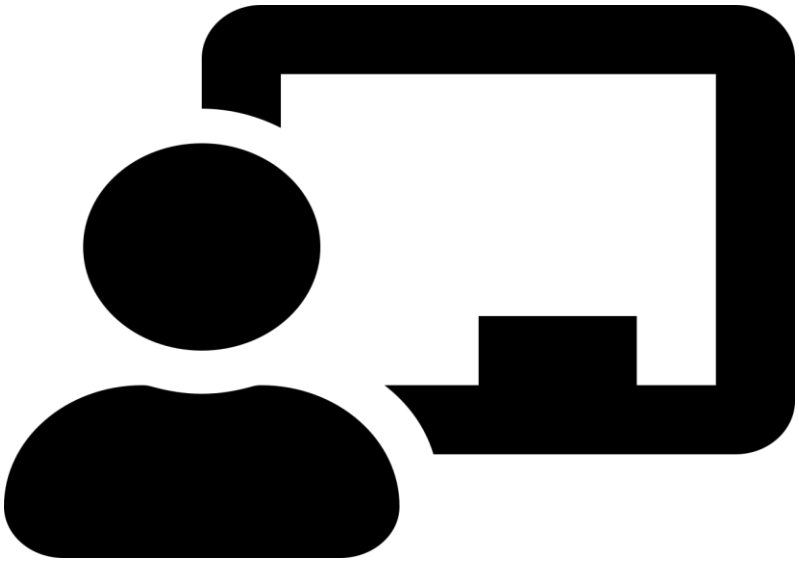
→ Static → info bootstrap



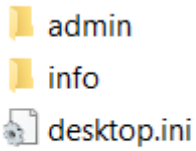
Homepage



Images



9. Templates



==========

__init__.py

==========

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='1998-06-06')

    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='1985-01-01')

    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='2025-09-09')
    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='rajnish'):
    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
+ '>'])



    #teacher

    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     Assign.objects.create(teacher=t, class_id=self.create_class(), course=self.create_course())
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     print(resp.content)
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")
```

```
    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")

    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")
```

==========

tests.py

___jb_tmp___

```
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='rajnish'):
        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
+ '>'])

    #teacher

    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     Assign.objects.create(teacher=t, class_id=self.create_class(), course=self.create_course())
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     print(resp.content)
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")

    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")

    # def test_attendance_class(self):
    #     t = self.create_teacher()
    #     self.client.login(username='test_user', password='test_password')
    #     resp = self.client.get(reverse('t_clas', args=(t.id, 1)))
    #     self.assertEqual(resp.status_code, 200)
    #     self.assertContains(resp, "Enter Attendance")
```

Urls.py

```
from django.urls import path, include
from . import views
from django.contrib import admin

urlpatterns = [
    path('', views.index, name='index'),
    path('student/<slug:stud_id>/attendance/',
        views.attendance, name='attendance'),
    path('student/<slug:stud_id>/<slug:course_id>/attendance/',
        views.attendance_detail, name='attendance_detail'),
    path('student/<slug:class_id>/timetable/',
        views.timetable, name='timetable'),
    # path('student/<slug:class_id>/search/', views.student_search, name='student_search'),

    path('student/<slug:stud_id>/marks_list/',
        views.marks_list, name='marks_list'),

    path('teacher/<slug:teacher_id>/<int:choice>/Classes/',
        views.t_clas, name='t_clas'),
    path('teacher/<int:assign_id>/Students/attendance/',
        views.t_student, name='t_student'),
    path('teacher/<int:assign_id>/ClassDates/',
        views.t_class_date, name='t_class_date'),
    path('teacher/<int:ass_c_id>/Cancel/',
        views.cancel_class, name='cancel_class'),
    path('teacher/<int:ass_c_id>/attendance/',
        views.t_attendance, name='t_attendance'),
    path('teacher/<int:ass_c_id>/Edit_att/', views.edit_att, name='edit_att'),
    path('teacher/<int:ass_c_id>/attendance/confirm/',
        views.confirm, name='confirm'),
```

```
    path('teacher/<slug:stud_id>/<slug:course_id>/attendance/',
        views.t_attendance_detail, name='t_attendance_detail'),
    path('teacher/<int:att_id>/change_attendance/',
        views.change_att, name='change_att'),
    path('teacher/<int:assign_id>/Extra_class/',
        views.t_extra_class, name='t_extra_class'),
    path('teacher/<slug:assign_id>/Extra_class/confirm/',
        views.e_confirm, name='e_confirm'),
    path('teacher/<int:assign_id>/Report/', views.t_report, name='t_report'),

    path('teacher/<slug:teacher_id>/t_timetable/',
        views.t_timetable, name='t_timetable'),
    path('teacher/<int:asst_id>/Free_teachers/',
        views.free_teachers, name='free_teachers'),

    path('teacher/<int:assign_id>/marks_list/',
        views.t_marks_list, name='t_marks_list'),
    path('teacher/<int:assign_id>/Students/Marks/',
        views.student_marks, name='t_student_marks'),
    path('teacher/<int:marks_c_id>/marks_entry/',
        views.t_marks_entry, name='t_marks_entry'),
    path('teacher/<int:marks_c_id>/marks_entry/confirm/',
        views.marks_confirm, name='marks_confirm'),
    path('teacher/<int:marks_c_id>/Edit_marks/',
        views.edit_marks, name='edit_marks'),
    path('api/auth/', include('djoser.urls')),
    path('add-teacher/', views.add_teacher, name='add_teacher'),
    path('add-student/', views.add_student, name='add_student'),
]
admin.site.site_url = None
admin.site.site_header = 'My Site'
```

Views.py

```
from django.shortcuts import render, get_object_or_404, redirect
from django.http import HttpResponseRedirect
from .models import Dept, Class, Student, Attendance, Course, Teacher, Assign, AttendanceTotal,
time_slots, \
    DAYS_OF_WEEK, AssignTime, AttendanceClass, StudentCourse, Marks, MarksClass
from django.urls import reverse
from django.utils import timezone
from django.contrib.auth.decorators import login_required
from django.contrib.auth import get_user_model


User = get_user_model()

# Create your views here.


@login_required
def index(request):
    if request.user.is_teacher:
        return render(request, 'info/t_homepage.html')
    if request.user.is_student:
        return render(request, 'info/homepage.html')
    if request.user.is_superuser:
        return render(request, 'info/admin_page.html')
    return render(request, 'info/logout.html')
```

```
@login_required()
def attendance(request, stud_id):
    stud = Student.objects.get(USN=stud_id)
    ass_list = Assign.objects.filter(class_id_id=stud.class_id)
    att_list = []
    for ass in ass_list:
        try:
            a = AttendanceTotal.objects.get(student=stud, course=ass.course)
        except AttendanceTotal.DoesNotExist:
            a = AttendanceTotal(student=stud, course=ass.course)
            a.save()
        att_list.append(a)
    return render(request, 'info/attendance.html', {'att_list': att_list})



@login_required()
def attendance_detail(request, stud_id, course_id):
    stud = get_object_or_404(Student, USN=stud_id)
    cr = get_object_or_404(Course, id=course_id)
    att_list = Attendance.objects.filter(course=cr, student=stud).order_by('date')
    return render(request, 'info/att_detail.html', {'att_list': att_list, 'cr': cr})



# Teacher Views

@login_required
def t_clas(request, teacher_id, choice):
    teacher1 = get_object_or_404(Teacher, id=teacher_id)
    return render(request, 'info/t_clas.html', {'teacher1': teacher1, 'choice': choice})
```

```
@login_required()
def t_student(request, assign_id):
    ass = Assign.objects.get(id=assign_id)
    att_list = []
    for stud in ass.class_id.student_set.all():
        try:
            a = AttendanceTotal.objects.get(student=stud, course=ass.course)
        except AttendanceTotal.DoesNotExist:
            a = AttendanceTotal(student=stud, course=ass.course)
            a.save()
        att_list.append(a)
    return render(request, 'info/t_students.html', {'att_list': att_list})


@login_required()
def t_class_date(request, assign_id):
    now = timezone.now()
    ass = get_object_or_404(Assign, id=assign_id)
    att_list = ass.attendanceclass_set.filter(date__lte=now).order_by('-date')
    return render(request, 'info/t_class_date.html', {'att_list': att_list})


@login_required()
def cancel_class(request, ass_c_id):
    assc = get_object_or_404(AttendanceClass, id=ass_c_id)
    assc.status = 2
    assc.save()
    return HttpResponseRedirect(reverse('t_class_date', args=(assc.assign_id,)))
```

```
@login_required()
def t_attendance(request, ass_c_id):
    assc = get_object_or_404(AttendanceClass, id=ass_c_id)
    ass = assc.assign
    c = ass.class_id
    context = {
        'ass': ass,
        'c': c,
        'assc': assc,
    }
    return render(request, 'info/t_attendance.html', context)


@login_required()
def edit_att(request, ass_c_id):
    assc = get_object_or_404(AttendanceClass, id=ass_c_id)
    cr = assc.assign.course
    att_list = Attendance.objects.filter(attendanceclass=assc, course=cr)
    context = {
        'assc': assc,
        'att_list': att_list,
    }
    return render(request, 'info/t_edit_att.html', context)


@login_required()
def confirm(request, ass_c_id):
    assc = get_object_or_404(AttendanceClass, id=ass_c_id)
    ass = assc.assign
    cr = ass.course
    cl = ass.class_id
    for i, s in enumerate(cl.student_set.all()):
```

```
            status = request.POST[s.USN]
            if status == 'present':
                status = 'True'
            else:
                status = 'False'
            if assc.status == 1:
                try:
                    a = Attendance.objects.get(course=cr, student=s, date=assc.date, attendanceclass=assc)
                    a.status = status
                    a.save()
                except Attendance.DoesNotExist:
                    a = Attendance(course=cr, student=s, status=status, date=assc.date, attendanceclass=assc)
                    a.save()
            else:
                a = Attendance(course=cr, student=s, status=status, date=assc.date, attendanceclass=assc)
                a.save()
                assc.status = 1
                assc.save()

        return HttpResponseRedirect(reverse('t_class_date', args=(ass.id,)))



@login_required()
def t_attendance_detail(request, stud_id, course_id):
    stud = get_object_or_404(Student, USN=stud_id)
    cr = get_object_or_404(Course, id=course_id)
    att_list = Attendance.objects.filter(course=cr, student=stud).order_by('date')
    return render(request, 'info/t_att_detail.html', {'att_list': att_list, 'cr': cr})
```

```
@login_required()
def change_att(request, att_id):
    a = get_object_or_404(Attendance, id=att_id)
    a.status = not a.status
    a.save()
    return HttpResponseRedirect(reverse('t_attendance_detail', args=(a.student.USN, a.course_id)))


@login_required()
def t_extra_class(request, assign_id):
    ass = get_object_or_404(Assign, id=assign_id)
    c = ass.class_id
    context = {
        'ass': ass,
        'c': c,
    }
    return render(request, 'info/t_extra_class.html', context)


@login_required()
def e_confirm(request, assign_id):
    ass = get_object_or_404(Assign, id=assign_id)
    cr = ass.course
    cl = ass.class_id
    assc = ass.attendanceclass_set.create(status=1, date=request.POST['date'])
    assc.save()

    for i, s in enumerate(cl.student_set.all()):
        status = request.POST[s.USN]
        if status == 'present':
            status = 'True'
        else:
```

```
            status = 'False'
        date = request.POST['date']
        a = Attendance(course=cr, student=s, status=status, date=date, attendanceclass=assc)
        a.save()

    return HttpResponseRedirect(reverse('t_clas', args=(ass.teacher_id, 1)))


@login_required()
def t_report(request, assign_id):
    ass = get_object_or_404(Assign, id=assign_id)
    sc_list = []
    for stud in ass.class_id.student_set.all():
        a = StudentCourse.objects.get(student=stud, course=ass.course)
        sc_list.append(a)
    return render(request, 'info/t_report.html', {'sc_list': sc_list})


@login_required()
def timetable(request, class_id):
    asst = AssignTime.objects.filter(assign__class_id=class_id)
    matrix = [[" for i in range(12)] for j in range(6)]

    for i, d in enumerate(DAYS_OF_WEEK):
        t = 0
        for j in range(12):
            if j == 0:
                matrix[i][0] = d[0]
                continue
            if j == 4 or j == 8:
                continue
            try:
                a = asst.get(period=time_slots[t][0], day=d[0])
```

```
                matrix[i][j] = a.assign.course_id
            except AssignTime.DoesNotExist:
                pass
            t += 1

    context = {'matrix': matrix}
    return render(request, 'info/timetable.html', context)


@login_required()
def t_timetable(request, teacher_id):
    asst = AssignTime.objects.filter(assign__teacher_id=teacher_id)
    class_matrix = [[True for i in range(12)] for j in range(6)]
    for i, d in enumerate(DAYS_OF_WEEK):
        t = 0
        for j in range(12):
            if j == 0:
                class_matrix[i][0] = d[0]
                continue
            if j == 4 or j == 8:
                continue
            try:
                a = asst.get(period=time_slots[t][0], day=d[0])
                class_matrix[i][j] = a
            except AssignTime.DoesNotExist:
                pass
            t += 1

    context = {
        'class_matrix': class_matrix,
    }
    return render(request, 'info/t_timetable.html', context)
```

```
@login_required()
def free_teachers(request, asst_id):
    asst = get_object_or_404(AssignTime, id=asst_id)
    ft_list = []
    t_list = Teacher.objects.filter(assign__class_id__id=asst.assign.class_id_id)
    for t in t_list:
        at_list = AssignTime.objects.filter(assign__teacher=t)
        if not any([True if at.period == asst.period and at.day == asst.day else False for at in at_list]):
            ft_list.append(t)

    return render(request, 'info/free_teachers.html', {'ft_list': ft_list})
```

 # student marks

```
@login_required()
def marks_list(request, stud_id):
    stud = Student.objects.get(USN=stud_id, )
    ass_list = Assign.objects.filter(class_id_id=stud.class_id)
    sc_list = []
    for ass in ass_list:
        try:
            sc = StudentCourse.objects.get(student=stud, course=ass.course)
        except StudentCourse.DoesNotExist:
            sc = StudentCourse(student=stud, course=ass.course)
            sc.save()
            sc.marks_set.create(type='I', name='Internal test 1')
            sc.marks_set.create(type='I', name='Internal test 2')
            sc.marks_set.create(type='I', name='Internal test 3')
            sc.marks_set.create(type='E', name='Event 1')
            sc.marks_set.create(type='E', name='Event 2')
```

```
            sc.marks_set.create(type='S', name='Semester End Exam')
        sc_list.append(sc)

    return render(request, 'info/marks_list.html', {'sc_list': sc_list})
```

 # teacher marks

```
@login_required()
def t_marks_list(request, assign_id):
    ass = get_object_or_404(Assign, id=assign_id)
    m_list = MarksClass.objects.filter(assign=ass)
    return render(request, 'info/t_marks_list.html', {'m_list': m_list})


@login_required()
def t_marks_entry(request, marks_c_id):
    mc = get_object_or_404(MarksClass, id=marks_c_id)
    ass = mc.assign
    c = ass.class_id
    context = {
        'ass': ass,
        'c': c,
        'mc': mc,
    }
    return render(request, 'info/t_marks_entry.html', context)
```

```
@login_required()
def marks_confirm(request, marks_c_id):
    mc = get_object_or_404(MarksClass, id=marks_c_id)
    ass = mc.assign
    cr = ass.course
    cl = ass.class_id
    for s in cl.student_set.all():
        mark = request.POST[s.USN]
        sc = StudentCourse.objects.get(course=cr, student=s)
        m = sc.marks_set.get(name=mc.name)
        m.marks1 = mark
        m.save()
    mc.status = True
    mc.save()

    return HttpResponseRedirect(reverse('t_marks_list', args=(ass.id,)))



@login_required()
def edit_marks(request, marks_c_id):
    mc = get_object_or_404(MarksClass, id=marks_c_id)
    cr = mc.assign.course
    stud_list = mc.assign.class_id.student_set.all()
    m_list = []
    for stud in stud_list:
        sc = StudentCourse.objects.get(course=cr, student=stud)
        m = sc.marks_set.get(name=mc.name)
        m_list.append(m)
    context = {
        'mc': mc,
        'm_list': m_list,
    }
    return render(request, 'info/edit_marks.html', context)
```

```
@login_required()
def student_marks(request, assign_id):
    ass = Assign.objects.get(id=assign_id)
    sc_list = StudentCourse.objects.filter(student__in=ass.class_id.student_set.all(),
course=ass.course)
    return render(request, 'info/t_student_marks.html', {'sc_list': sc_list})



@login_required()
def add_teacher(request):
    if not request.user.is_superuser:
        return redirect("/")

    if request.method == 'POST':
        dept = get_object_or_404(Dept, id=request.POST['dept'])
        name = request.POST['full_name']
        id = request.POST['id'].lower()
        dob = request.POST['dob']
        sex = request.POST['sex']

        # Creating a User with teacher username and password format
        # USERNAME: firstname + underscore + unique ID
        # PASSWORD: firstname + underscore + year of birth(YYYY)
        user = User.objects.create_user(
            username=name.split(" ")[0].lower() + '_' + id,
            password=name.split(" ")[0].lower() + '_' + dob.replace("-","")[:4]
        )
        user.save()

        Teacher(
            user=user,
            id=id,
```

```
            dept=dept,
            name=name,
            sex=sex,
            DOB=dob
        ).save()
        return redirect('/')
    all_dept = Dept.objects.order_by('-id')
    context = {'all_dept': all_dept}
    return render(request, 'info/add_teacher.html', context)
@login_required()
def add_student(request):
    # If the user is not admin, they will be redirected to home
    if not request.user.is_superuser:
        return redirect("/")

    if request.method == 'POST':
        # Retrieving all the form data that has been inputted
        class_id = get_object_or_404(Class, id=request.POST['class'])
        name = request.POST['full_name']
        usn = request.POST['usn']
        dob = request.POST['dob']
        sex = request.POST['sex']

        # Creating a User with student username and password format
        # USERNAME: firstname + underscore + last 3 digits of USN
        # PASSWORD: firstname + underscore + year of birth(YYYY)
        user = User.objects.create_user(
            username=name.split(" ")[0].lower() + '_' + request.POST['usn'][-3:],
            password=name.split(" ")[0].lower() + '_' + dob.replace("-","")[:4]
        )
        user.save()
```

```
    # Creating a new student instance with given data and saving it.
    Student(
        user=user,
        USN=usn,
        class_id=class_id,
        name=name,
        sex=sex,
        DOB=dob
    ).save()
    return redirect('/')

all_classes = Class.objects.order_by('-id')
context = {'all_classes': all_classes}
return render(request, 'info/add_student.html', context)
```

Usage

Go to the College-ERP folder and run

python manage.py runserver

Then go to the browser and enter the url http://127.0.0.1:8000/

Login

The login page is common for students and teachers.

You can access the django admin page at http://127.0.0.1:8000/admin and login with username 'admin' and the above password.

Also a new admin user can be created using

python manage.py createsuperuser

Users

New students and teachers can be added through the admin page. A new user needs to be created for each.

The admin page is used to modify all tables such as Students, Teachers, Departments, Courses, Classes etc.

For more details regarding the system and features please refer the reports included.

Added method to reset attendance time range in Django Admin page.

alt_text

This is present in Django Admin -> Attendance (http://127.0.0.1:8000/admin/info/attendanceclass/).

Start Date: Start Date of Attendance period

End Date: End Date of Attendance period

This will delete all present attendance data and create new attendance objects for the given time range.

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)
```

**** Some Important Commands for use: **********

```
python manage.py migrate
python manage.py makemigrations
python manage.py createsuperuser
python manage.py runserver
```

7. TESTING

7.1. Testing Confirmation and Methods

College ERP system is vast with a lot of asked features and functionality. Each stakeholder gives their list of conditions. As a design with four group members, we don't have the coffers and tools to apply all the conditions. Therefore, it's essential to find a balance among the colorful stakeholders where they can be satisfied with the outgrowth of the design. This is achieved through concession among the colorful classes of stakeholders.

It's in our interest to develop a Web app that's functional, dependable, harmonious and easy to use. We collected the conditions from the different stakeholders that include, the tutoring staff, specialized staff, scholars and the administration. We reviewed the list of conditions and made a list of doable and non-feasible conditions. We meet the stakeholders again and explain why some conditions weren't doable. For illustration, the leave module for the preceptors can't to enforce as that point has a lot functionality that's beyond the compass of this design.

We also set up that some conditions from different stakeholders were clashing. For illustration, the scholars had requested for an option to appeal wrong marking of attendance or marks by the schoolteacher. But the preceptors were against this point as that would increase the burden on the preceptors and there was also a possibility of false operation of this point by the scholars. Considering both perspectives, we decided to agree with the preceptors as this point would disaffect preceptors. The scholars were given the reason for not including their demand and an agreement was reached.

The scholars wanted a social media type point enforced on the ERP where the scholars from the council can communicate with each other and have a feed of the events in the council. While the point would have been nice to see, it was beyond the compass of this design. We stated that such an advanced interpretation of the conditions wasn't possible. But, a perpetration of the point on a lower compass with lower functionality was possible. Thus, we negotiated the features until both parties were satisfied.

A. Testing Confirmation

Conditions confirmation examines the specification to ensure that all software conditions have been stated unambiguously, so that inconsistencies, deletions, and crimes have been detected and corrected.
This roster is a list of questions that helps us to validate our conditions. They're as follows Are conditions stated easily?

Can they be misinterpreted?

There will be a chance of misinterpreting the conditions specified by the stakeholders. But we've collected conditions from numerous sources and those conditions are understood rightly.

Is the source (e.g., a person, a regulation, a document) of the demand linked? Has the final statement of the demand been examined by or against the original source?

All the sources of the conditions are rightly linked. And all the conditions are vindicated.

Does the demand violate any system sphere constraints?

Those conditions violating the system sphere constraints were neglected during the concession of conditions. So, no conditions are violating the system sphere constraint.

Is the demand testable?

All the conditions collected are unequivocal, clear and precise. This makes the conditions testable.

Is the demand traceable to any system model that has been created?

The demand is traceable i.e., the capability to describe and follow the life of a demand in both a forwards and backwards direction. (i.e., from its origins, through its development and specification, to its posterior deployment and use, and through ages of ongoing refinement and replication in any of these phases)

Conditions Management

Conditions operation can be defined as a process of inspiring, establishing, organizing, and control- ling changes to the conditions. Generally, the process of conditions operation begins as soon as the conditions document is available, but' planning' for managing the changing conditions should start during the condition's elicitation process.

The essential conditioning performed in conditions operation is listed below.

1. Feting the need for change in the conditions
2. Establishing a relationship amongst stakeholders and involving them in the conditions masterminding process.
3. Relating and tracking conditions attributes.

Conditions operation enables the development platoon to identify, control, and track conditions and changes that do as the software development process progresses. Other advantages associated with the condition's operation are listed below.

More control of complex systems this provides the development platoon with a clear understanding of what, when, and why the software is to be delivered. The coffers are allocated according to stoner- driven precedence's and relative perpetration trouble.

Advanced software quality this ensures that the software performs according to enhance software quality. This can be achieved when the inventors and testers have a precise understanding of what to develop and test.

Reduced design costs and detainments this minimizes crimes beforehand in the development cycle as it's precious to' fix' crimes at the after stages of the development cycle. As a result, the design costs also reduce.

Advanced platoon communication this facilitates early involvement of druggies to ensure that their requirements are achieved.

Conditions Change Operation

Conditions change operation is used when there's a request or offer for a change in the requirements. The advantage of this process is that the changes to the proffers are managed constantly and in a controlled manner. Note that numerous conditionings of conditions operation are like software configuration operation conditioning.

An effective demand change operation process undergoes a number of stages for changes to the conditions. These stages are listed below-

Problem analysis and change specification the entire process begins with identification of problems to the conditions. The problem or offer is anatomized to corroborate whether the change is valid. The outgrowth of the analysis is handed to the' change panhandler' and a more specific conditions change offer is also made.

Change analysis and going the effect of a change requested on the demand is assessed according to traceability information. The cost for this can be estimated on the base of modification made to the design and perpetration. After the analysis is over, a decision is made whether changes are to be made.

Change perpetration eventually, the changes are made to the conditions document, system design and perpetration. The conditions document is organized in such a manner so that changes to it can be made without expansive rewriting. Minimizing the external references and making document sections modular achieves insecurity in the document. By doing this, individual sections can be changed and replaced without affecting another corridor of the document.

B. Testing Methods

Software testing methods are traditionally divided into black box testing and white box testing. These two approaches are used to describe the point of view that a test engineer takes when designing test cases.

White Box Testing

White box testing, by contrast to black box testing, is when the tester has access to the internal data structures and algorithms (and the code that implement these). White box testing methods can also be used to evaluate the completeness of a test suite that was created with black box testing methods. This allows the software team to examine parts of a system that are rarely tested and ensures that the most important function points have been tested.

This project is implemented using python with the Django framework. The code consists of models and views which can be tested. Models define the tables stored in SQL and the relationship between the different tables using foreign keys. A view function, or "view" for short, is simply a Python function that takes a web request and returns a web response. This response can be the HTML contents of a Web page, or a redirect, or a 404 error, or an XML document, or an image, etc.

Python also provides a file called test.py where we can write unit tests for the models and views. This is very useful as it automates the testing and we no longer have to manually test every page after there were any changes. The python code is pasted below and each test is explained using comments in the code.

Black Box Testing

Black box testing treats the software as a "black box" without any knowledge of internal implementation. Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

We performed black box testing on the teacher page to make sure every page was working as desired. We took into consideration various test cases and noted down the results. Below we have recorded various test cases and their respective results.

Test Case: 1

- Request the attendance page for a teacher with no assigned classes.
- The web page loaded with message “Teacher has no classes assigned”.

Test Case: 2

- Request the attendance page for a teacher with 1 assigned class.
- The web page displayed the assigned class and options to enter attendance and view the students.

Test Case: 3

- Request to enter the attendance for an assigned class with one test student
- The web page displays the student with his/her details and an option to mark present or absent. On marking absent, it can be viewed by the student.

Test Case: 4

- Request to edit the attendance for an assigned class with one test student
- The student is listed with his/her details and is initially marked as absent from the previous test. On marking present, the attendance for that student and can be viewed by the student.

Test Case: 5

- Request to enter the marks for an assigned class with one student
- Initially, a list of tests displays such as internals 1, SEE etc. On selecting one of internals 1, the teacher can enter the marks for the student out of 20. On submitting, the status for that test turns green denoting that it has been successfully entered.

Test Case: 6

- Request to edit the marks for an assigned class with one student
- For each class, there is a list of tests such as internals 1, SEE etc. As the marks for internals 1 was already entered in the previous test, it is marked green and there is an option to edit. When editing, the marks already stored is displayed and appropriate changes can be made and saved.

Test Case: 7

- Request to view the student information for an assigned class with no students
- The requested page is display with no content and a message stating "This class has no students assigned"

Test Case: 8

- Request to view the student information for an assigned class with 1 student
- The web page is the form of a table with entries for student name, USN and their attendance percentage, marks in each test including 3 internals, 2 events and 1 SEE. IF the attendance status is below 75%, it is marked in red.

Acceptance Testing

Acceptance testing performed by the customer is known as user acceptance testing (UAT). Since our project is on college management system, the teachers are a key stakeholder. Hence, it was important to allow the teachers to test the software and get their approval as they intend to use the software the most. Therefore, we met and a gave a demonstration of the project to our teacher Dr. Trisiladevi C. Nagavi. We showed her all the features and functionality of the website. She went through all the different web pages and asked several questions on the working of the code. Overall, she was happy with the working and results of the software.

7.2. System Testing and Results Analysis

The completion of a system will be achieved only after it has been thoroughly tested. Though this gives a feel the project is completed, there cannot be any project without going through this stage. Hence in this stage it is decided whether the project can undergo the real time environment execution without any break downs, therefore a package can be rejected even at this stage.

Results of Testing

After applying various testing methods such as black box testing, white box testing and acceptance testing, we can conclude that the testing for the software is completed. To summarize the testing phase, white box testing is done using the inbuilt feature of Django to apply unit tests to all the components in the software. After any changes to the software, we can run the tests on the software automatically and thus we can find and eliminate any bugs or errors in the system easily instead of performing rigorous manual testing after every change.

In black box testing, we testing all the components and system as a whole. Several test cases were considered and extensive tests were conducted. The results of these tests were positive and any errors were fixed during the testing phase.

For acceptance testing, we gave a demonstration of the software to our teacher, who is a key stake- holder. After several tests and questions, she was content with results of the tests and software.

8. SYSTEM SECURITY MEASURES

The College ERP System handles sensitive academic and personal data for students, teachers, and staff. Therefore, a multi-layered security approach has been implemented to ensure Confidentiality, Integrity, and Availability (CIA triad) of the information. The security measures are ingrained at every level of the application's architecture.

1. **Authentication & Access Control**

This is the primary layer of defense, ensuring only authorized users can access the system.

Role-Based Access Control (RBAC)

The system's core security model is RBAC. Users are categorized into distinct roles:

Students

Can only view their own personal information, attendance, marks, and timetable. They have no edit privileges.

Teachers

Can view and edit data (attendance, marks) but only for the students and courses they are assigned to. They cannot access data outside their jurisdiction.

Administrators

Have full CRUD (Create, Read, Update, Delete) capabilities across all database tables.

Superusers (Django Admin)

Have ultimate system-level access, managed via Django's built-in admin interface.

Strong Password Policy

While the initial password is generated (e.g., firstname_DOB), the use of Django's built-in User model allows for the enforcement of strong password policies through AUTH_PASSWORD_VALIDATORS in settings.py, which includes checks for minimum length, common passwords, and purely numeric passwords.

Django Authentication Framework

The system leverages Django's robust and battle-tested authentication system to handle user sessions, login, and logout securely, preventing common vulnerabilities like session hijacking.

2. **Data Security & Integrity**

These measures protect the data itself from corruption and unauthorized modification.

Database Constraints

The data models are designed with strict constraints at the database level.

Primary Keys & Unique Constraints

Fields like USN (Student) and id (Teacher) are enforced as unique to prevent duplicates.

Foreign Key Constraints

Relationships (e.g., a student belongs to a Class) are enforced with foreign keys, maintaining referential integrity and preventing orphaned records.

Validation Rules

Models use validators (e.g., MinValueValidator, MaxValueValidator for marks) to ensure data entered is within logical bounds before it even reaches the database.

SQL Injection Prevention

By using Django's Object-Relational Mapper (ORM) for all database queries, the system is inherently protected against SQL injection attacks. The ORM escapes parameters automatically, ensuring user input is treated as data, not executable code.

3. **Application-Level Security**

These measures protect the application logic and handling of user requests.

Cross-Site Request Forgery (CSRF) Protection:

Django provides automatic CSRF protection for all state-changing HTTP requests (POST, PUT, DELETE). The CsrfViewMiddleware ensures that requests originate from the application's own forms, preventing malicious sites from performing actions on behalf of an authenticated user.

Cross-Site Scripting (XSS) Prevention

Django templates automatically escape variables, meaning HTML tags input by users are rendered as harmless text rather than executable code. This prevents malicious scripts from being injected into pages viewed by other users.

Authorization Checks in Views

Every view function is decorated with @login_required. Furthermore, critical views include additional checks to ensure a user can only access data they are permitted to. For example, the attendance_detail view checks that a student is only trying to view their own attendance records.

4. **API Security**

For the REST API endpoints (/api/), additional security measures are in place:

Token Authentication

The API uses Django REST Framework's Token Authentication scheme. Clients must include a valid token in the header of every HTTP request to access API resources. This is a stateless and secure method for programmatic access.

Permission Classes

API views explicitly set permission_classes = [IsAuthenticated], ensuring anonymous requests are rejected. The API logic also includes checks to ensure a user can only retrieve their own data.

5. **Administrative Security**

Django Admin Interface

The admin site (/admin/) is a powerful tool and is protected by the same authentication system. Access is restricted only to users with the is_staff or is_superuser flag set to True.

Audit Trail

While not explicitly detailed, the database design allows for an audit trail. For instance, the Attendance and Marks tables store historical records, making it possible to track changes over time.

Screenshot & System Implementation.

The College ERP System employs a defense-in-depth strategy, integrating security at the authentication, data, application, and API layers. By leveraging Django's built-in security features and adhering to best practices in database and application design, the system effectively mitigates common web application vulnerabilities and ensures the protection of sensitive academic information.

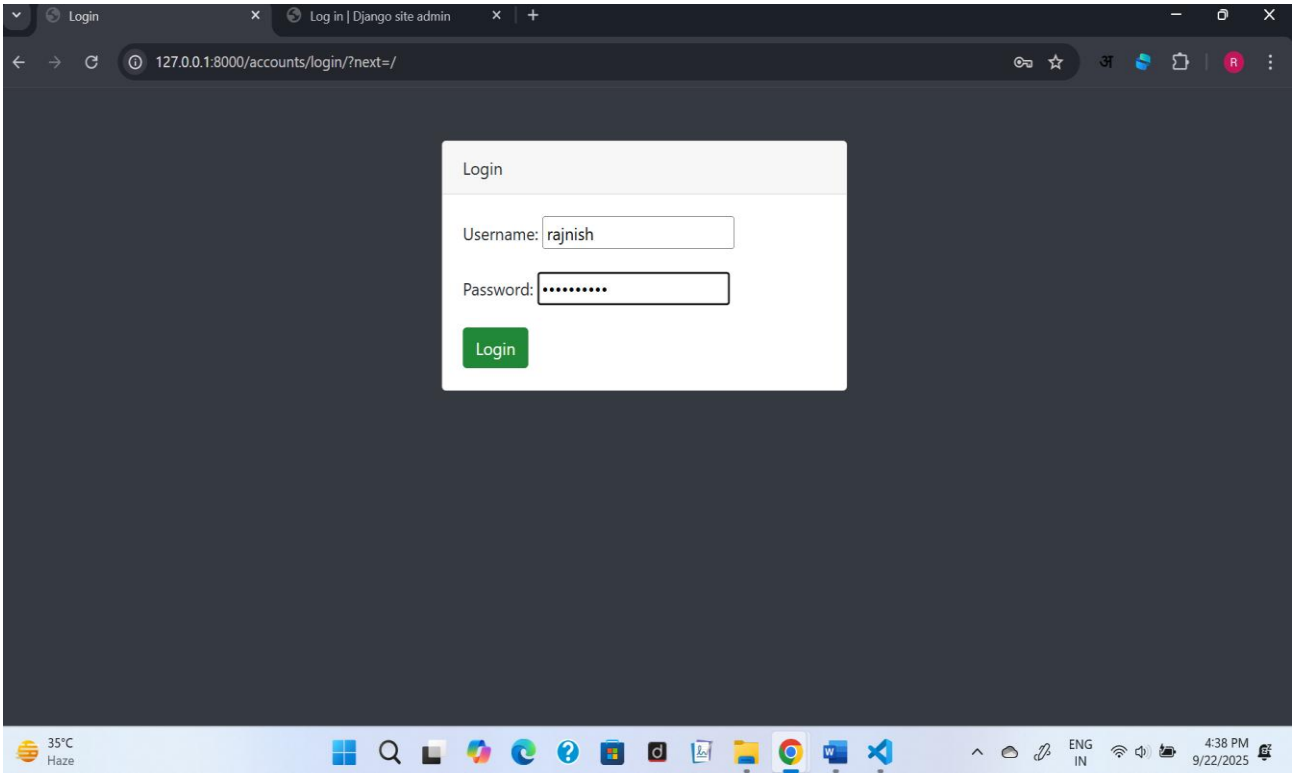
9. OUTPUT AND REPORTS

A. Output

The council ERP system has three main stoner classes. These include the scholars, preceptors and announcement- administrator. This section will explain in detail all the features and the working of those for each stoner class.

Student Login

Each pupil in the council is assigned a unique username and word by the director. The stoner- name is the same as their USN and so is the word. They may change it latterly according to their want.

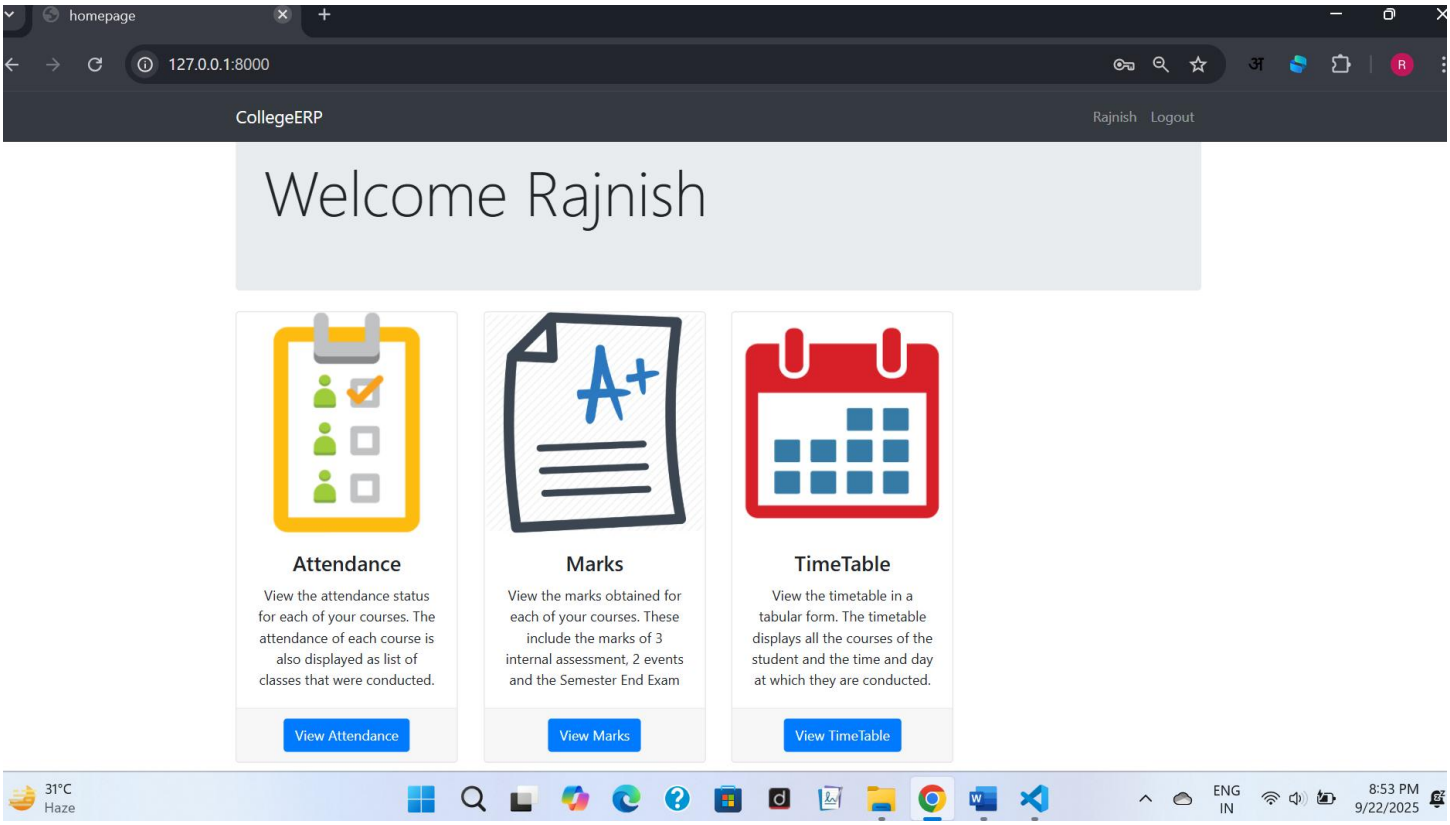


Student Login Page

Homepage

After successful login, the pupil is presented a homepage with their main sections, attendance, marks and schedule. In the attendance section the pupil can view their attendance status which includes the total classes, attended classes and the attendance chance for each of their courses.

In the marks section, the pupil 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. Incipiently, the schedule provides the classes assigned to that pupil and day and time of each in an irregular form.



Student Home Page

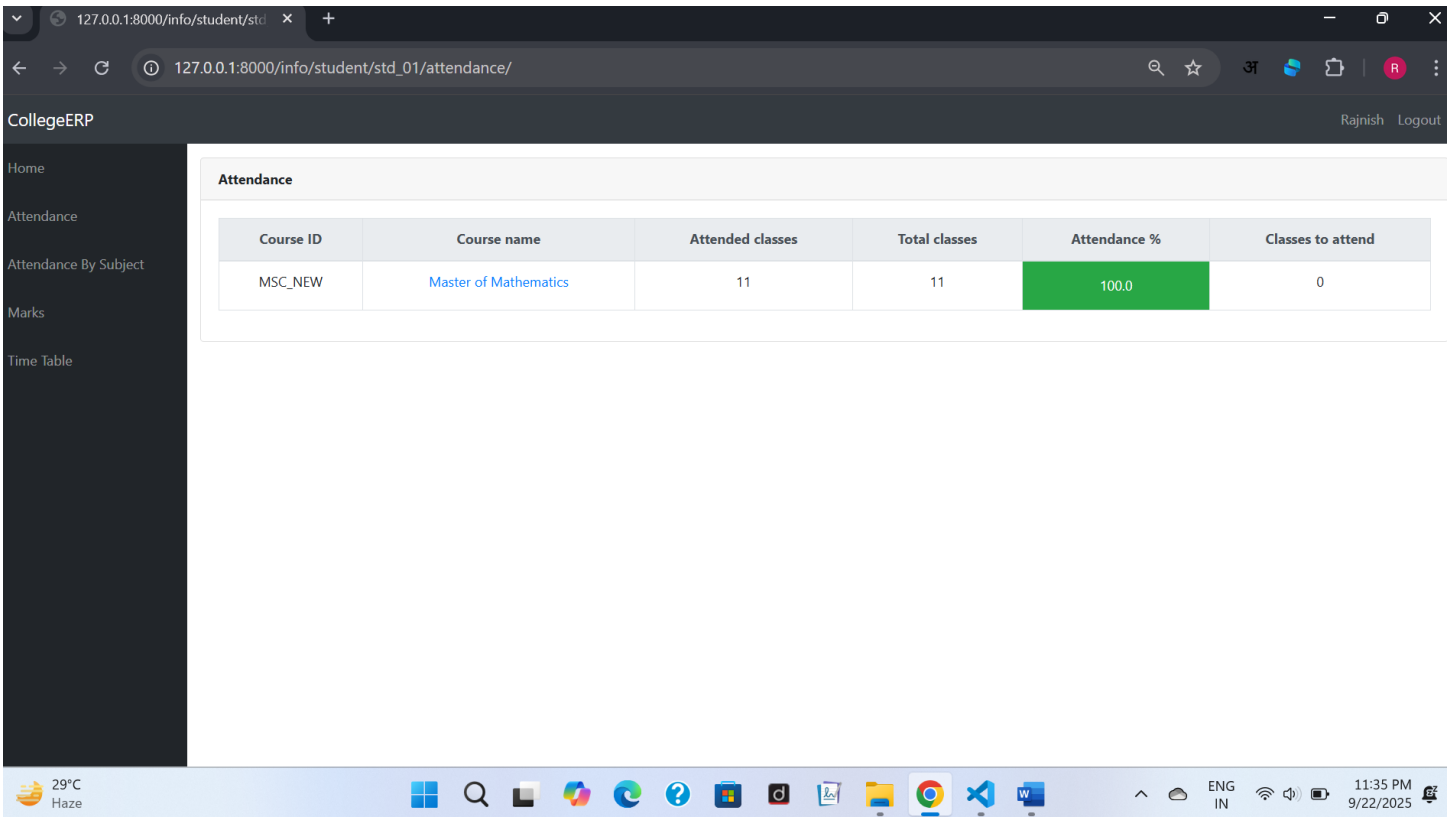
Attendance

On the attendance runner, there's a list of courses that's dependent on each pupil. For each course, the course id and name are display along with the attended classes, total classes and the attendance chance for that particularcourse.

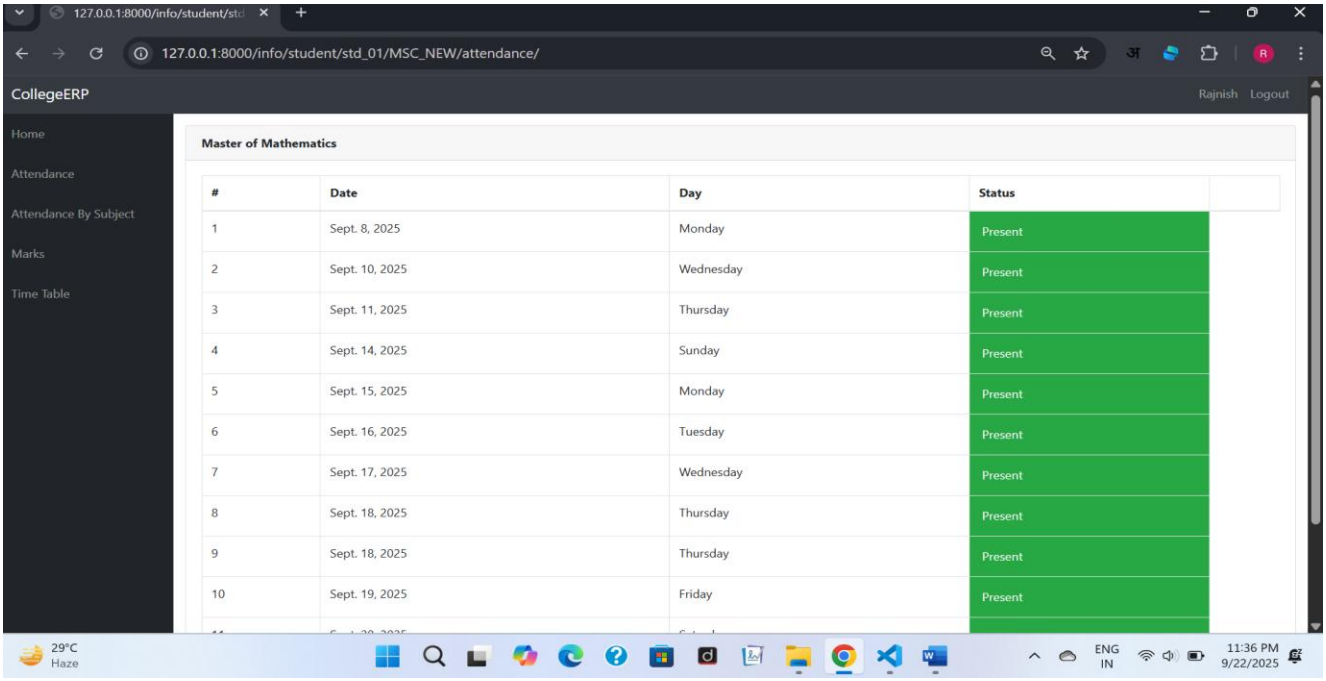
However, it's displayed in red denoting deficit of attendance; else it's green, if the attendance chance is below 75 for anycourse. However, it specifies the number of classes to attend to make up for it, if there's any shortage. However, it takes you to the attendance detail runner, if you click on each course.

Attendance Detail

This runner displays further details for the attendance in each course. For each the course, there's a list of classes conducted and each is marked with the date, day and whether the pupil was present or absent on that particular date.



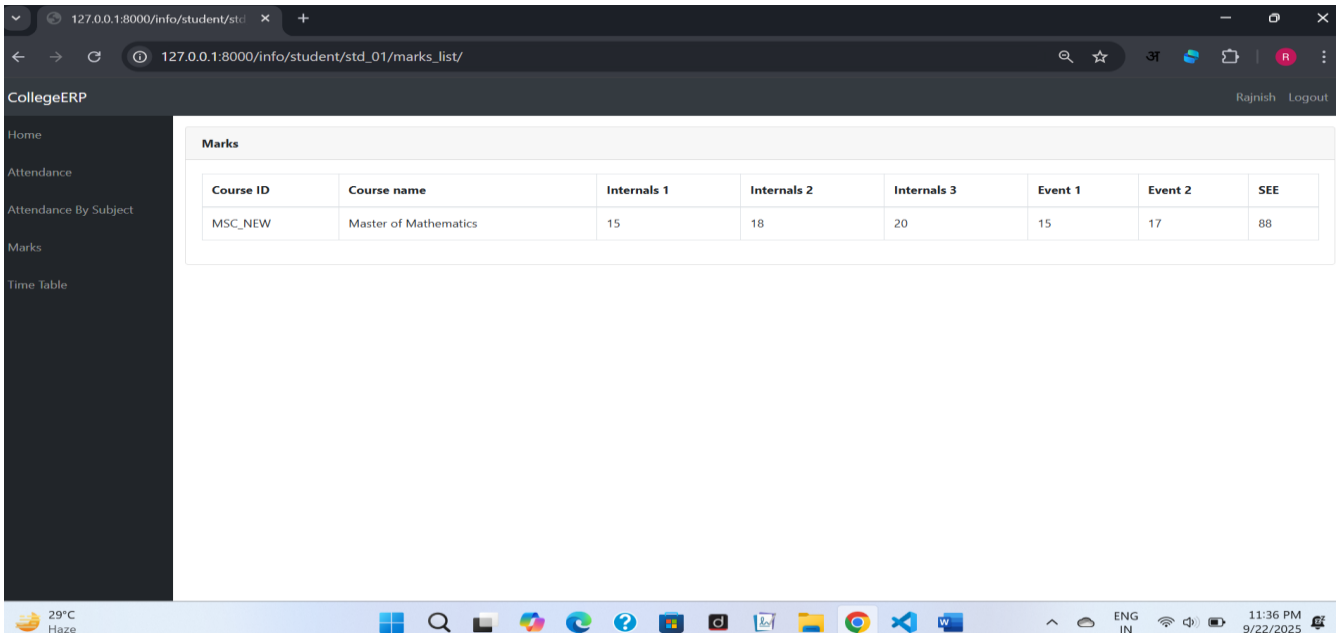
Student Attendance Page



Student Attendance Detail Page

Marks

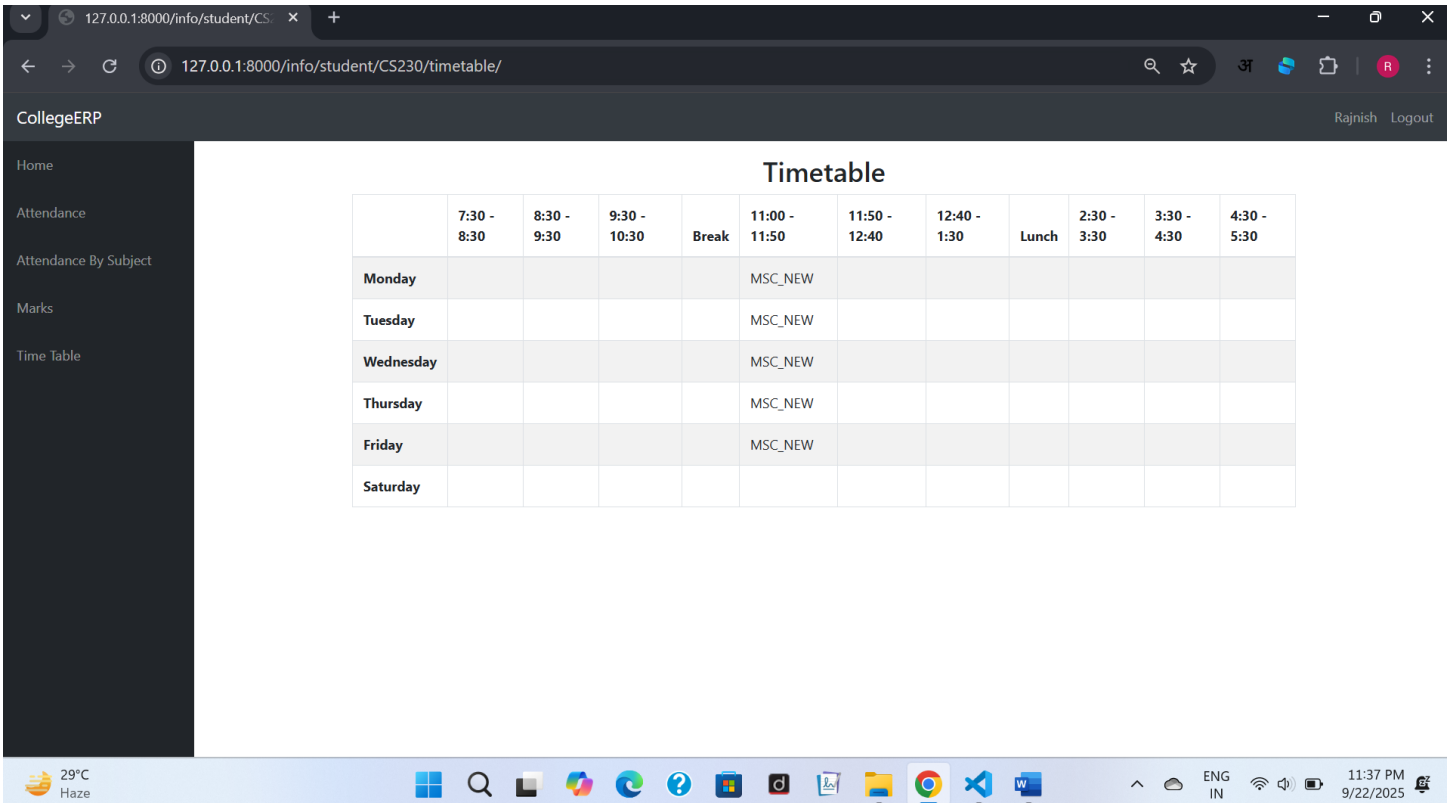
The Marks runner is a table with an entry for each of their courses. The course id and name are specified along the marks attained in each of the tests and examinations. The tests include 3 internal assessments with marks attained out of a aggregate of 20, 2 events similar as design, assignment, quiz, etc. with marks out of. Incipiently, one semester end test with marks out of 100.



Student Marks Page

Timetable

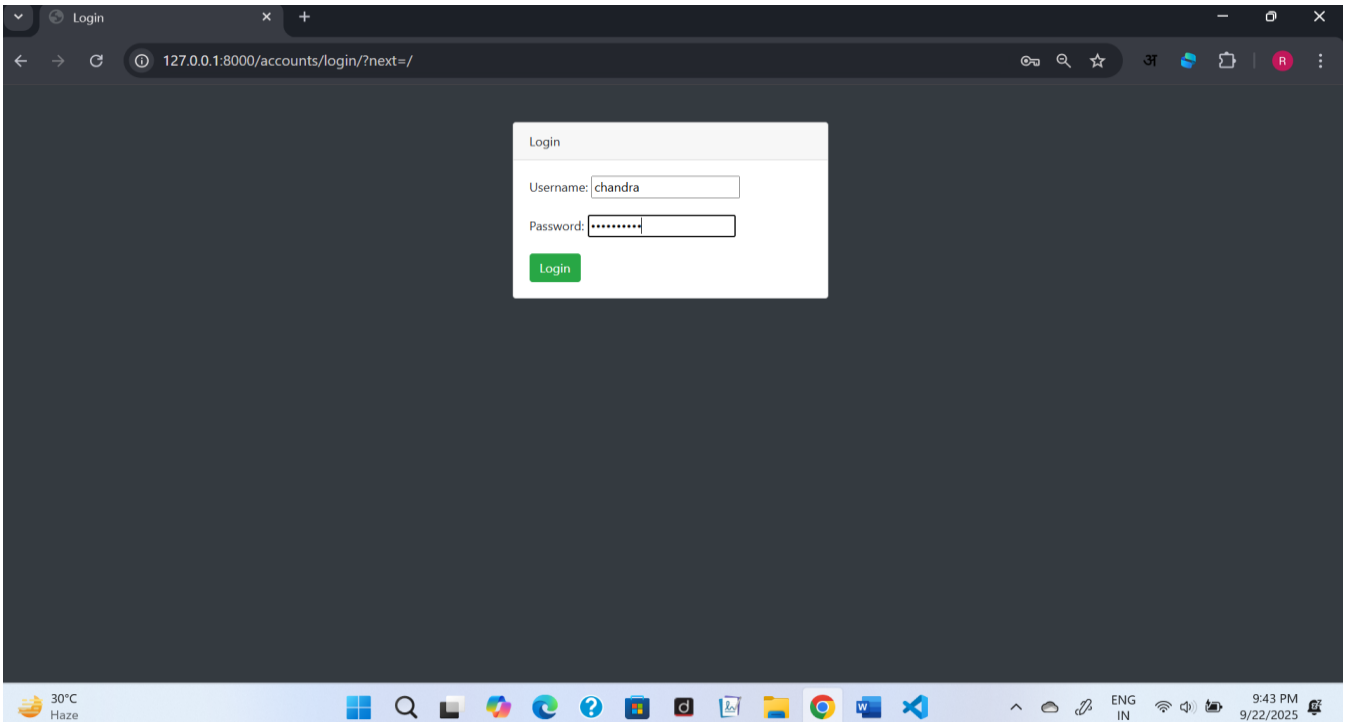
This runner is a table which lists the day and timings of each of the classes assigned to the pupil. The row heads are the days of the week and the column heads are the time places. So, for each day, it specifies the classes in the time places. The schedule is generated automatically from the assign table, which a table containing the information of all the preceptors is assigned to a class with a course and the timings the classes.



Student Timetable

Teacher Login

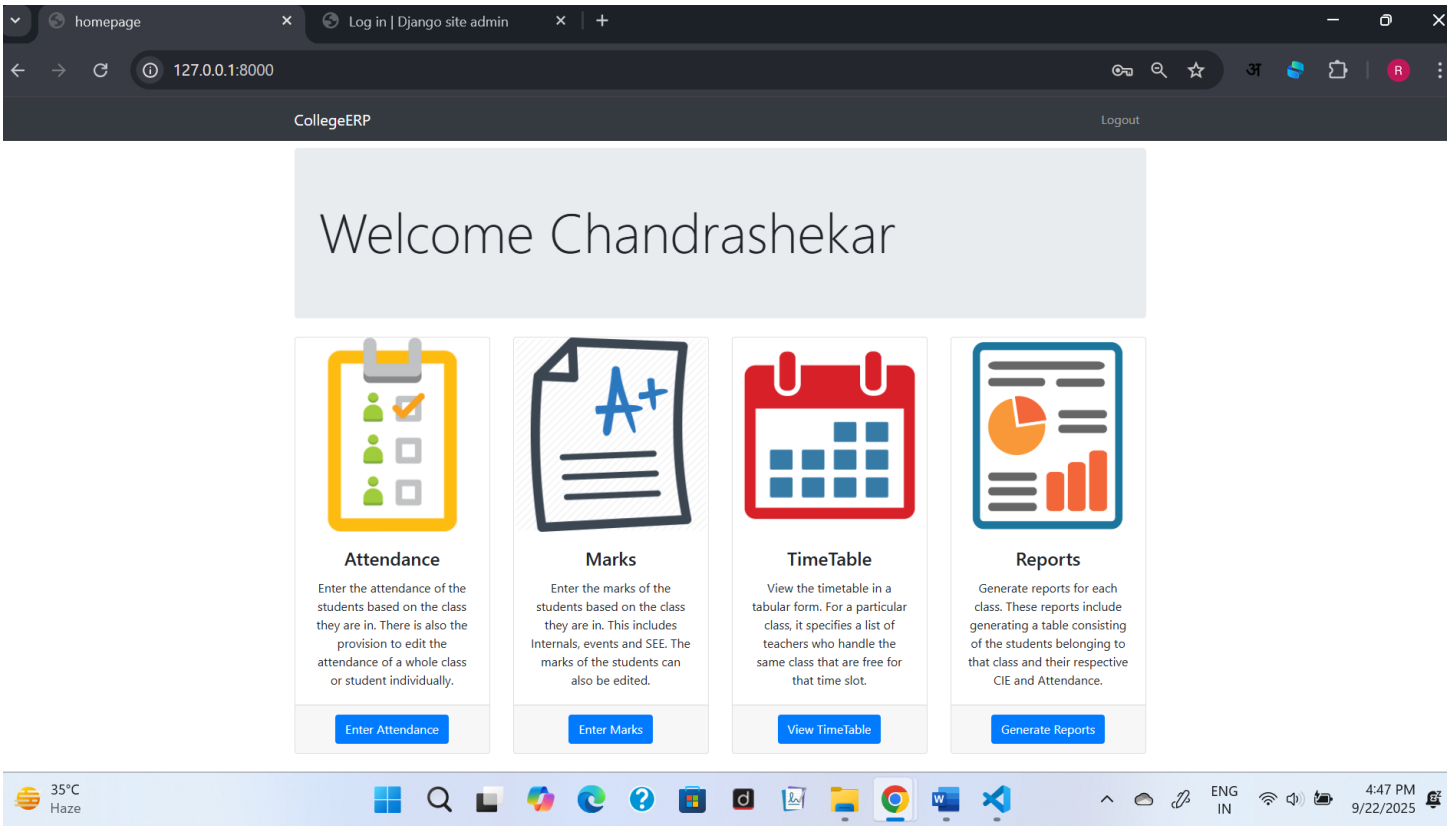
Each schoolteacher in the council is assigned a unique username and word by the director. The username is their schoolteacher ID and the same for word. The schoolteacher may change the word latterly.



Teacher Login

Homepage

After successful login, the pupil is presented a homepage with their main sections, attendance, marks, schedule and reports. In the attendance section, the schoolteacher can enter the attendance of their separate scholars for the days on which classes were conducted. There's a provision to enter redundant classes and view/ edit the attendance of each individual pupil. In the marks section, the schoolteacher may enter the marks for 3 internals, 2 events and 1 SEE for each pupil. They can also edit each of the entered marks. The schedule provides the classes assigned to the schoolteacher with the day and timings in a irregular form. Incipiently, the schoolteacher can induce reports for each of their assigned class.



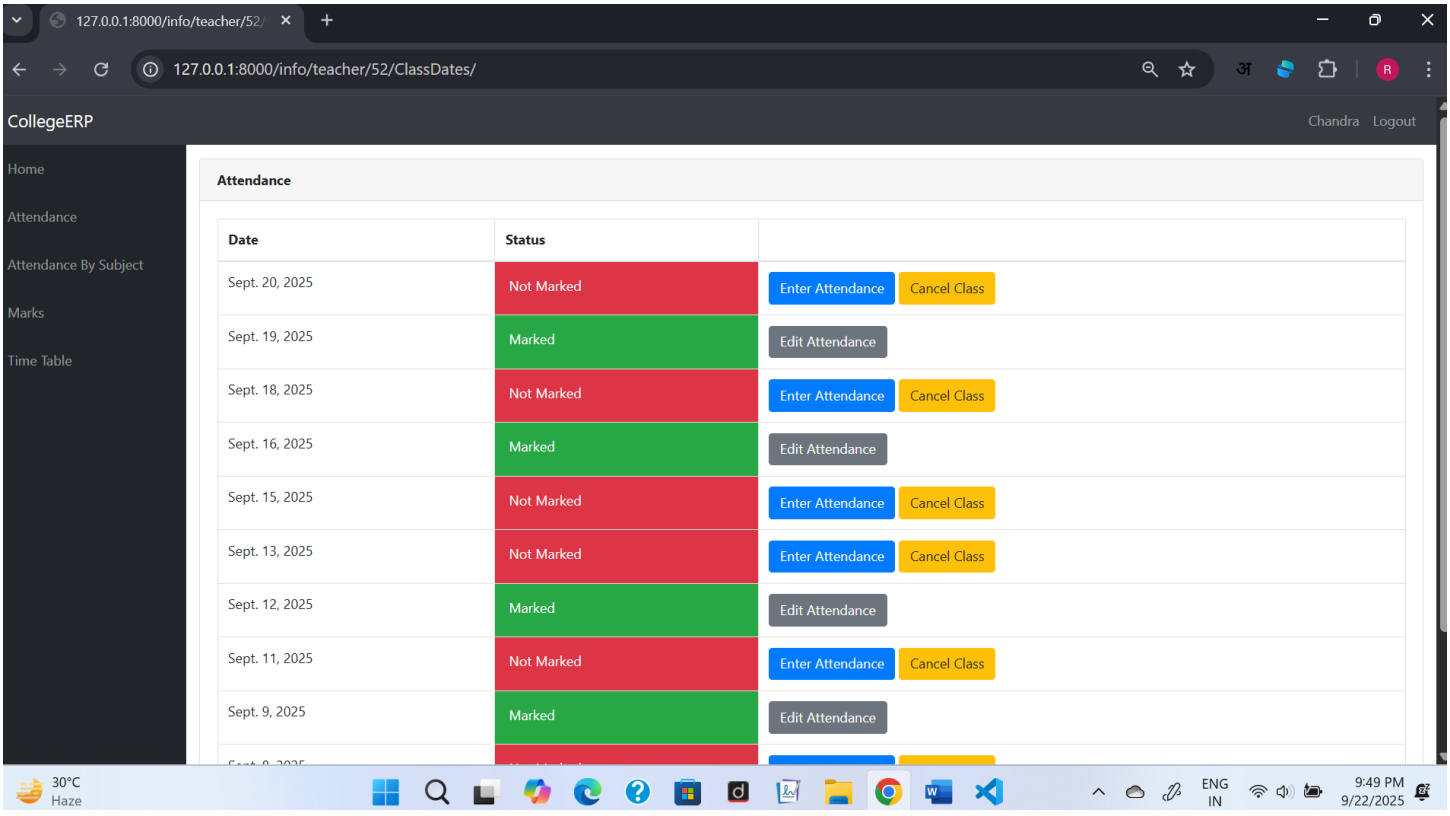
Teacher homepage

Attendence

There's a list of all the class assigned to schoolteacher. So, for each class there are 3 conducts available.

Enter Attendence

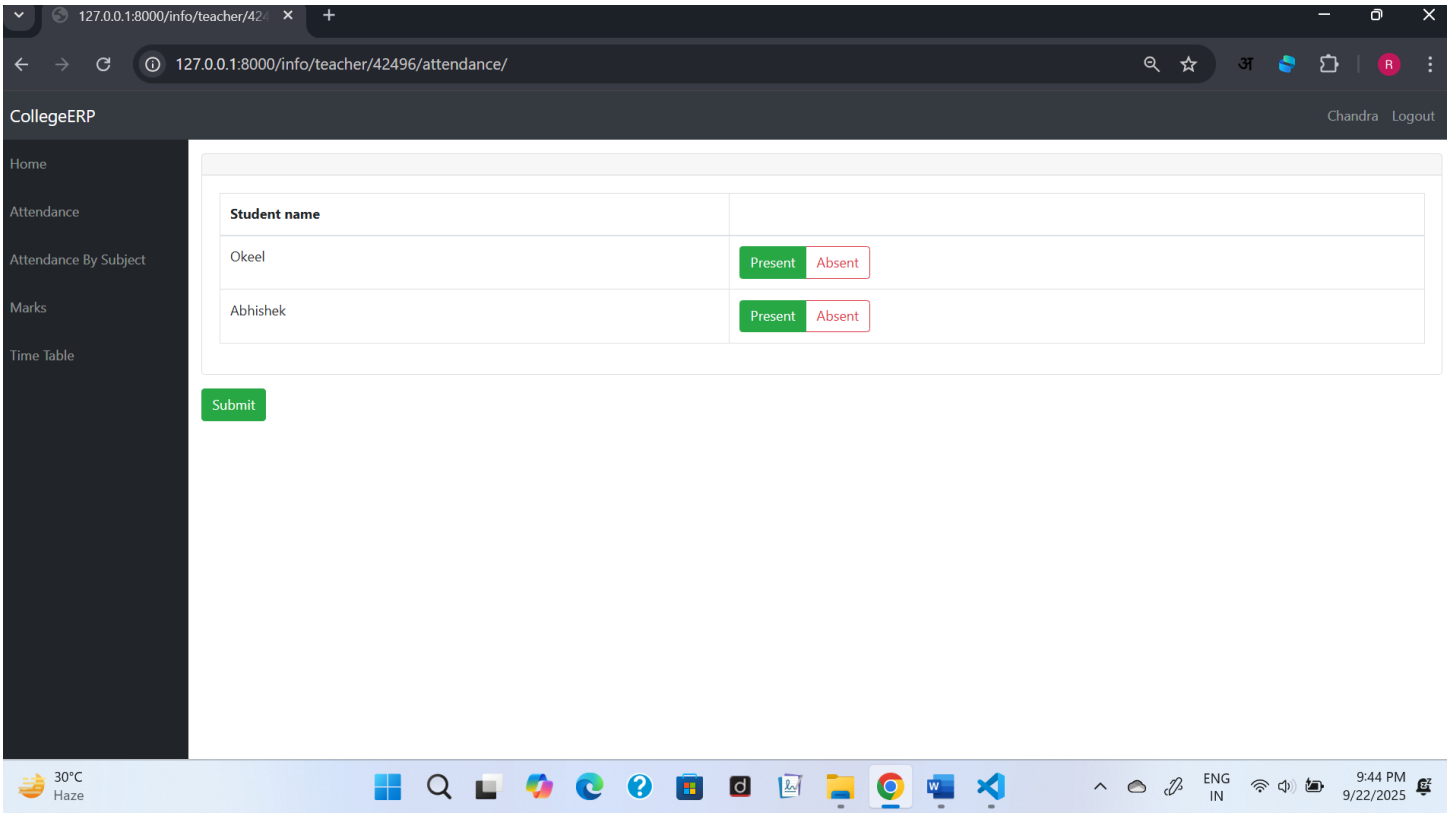
On this runner, the classes listed or conducted are listed in the form of a list. Originally, all the listed classes will be listed from the launch of the semester to the current date. Therefore, if there's class listed for moment, it'll automatically appear on top of thelist. However, else green if pronounced, if the attendance of any day isn't marked it'll be red. Classes can also be cancelled which will make that date as unheroic. While entering the attendance, the list of scholars 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 the buttons are originally marked as present and the schoolteacher just needs to change for the absent scholars.



Entering attendance

Edit Attendence

After entering attendance, the schoolteacher can also edit it. It's analogous to screen for entering attendance, only the entered attendance is saved and display. The schoolteacher can change the applicable attendance and save it.



Editing attendance

Extra Class

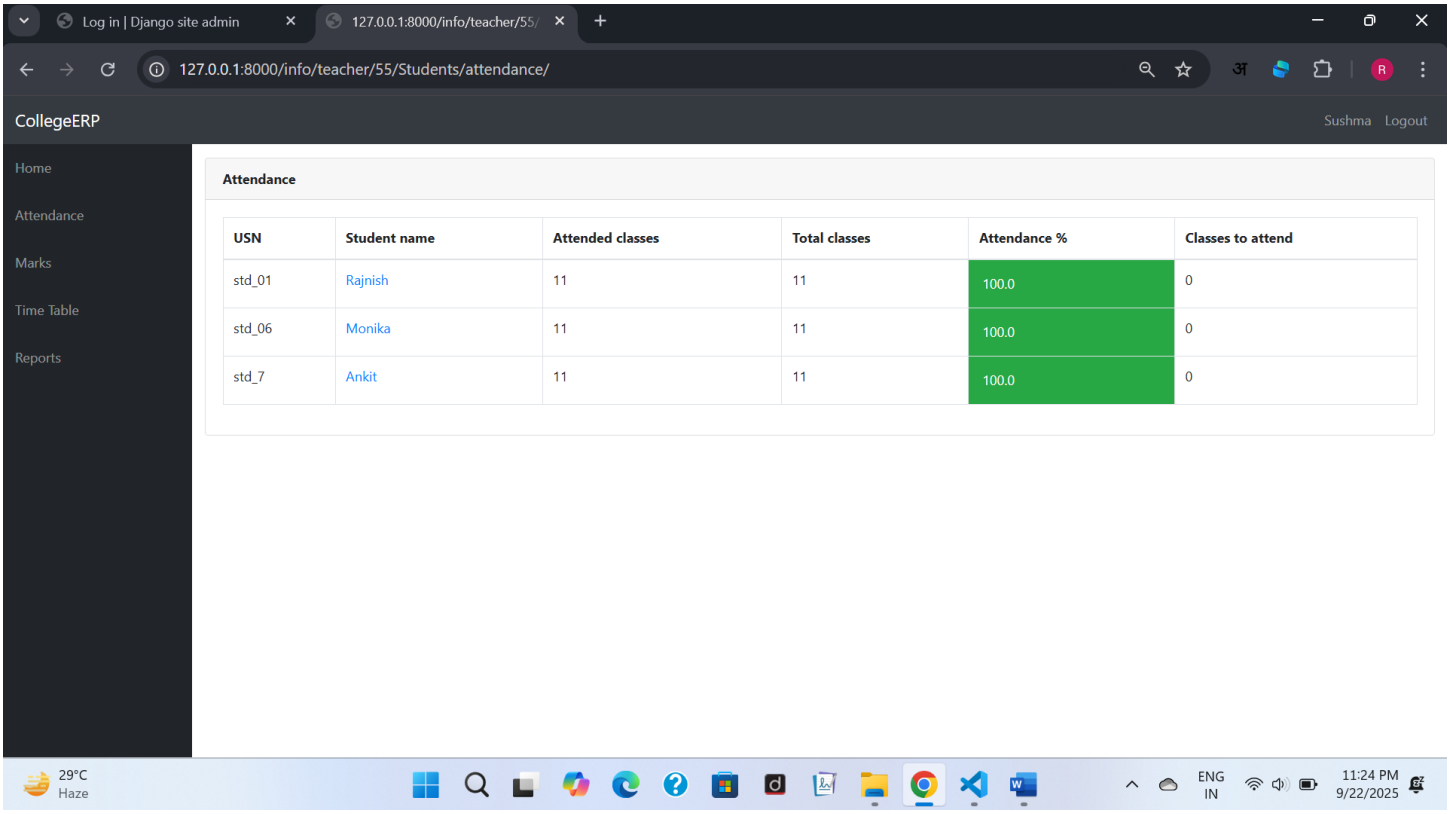
Still, they may enter the attendance for that as well, if a schoolteacher has taken a class other than at the listed timings. While entering the redundant class, the schoolteacher just needs to specify the date it was conducted and enter the attendance of each of the scholars. After submitting redundant class, it'll appear in the list of conducted classes and therefore, it can be edited.

Student Attnedence

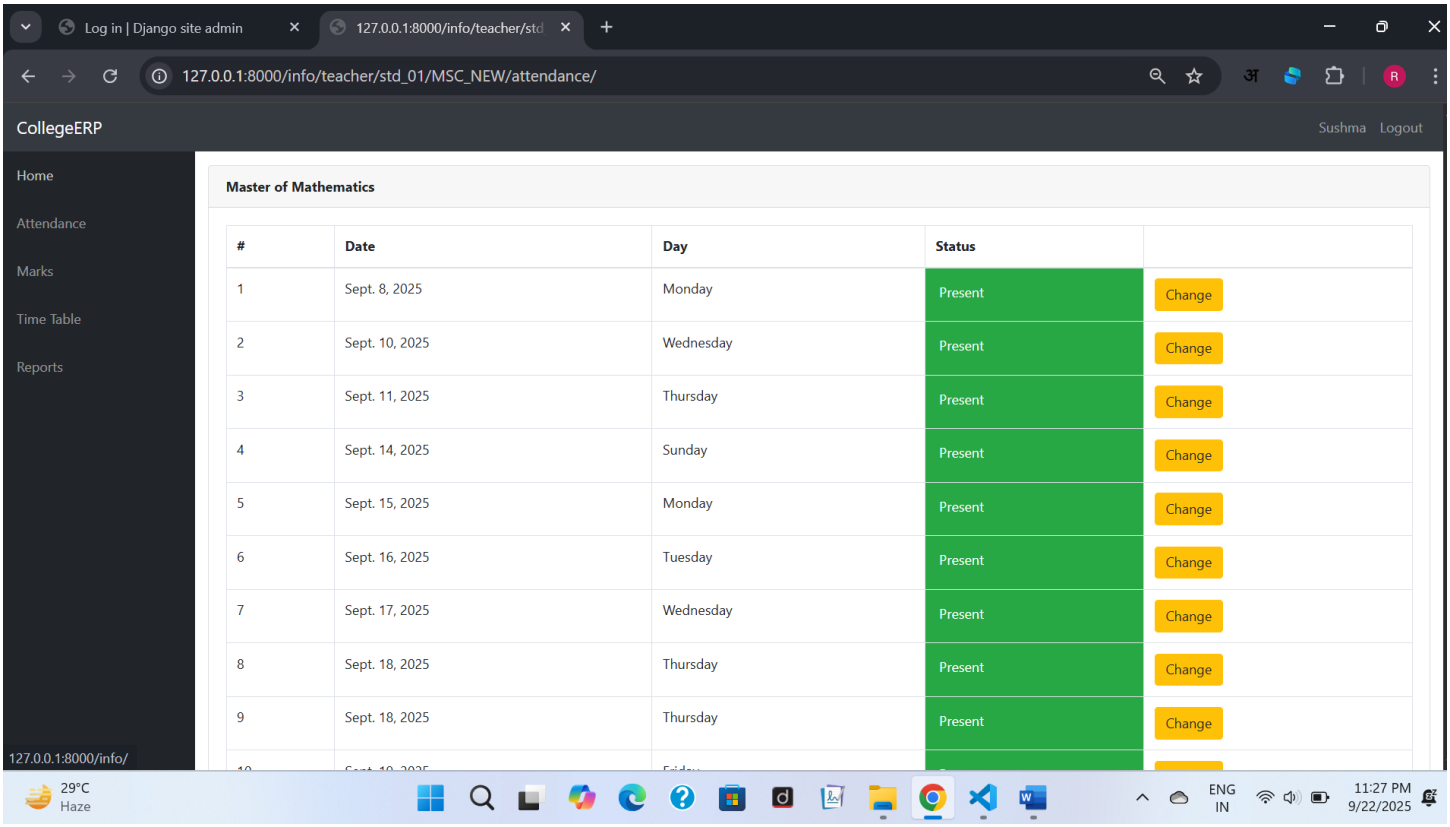
For each assigned class, the schoolteacher can view the attendance status of the list of scholars. The number of attended classes, total number of classes conducted and the attendance chance isdisplayed. However, it'll be displayed in red, if the attendance chance of any of the scholars is below 75. Therefore, the schoolteacher may fluently find the list of scholars not eligible to take a test.

Student Attendance Details

The schoolteacher can view the attendance detail of all their assigned scholars collectively. That is, for all the conducted classes, it'll display whether that pupil was present or absent. The schoolteacher can also edit the attendance of each pupil collectively by changing the attendance status for each conducted class.



Attendance of students in a class



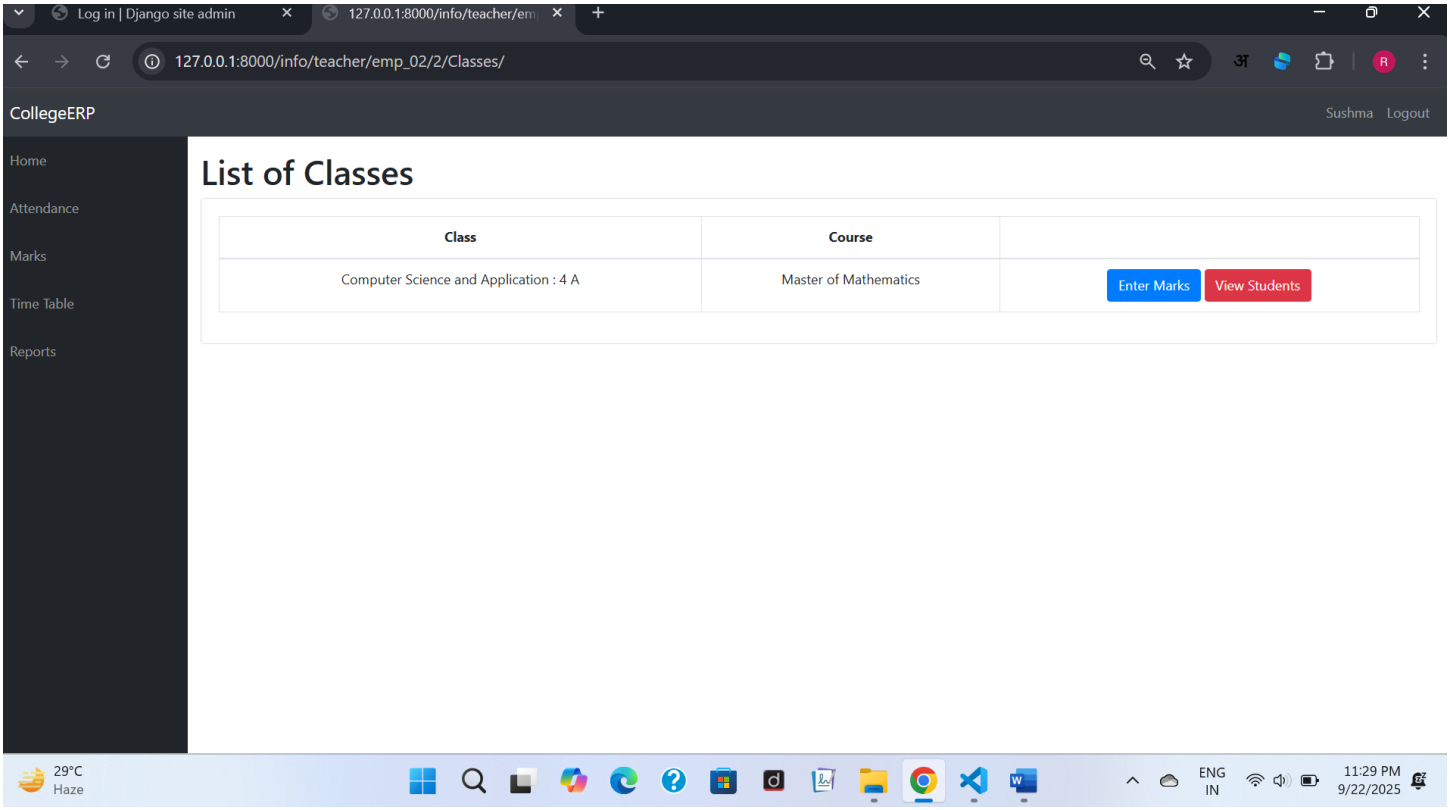
Attendance details of an individual student

Marks

On this runner, the list of classes assigned to the schoolteacher is displayed along with two conducts for each class.

Enter Marks

On this runner, the schoolteacher can enter the marks for 3 internal assessments, 2 events and one semester end test. Originally all of them are pronounced red to denote that the marks haven't been entered yet. Once the marks for a test are entered, it turns green. While entering the marks for a particular test, the list of scholars in that class is listed and marks can be entered for all of them and submitted. formerly, the marks are submitted, the scholars can view their separate marks. Incase if there's a need to change the marks of any pupil, it's possible to edit the marks.



Entering marks

Edit Marks

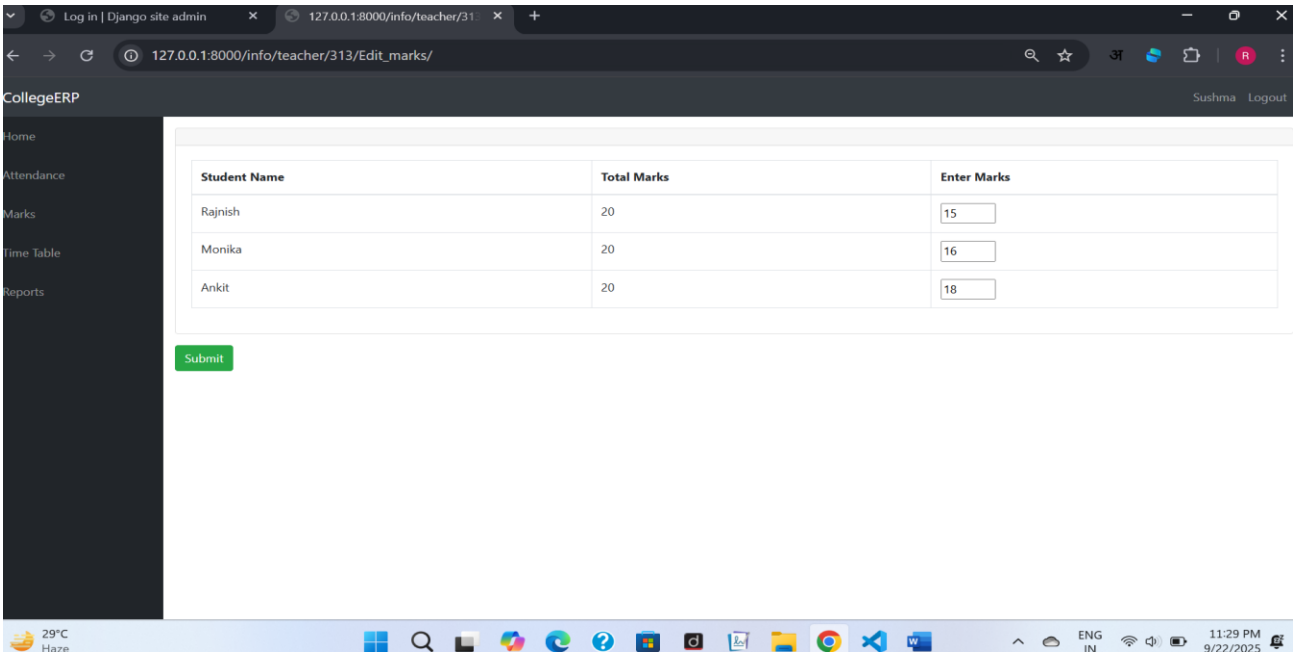
Marks for a test can be edited. While editing, the list of scholars in that class is displayed along with formerly entered marks. The marks to be streamlined can be changed and submitted. The scholars can view this change incontinently.

Students Marks

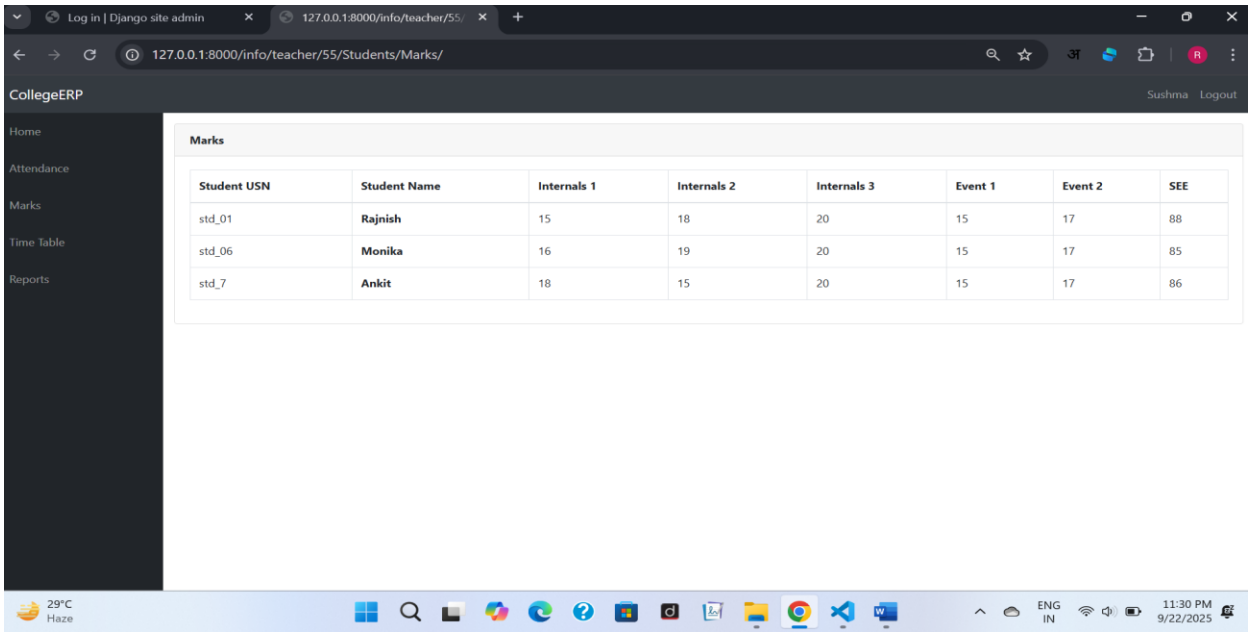
For each assigned class, the schoolteacher has access to the list of scholars and the marks they attained in all the tests. This is displayed in an irregular form.

Timetable

This runner is a table which lists the day and timings of each of the classes assigned to the schoolteacher. The row heads are the days of the week and the column heads are the time places. So, for each day, it specifies the classes in the time places. The schedule is generated automatically from the assign table, which a table containing the information of all the preceptors is assigned to a class with a course and the timings the classes.



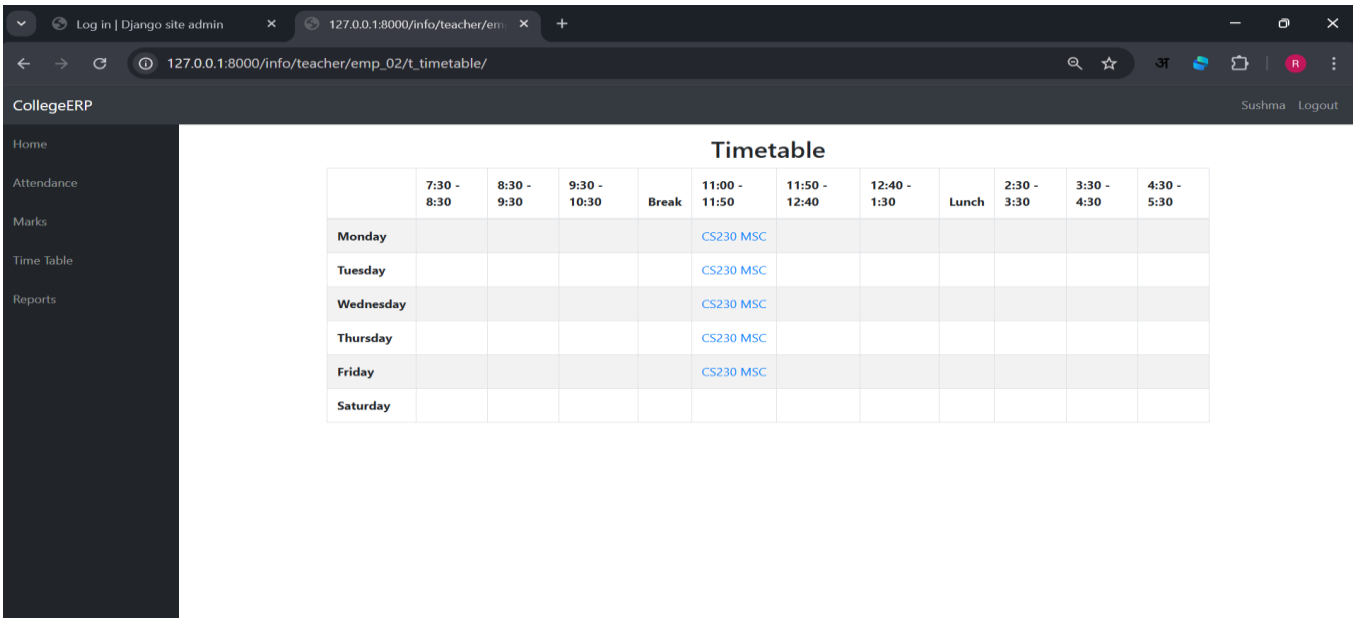
Editing marks



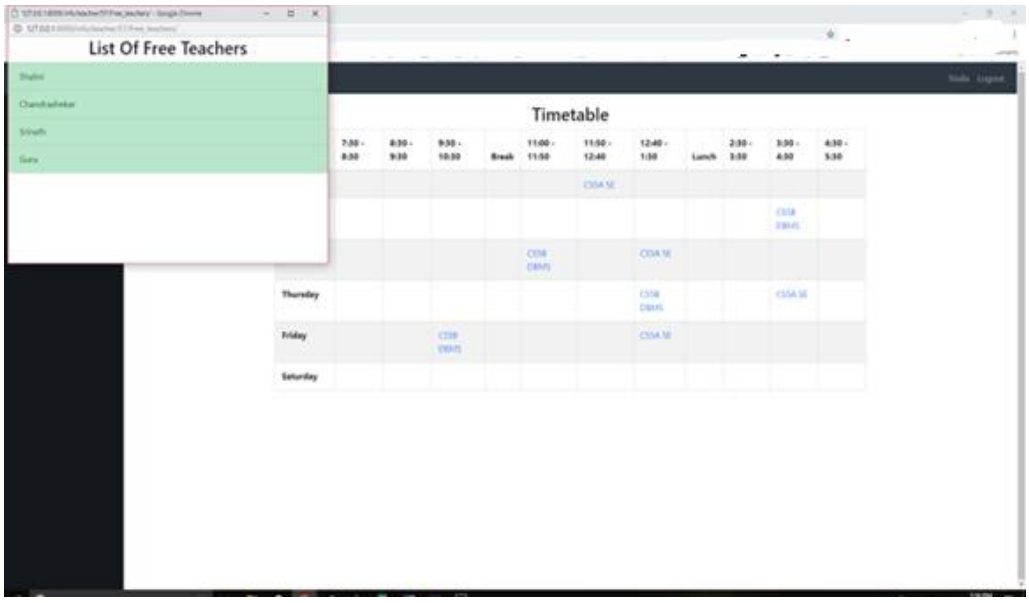
Marks of all the students in a class

Free Teachers

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



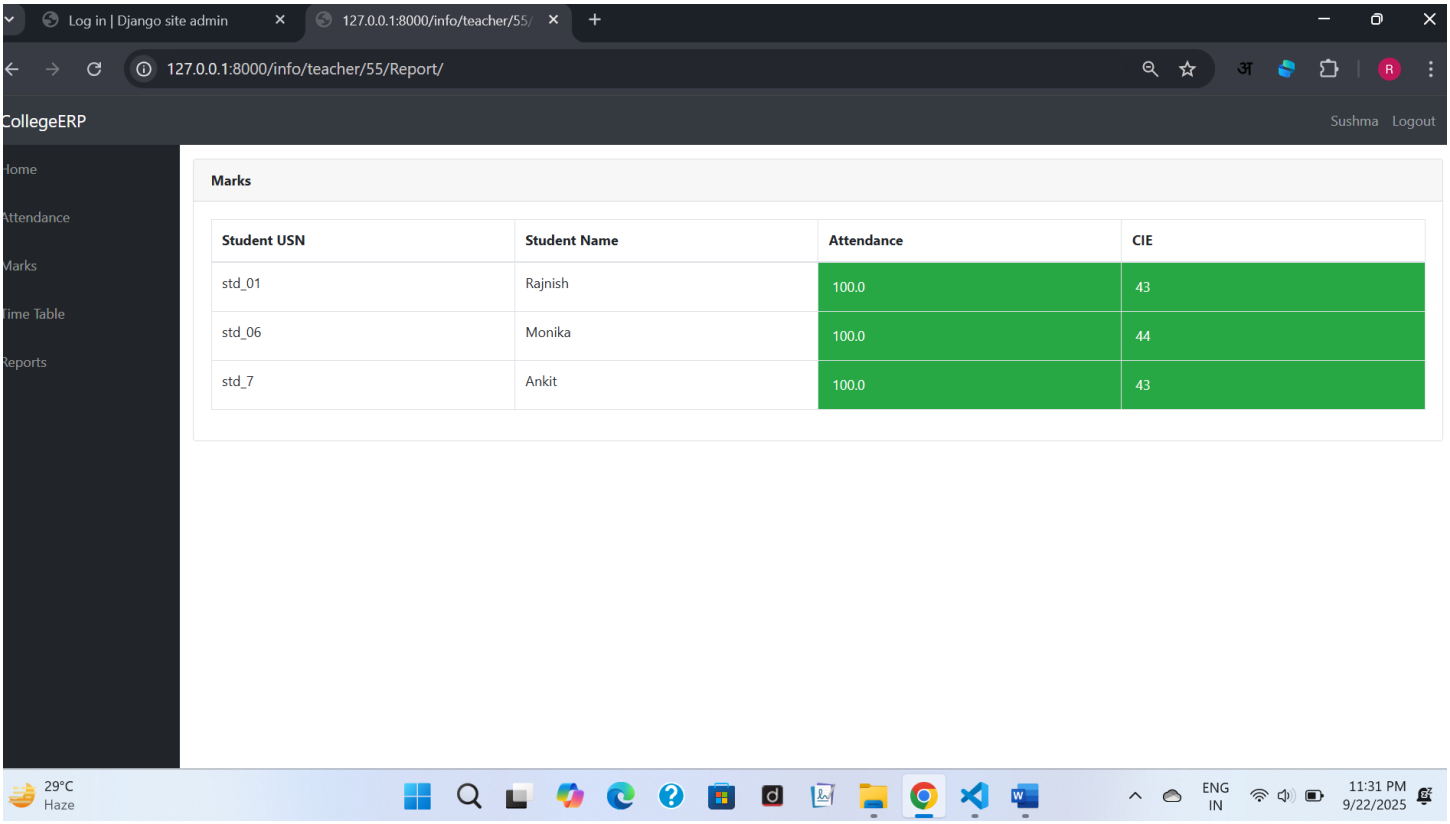
Teacher Timetable



List of free teachers for a time slot

B. Reports

The last runner for the preceptors is used to induce reports for each class. The report specifies the list of scholars in that class and their separate CIE and attendance chance. CIE is the normal of the marks attained from the tests, 3 internals and 2 events. The CIE is out of 50 and the scholars with CIE below 25 are marked in red and aren't eligible to write the semester end test. Also, the attendance chance is displayed with scholars below 75 marked in red.



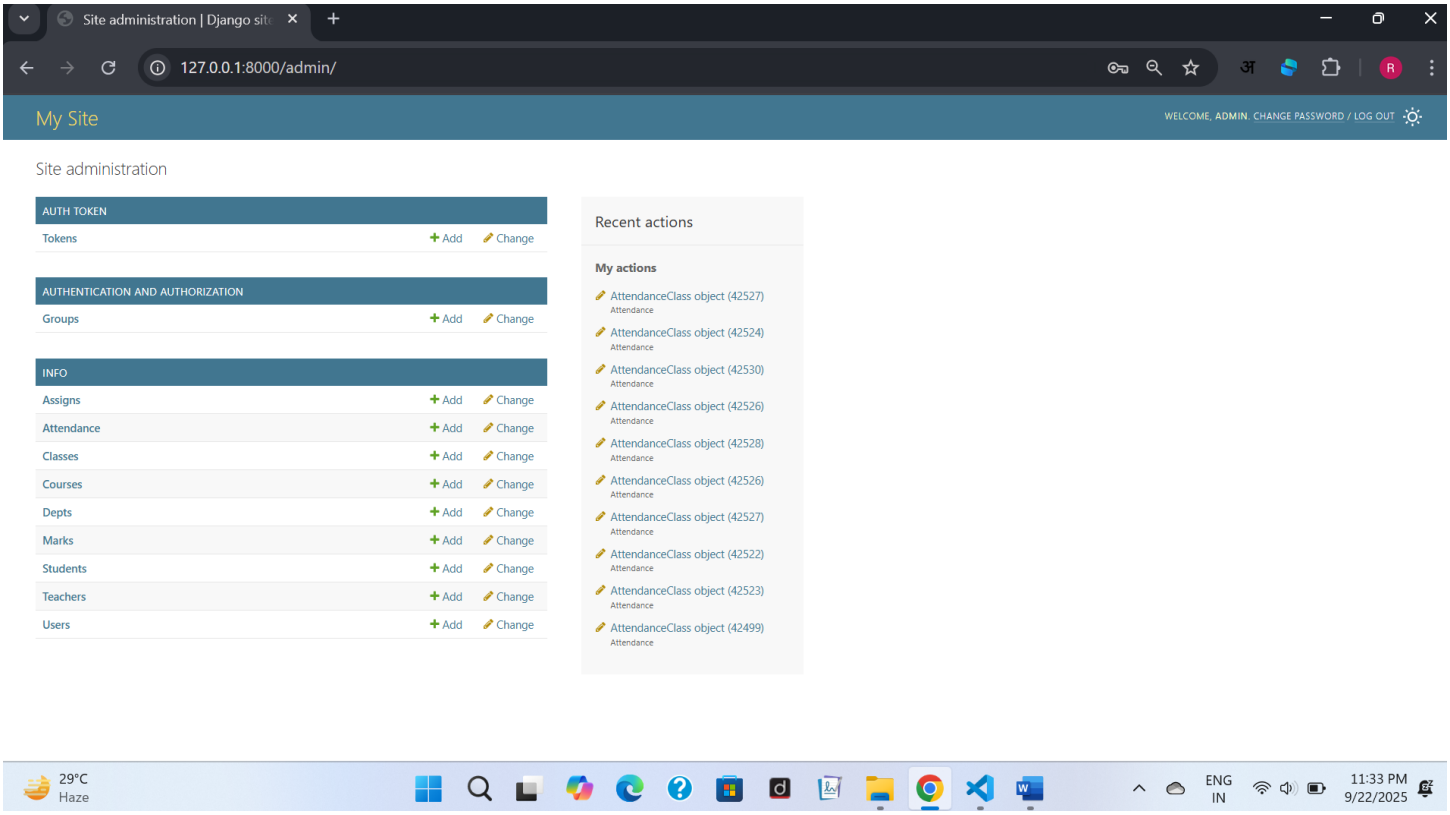
CIE and attendance for a class of students

Administrator

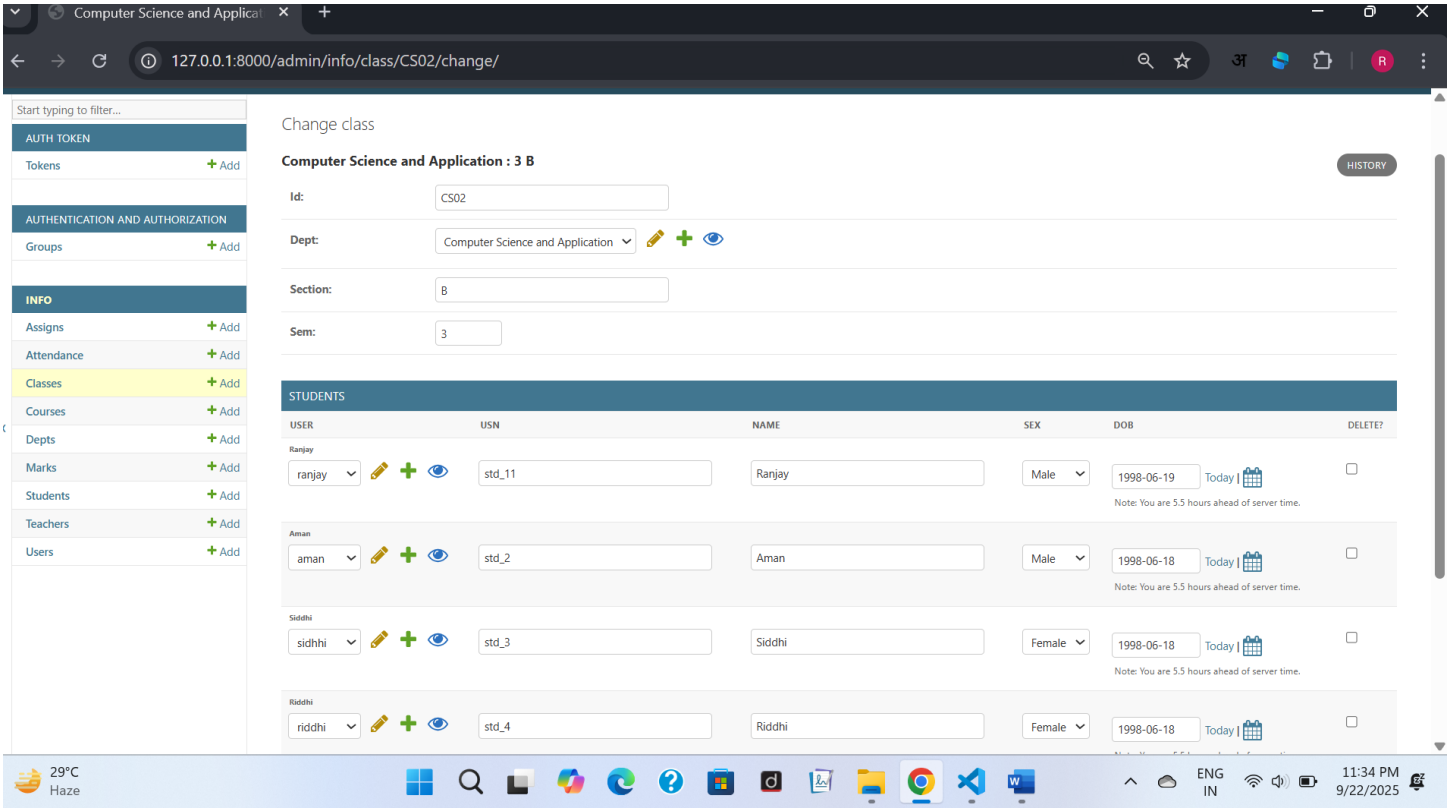
The director is responsible for adding and maintaining all the departments, scholars, preceptors, classes and courses. All this data is stored in the database in their separate tables. The admin is also responsible for adding and maintaining the list of preceptors assigned to class with a course and the timings. This information is stored in the Assign table. The admin also has access to the marks and attendance of each pupil and can modify them.

There are several features in place to insure that querying the database is quick and effective for the director. As the database has the implicit to come huge, there's a hunt point for every table including pupil, schoolteacher etc. The hunt has got a specific record grounded on name or id. Also, it can filter the record grounded on department, classetc.

In figure shows the homepage for the admin, it lists all the different tables in the database. Figure shows the details of the class table. Each class consists of a list of scholars as shown.



Admin homepage



Admin students table page

10. FUTURE SCOPE AND CONCLUSION

A. Future of The Work

The future of work is rapidly evolving as technology advances, automation becomes more prevalent, and the global economy becomes more interconnected. These changes will have a significant impact on the way we work and the skills needed to succeed in the workplace. One industry that will be particularly affected is education, and more specifically, college ERP systems.

College ERP systems are software solutions that help colleges and universities manage their academic and administrative activities. These systems are used to streamline processes such as student admissions, course registration, attendance tracking, and grading. With the advent of new technologies, college ERP systems are becoming more advanced and sophisticated, offering new features and capabilities.

One of the biggest trends in the future of work is the rise of remote work. With the COVID-19 pandemic, many companies have shifted to remote work, and this trend is likely to continue. College ERP systems will need to adapt to this new reality by offering more online tools and features that can be accessed from anywhere. This will require colleges and universities to invest in new technologies and infrastructure to support remote learning and administration.

Another trend that will impact college ERP systems is the rise of automation. As more tasks become automated, the need for human intervention will decrease, and the focus will shift to higher-level tasks that require critical thinking and problem-solving skills. College ERP systems will need to incorporate more AI and machine learning technologies to automate routine tasks, freeing up staff to focus on more complex work.

Finally, the future of work will be characterized by a need for continuous learning and upskilling. As technologies evolve and new skills become necessary, employees will need to continually learn and adapt to remain competitive. College ERP systems can play a crucial role in this process by offering online learning platforms and personalized learning pathways that can be tailored to individual needs.

B. Conclusion

In conclusion, the future of work will bring significant changes to the way we work, and college ERP systems will need to adapt to these changes to remain relevant. By embracing new technologies and offering innovative solutions, college ERP systems can play a vital role in preparing students for the jobs of the future.

By using Being System penetrating information from lines is a delicate task and there's no quick and easy way to keep the records of scholars and staff. Lack of robotization is also there in the Being System. The end of Our System is to reduce the workload and to save significant staff time.

Tittle of the design as College ERP System is the system that deals with the issues related to a particular institution. It's the veritably useful to the pupil as well as the faculties to easy access to chancing the details. The council ERP provides applicable information to druggies grounded on their biographies and part in the system. This design is designed keeping in view the day-to-day problems faced by a council system.

The abecedarian problem in maintaining and managing the work by the director is hence over- come. Prior to this it was a bit delicate for maintaining the time table and also keeping track of the diurnal schedule. But by developing this web- grounded operation the director can enjoy the task, doing it eases and also by saving the precious time. The quantum of time consumption is reduced and also the homemade computations are neglected, the reports can be attained regularly and also whenever on demand by the stoner. The effective application of the work, by proper sharing it and by furnishing the accurate results. The storehouse installation will ease the job of the driver. Therefore, the system developed will be helpful to the director by easing his/ her task.

This System give the automate admissions no primer processing is needed. This is a paperless work. It can be covered and controlled ever. It reduces the man power needed. It provides accurate information always. All times together gathered information can be saved and can be penetrated at any time. The data which is stored in the depository helps in taking intelligent opinions by the operation furnishing the accurate results. The storehouse installation will ease the job of the driver. Therefore, the system developed will be helpful to the director by easing his/ her task furnishing the accurate results. The storehouse installation will ease the job of the driver.

This design is successfully enforced with all the features and modules of the council operation system as per conditions.

11. BIBLIOGRAPHY

1. Elmasri and Navathe: Fundamentals of Database Systems, 7th Edition, Pearson Education, 2016.
   (The authoritative source for database concepts, which underpins the entire Database Design, ER Diagrams, and Data Integrity section of the report).

2. Ian Sommerville: Software Engineering, 10th edition, Person Education Ltd, 2015.
   (Provides comprehensive coverage of requirements engineering, system models, and design, supporting sections on Elicitation, Analysis, and Design).

3. Roger S Pressman: Software Engineering- A Practitioners approach,8th edition, McGraw-Hill Publication, 2015.
   (This is a cornerstone text for software engineering processes, providing the foundation for chapters on Project Planning, SRS, Testing, and the overall SDLC methodology).

4. https://en.wikipedia.org/wiki/Requirements-engineering

5. https://web.cs.dal.ca/ hawkey/3130/srs-template-ieee.doc

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

7. https://en.wikipedia.org/wiki/Class diagram

8. https://www.djangoproject.com/

9. https://getbootstrap.com/

10. https://www.tutorialspoint.com/

11. https://creately.com/

12. GLOSSARY

A. Academic Terms

Term	Definition
Attendance Percentage	The ratio of classes attended by a student to the total classes conducted, expressed as a percentage. A threshold (e.g., 75%) is often used to determine eligibility for exams.
CIE (Continuous Internal Evaluation)	The process of assessing a student's performance throughout the semester through internal tests, quizzes, projects, and assignments. In this system, it is calculated from the best of 3 internal tests and 2 event marks.
Course	A subject or module offered by a department (e.g., "Data Structures," "Calculus"). Each course has a unique ID, name, and is assigned credits.
Credit	A unit that represents the weightage or value of a course towards a degree program.
Department (Dept)	An academic division within a college focused on a specific field of study (e.g., Computer Science, Civil Engineering).
SEE (Semester End Examination)	The final examination for a course, typically held at the end of the semester and carrying high marks (e.g., 100).
Semester	An academic term, typically half of a full academic year (e.g., Semester 4).
Section	A subdivision of a class within a semester and department (e.g., Section A, Section B).
Syllabus	The outline of topics, contents, and schedule of a course to be covered within a semester.

B. System & User Roles

Term	Definition
Administrator (Admin)	A superuser with full system access. Responsible for managing all data: adding students/teachers, creating courses/classes, and assigning teachers to classes.
Stakeholder	Any individual or group with an interest in the system's functionality and success (e.g., Students, Teachers, Administrators, Parents).
Student	The primary end-user of the system. Can view their personal information, attendance, marks, and timetable. Has no editing rights.
Teacher / Faculty	A user who manages academic data. Can enter and edit attendance and marks for students in their assigned classes. Can apply for leave.
User	Any individual who interacts with the ERP system, possessing a unique username and password.

C. Technical & Techno-Functional Terms

Term	Definition
API (Application Programming Interface)	A set of rules that allows different software applications to communicate with each other. The system provides APIs for fetching student details, attendance, etc.
Authentication	The process of verifying a user's identity, typically through a username and password.
Authorization	The process of determining what an authenticated user is allowed to do and access within the system (e.g., a teacher can only mark attendance for their class).
Black Box Testing	A software testing method where the internal structure/design is unknown to the tester. Focus is on input and output functionality.
CRUD (Create, Read, Update, Delete)	The four basic functions of persistent storage. Administrators typically have full CRUD capabilities.
Database	An organized collection of structured data stored electronically (e.g., SQLite). It holds all information like student records, attendance, etc.
Django	A high-level Python web framework that encourages rapid development and clean, pragmatic design. It is the core technology used to build this ERP system.
ERP (Enterprise Resource Planning)	Integrated management software used by organizations to manage day-to-day business activities. A *College ERP* automates academic and administrative tasks.
Frontend	The part of the website users interacts with directly (e.g., web pages built with HTML, CSS, and Bootstrap).
Backend	The server-side logic that powers the application, handles database operations, and serves the frontend (e.g., built with Django and Python).

Term	Definition
Module	A self-contained unit of software that handles a specific function (e.g., Student Module, Attendance Module, Marks Module).
ORM (Object-Relational Mapper)	A programming technique that converts data between incompatible type systems (OOP and SQL databases). Django's ORM lets developers use Python code instead of SQL to query the database, enhancing security and productivity.
RBAC (Role-Based Access Control)	A security method that restricts system access to users based on their roles within an organization (Student, Teacher, Admin).
SRS (Software Requirement Specification)	A detailed description of the software system to be developed, including its purpose, features, constraints, and interfaces.
SQL Injection	A code injection technique that attackers use to exploit vulnerabilities by inserting malicious SQL statements into an input field. Prevented by using Django's ORM.
UI (User Interface) / GUI (Graphical User Interface)	The point of human-computer interaction in a device. The system uses a web-based GUI designed with Bootstrap for a clean and intuitive experience.
White Box Testing	A software testing method where the internal structure/design is known to the tester. Focus is on testing internal code paths and logic.

D. Project Management Terms

Term	Definition
Elicitation	The practice of gathering requirements from stakeholders through interviews, surveys, and meetings.
Gantt Chart	A type of bar chart that illustrates a project schedule, showing tasks, durations, and dependencies.
SDLC (Software Development Life Cycle)	A process used by the software industry to design, develop, and test high-quality software. The Waterfall model was adopted for this project.
Stakeholder Identification	The process of identifying all individuals or groups impacted by the project and documenting their interests and influence.
Waterfall Model	A sequential (non-iterative) software development process where progress flows steadily downwards (like a waterfall) through phases (Conception, Initiation, Analysis, Design, Construction, Testing, Deployment).

TABLE OF CONTENTS

Introduction

The objective of College Information Management System is to allow the administrator of any organization the ability to edit and find out the personal details of a student and allows the student to keep up to date his profile. It'll also facilitate keeping all the records of students, such as their id, name, mailing address, phone number, DOB etc. So, all the information about a student will be available in a few seconds. Overall, it'll make Student Information an easier job for the administrator and the student of any organization.

The main purpose of this project is to illustrate the requirements of the project College Information Management System and is intended to help any organization to maintain and manage personal data. It is a comprehensive project developed from the ground up to fulfill the needs of colleges as they guide their students. This integrated information management system connects daily operations in the college environment ranging from Attendance management to communicational means among students and teachers. This reduces data error and ensures that information is always up-to-date throughout the college. It provides a single source of data repository for streamlining our processes and for all reporting purposes. It has a simple user interface and is intuitive. This ensures that the users spend less time in learning the system and hence, increase their productivity. Efficient security features provide data privacy and hence, increase their productivity.

As we know that, a college consists of different departments, such as course departments, fees management, library, event management etc. Nowadays applications and uses of information technologies is increased as compared to before, each of these individual departments has its own computer system to do their own functionalities. By having one main system they can interact with each other from their respected system by having valid user id and password.

Time schedule for completion of the project work

The Project schedule activities will consist of following:

1. Selecting The Project Title
2. System Requirement Collection
3. System Design
4. Acquiring the required resources
5. Coding
6. Testing of the Application
7. Deployment

Objective

The primary aim of a College ERP System is to create a user-friendly web platform accessible to students, faculty, and staff. This system should streamline administrative tasks and enhance overall efficiency by centralizing college data and automating key processes. For students, the objective is to provide easy access to academic information, simplify enrollment and fee payment, and improve communication with the college. For faculty, the system should facilitate efficient management of courses, student records, and communication. Ultimately, the ERP system aims to empower administrators with comprehensive tools for managing all aspects of college operations, including student information, academics, finances, and resources, leading to better decision-making and a more effective educational environment.

Project Plan and Problem Definition

Inception:

Inception is a process of establishing a basic understanding of the problem and the nature of the solution. This includes the need for this software, identification of stakeholders and defining multiple viewpoints.

What is the purpose of this project?

There is currently an ERP system in our college. But not everyone is happy with the system. While it is a step towards automating the college activities, it comes with its own set of problems. This project is designed to implement a college ERP system to eradicate some of these problems and add some features of our own that would add value to system.

Why do we need ERP?

Nowadays, in schools and colleges, it is very difficult to manage each and everything manually. Supervising and maintaining the whole database of a school or college can be time-consuming and challenging especially if it's done on a regular basis. So, we need to handle and manage everything smartly.

To solve this problem ERP (Enterprise Resource Planning) is used. ERP software makes it easy to track the progress of every department of school and automate different functions. With ERP every- thing can be seen on a single dashboard. The administrator can manage the college from anywhere. The possibilities of maintaining the whole database of a college with ERP software are endless.

Some of the prominent roles of ERP are:

- Manages the office and automates different functions.
- Helps in long-term management and planning of all departments of college.
- Eliminates the need for having multiple management software for each department.
- Daily activities like attendance can be digitalized and automated.
- Leave module for teachers can be automated.

Identification of stakeholders

Enterprise Resource planning implementation is a difficult and complex decision where it involves people issues more than technological issues. Identification of stakeholders is a key step during the process of ERP implementation, because if done improperly, it will lead to failure of the implementation project. The stakeholders are listed below:

Teachers

Teachers are the key stakeholders of the college ERP. Because they are the one who manage, edit, update the contents of the database of students such as attendance, internal marks, CGPA etc.

It also helps them to assign their class to other teachers when they are on leave. This makes it easier to identify who among them are free to take the class at that time. So. this software helps them reduce their overhead and make their tasks easier and simple.

Students

Students are end users of ERP software. The attendance, internals marks uploaded by the teachers is viewed by students. It helps them track their attendance status. It also helps them to communicate with teachers and their classmates. So, students make up another set of stakeholders of this software.

Administrator

College administrator is responsible for maintaining the database of the college. They will have the privilege to modify the database i.e., to add/remove students/teachers/staff, update information regarding each of these.

Viewpoints

For a teacher, this software must be easy to use. It should be easy to find different modules like attendance, leave module, internals marks, result etc... Teachers are the one who updates the contents of the database, so it should be update save modify it.

Student's viewpoint

A student can only view the information about himself, other than that everything will be hidden from them. They will not have the option to edit anything. So, the graphical user interface must be good. They expect it to be functional.

Administrator's viewpoint

Administrator will have the privilege to view all the information about the college. They will have the option to track goals like, Average marks of all the students in a subject, Average attendance of all the students of a class etc.

What do you expect from the module the lets you enter the marks of the students?

There will be another section to enter the CIE of all the students. The internals will be for 20 marks and when the faculty enters it into the ERP, it must automatically convert it into 10 marks. Generally, there will be 5 events. There will be 3 internals, followed by two events such as quiz, project. If the student scores below 50% of the allocated marks in the subject, then there must be a warning message sent to the student to score more marks in the upcoming internals.

At the end of all the events if the student could not mark the 50 marks, then there will be a make-up test conducted by the faculty so that the student would be having another chance to come up to the mark of 50%. These make-up test marks must be altered with the minimum marks of the CIE scored. And the final CIE marks should be displayed and be stated that the student is eligible or not eligible to take up the Semester End Examination. If the student is not able to take up the CIE due to personal reasons or if he is representing the college in any form of the activity, then it must be brought into the notice of the lecturer and the leave can be availed. If the student is ill, then the medical certificate must be attested, and a letter must be sent to the HOD to take up re-test. After the faculty enters the CIE there must be an option to save the CIE marks. When the CIE marks are saved then the students will not be able to see the marks in their marks. They can view their CIE only when the marks are locked by the faculty. If the faculty locks the CIE, then there would not be any chance to change the CIE. The CIE must be locked after confirming the marks with the students only.

As a student, what are some problems you are facing with the current ERP system?

The ERP status was not updated regularly, and they could not track their attendance status as the app would crash. The GUI that is used in the interface is not up to the mark. It is difficult to keep the track of the attendance and the CIE. It would be easy if the attendance would be shown in a calendar like format so that it would be convenient and can also keep a track of the status of the attendance. There should also be forums where the teacher and the students are active. This will help the students in many ways such as studies, assignments, projects and so on. There should be interaction with the student-student and student-teacher so that the students can clear their doubts with any teacher as well as any student at any point of time. The forum will also help the students in conveying the information to all the students at a faster rate.

For the students who were in supplementary batch, they could not attend the first few weeks of class as they had exams. But, in the ERP they were marked as absent which made their attendance drastically low.

When the students are into college activities such as LCC sessions, IEEE sessions, representing our college in sports or any other activities then students are marked absent. There must be another way to handle these problems so that there will be justice for the students for their hard work.

Software Requirement Specification

Purpose

The purpose of this project is to develop a College Management System that helps the teachers and students in easier management of college activities such as attendance, marks.

Overall Description

Product Perspective:

This project is modeled based on the current ERP system in the college. Students and teachers face several problems while using the system. Therefore, we wanted to build a system that has lesser number of features than the current system but, has more functionality.

Product Features:

- Each teacher will be able to enter attendance and marks for their respective students.
- Each student will be able to view the attendance status for their respective courses.
- The teachers will be able to apply for various types of leave directly through the system.
- The students will be able to Communicate and provide feedback to their teachers.
- The students will have access to a forum page where they are communicating will each other.

User Classes and Characteristics:

There are several types of end users for the college ERP system. They are broadly divided as Students, Staff and the Administrator. Each of these classes has their own set of features.

The student should have the following features:

- View the Attendance status of the courses to which they are enrolled.
- View the Marks of the courses to which they are enrolled.
- View the notification from the college administrator.
- Communicate or give feedback to their respective teachers.
- Communicate with other students of the same university.

The staff should have the following features:

- Access to the information of all students that attend their courses.
- Add and edit the Attendance status of those students.
- Add and edit the exam marks of those students.
- Avail the different types of leave.

The administrator should have the following features:

- Add and update students, teachers and courses.
- Assign teachers and students to courses.

System features

Expected requirement: Student and Staff information

Description and priority Information regarding students, teachers and courses are stored in the database. Every user can view only certain information based on their user class. For example, a teacher can view student and course information that they are handling. This feature is of high priority as the information must be viewed by only the authorized users.

Functional requirements

- Each user shall be able to view information in the database based on their user class.
- The administrator shall be able to view all the information in the database.

Normal requirement: Attendance and marks entry

Description and priority Attendance and marks entry is the main feature of the College ERP system. Hence, the priority is high. Teachers update the attendance and marks of the students who are part of her class. Students can view their respective Attendance and marks of the courses they have taken.

Functional requirements

- Teachers shall be able to view, update and edit the attendance and marks of the students, part of their class.
- Teacher shall be able to take extra classes, switch classes with other teachers.

Exciting requirement: Communication among students and teachers

Description and priority Students and teacher will be able to communicate with each other directly using the ERP system. Students may give their queries and feedback to a teacher and they may respond accordingly.

Functional requirements

- Students shall be able to communicate with their teachers by sends personal messages.
- Students shall be able to communicate with other students through a forum section.

External Interface Requirements

User Interfaces

The User interface is made using Bootstrap. Firstly, there will be a simple login page separate for students and teachers. Each student and teacher will have a unique interface. There will be a fixed sidebar with links to all the modules.

Hardware Interfaces

Since neither the mobile application nor the web portal have any designated hardware, it does not have any direct hardware interfaces. Any browser can be used to access the web-app.

Software Interfaces

The following is a list of software used in making of the project:

Operating System: We have chosen Windows operating system for its best support and user- friendliness.

Django: We have chosen to use Django for the back-end of the website as Django is a simple python framework and is suitable for beginners.

Database: We are using SQLite database, which comes as default with Django.
Communications Interfaces.

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.

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 in- formation to the users.

Portability: The users access the ERP from various platforms such as desktops and mobile phones. The web-app must be portable to all platforms and the user experience must be optimal.

Gantt chart

Test	3 Days	5 Days	20 Days	42 Days	4 Days	3 Days
Feasibility						
Requirement Analysis						
Design						
Coding						
Testing						
Implementation						

Task	Start Date	Duration
Feasibility	7/1/2025	3
Requirement Analysis	7/5/2025	5
Design	7/11/2025	20
Coding	8/1/2025	42
Testing	8/19/2025	4
Implementation	9/25/2025	5

Technical details

Operating Environment

The operating environment for College ERP system is listed below:

- Operating System: Windows 10 /11
- Database: MySQL database
- Front end: HTML/CSS/Bootstrap
- Back end: Python Django

We will be using HTML, CSS and Bootstrap as a frontend and Django as a backend to enhance the College ERP System.

1. **HTML** - HTML (Hypertext Markup Language) is the standard markup language for creating web pages. It defines the structure and content of a web page, including text, images, links, and multimedia elements.

2. **CSS** - CSS (Cascading Style Sheets) is used for styling and formatting web pages. It allows you to define the look and feel of a web page, including colors, fonts, and layouts.

3. **Bootstrap** - Bootstrap is a free and open-source CSS framework directed at responsive, mobile-first front-end web development. It contains HTML, CSS and JavaScript-based design templates for typography, forms, buttons, navigation, and other interface components.

4. **Django** - Django is a free and open-source, Python-based web framework that follows the model–template–views architectural pattern. It is maintained by the Django Software Foundation, an independent organization established in the US as a 501 non-profit.

Data Flow Diagram

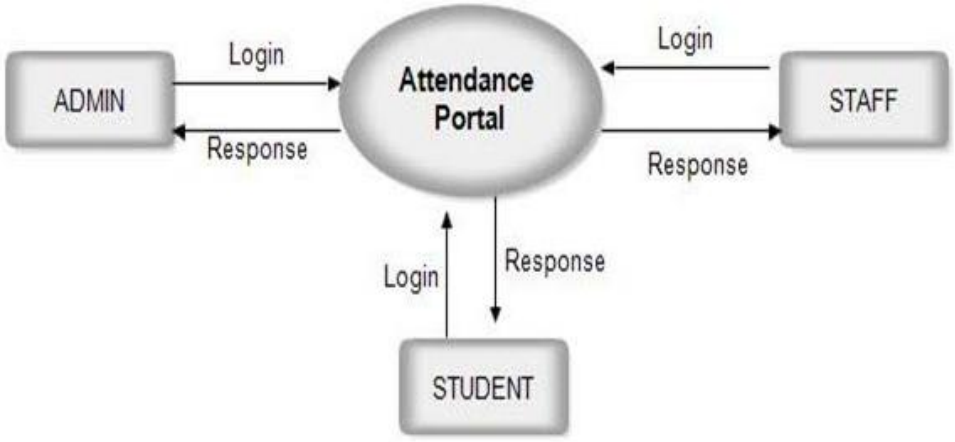


Fig: Level – 0

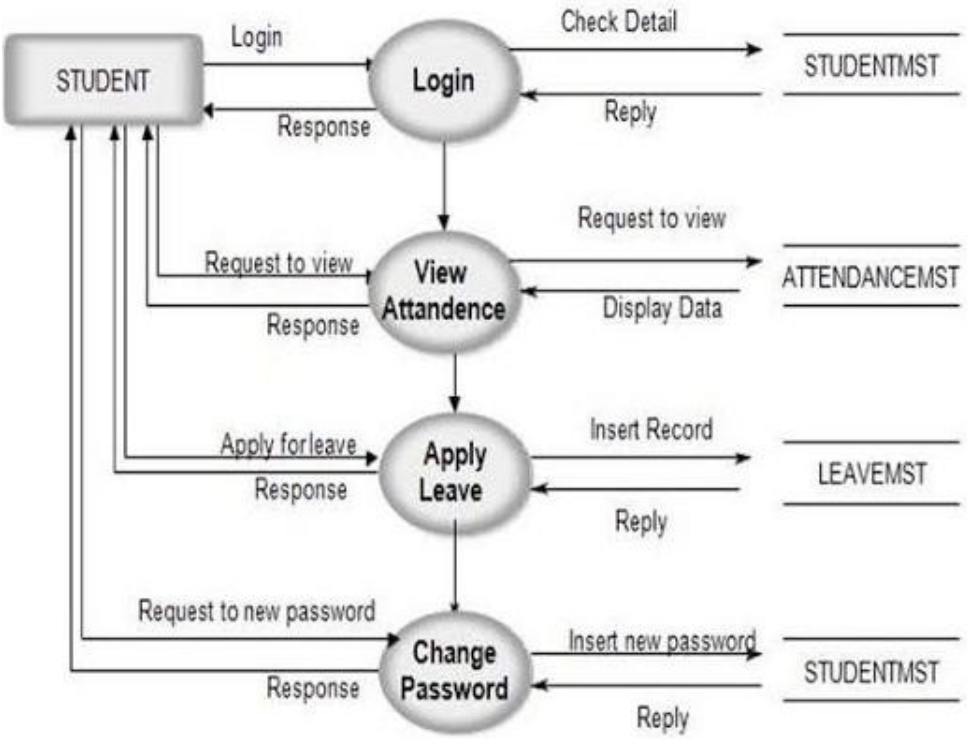


Fig: Level -1 Student

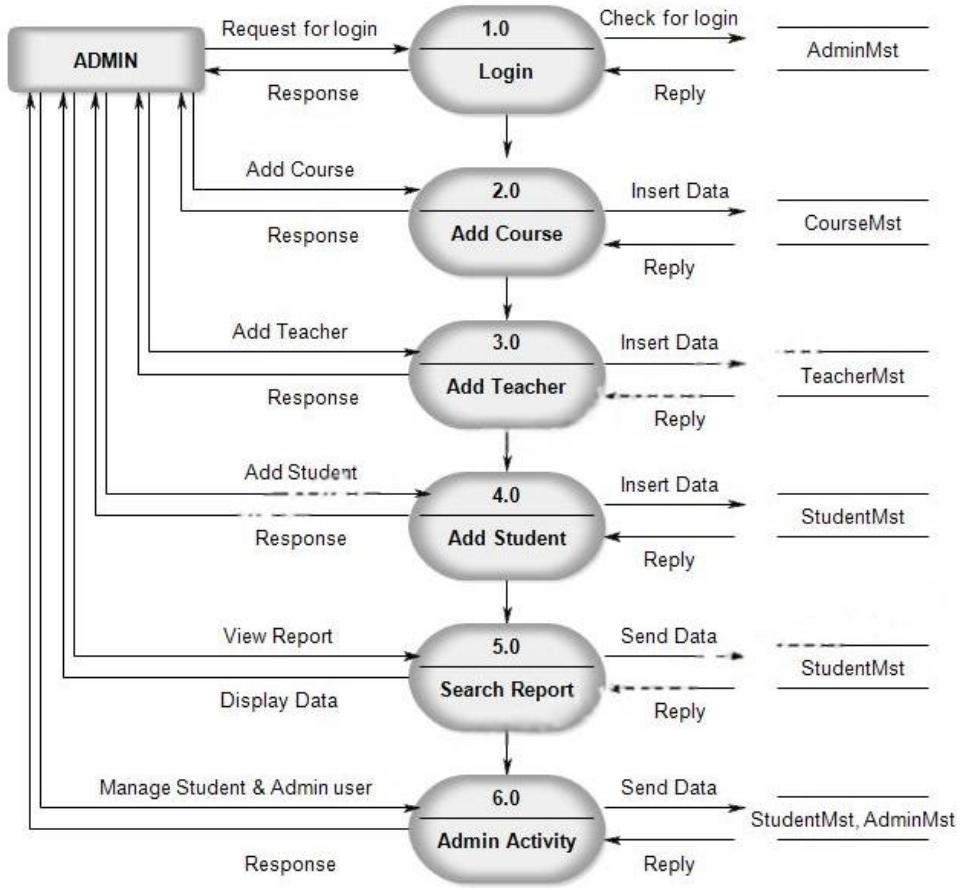


Fig: Level -1 Admin / Teacher

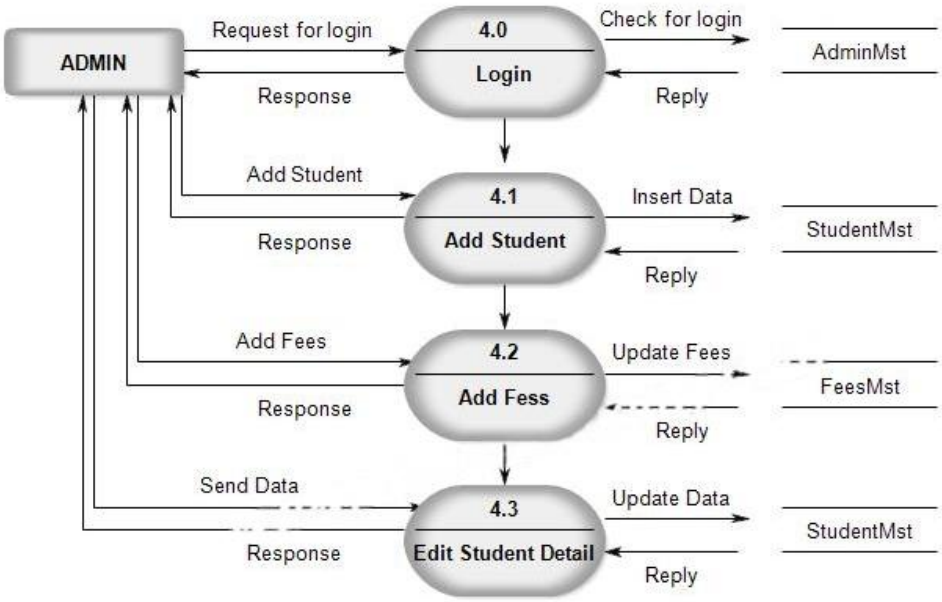


Fig: Level -2 Admin

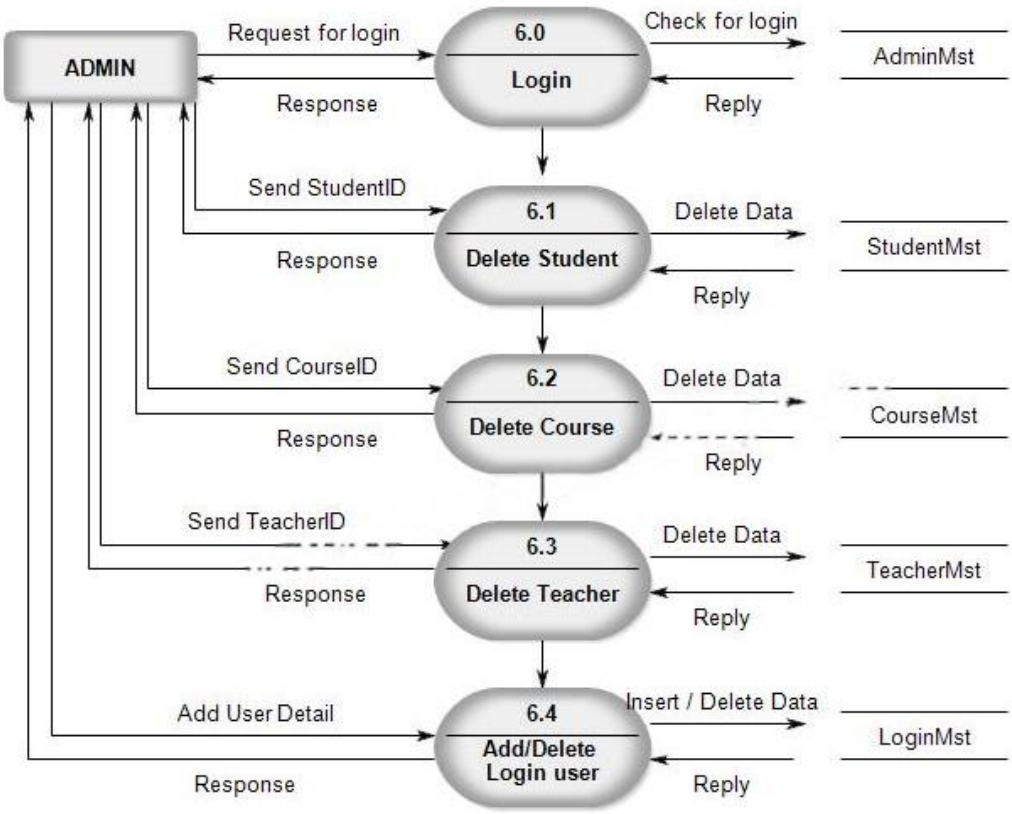


Fig: Level -2 Admin / Teacher

Activity chart

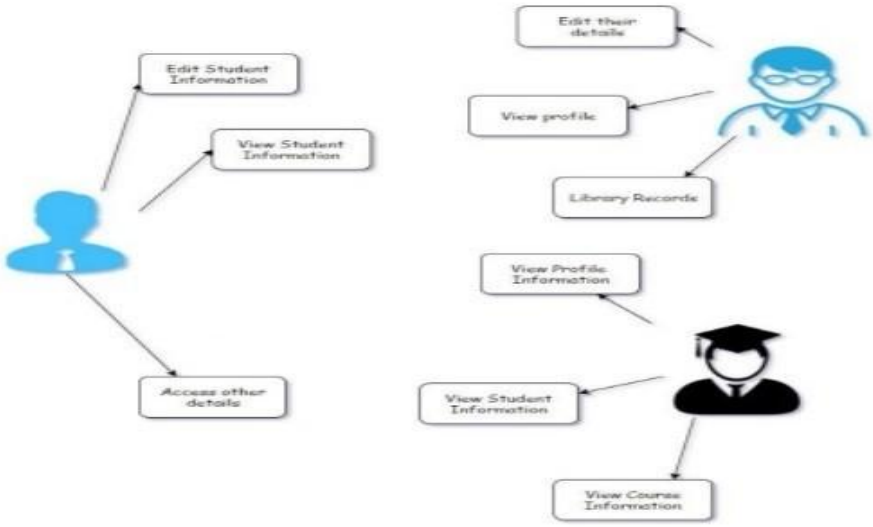


Fig: Activity diagram of college ERP

Class Diagram

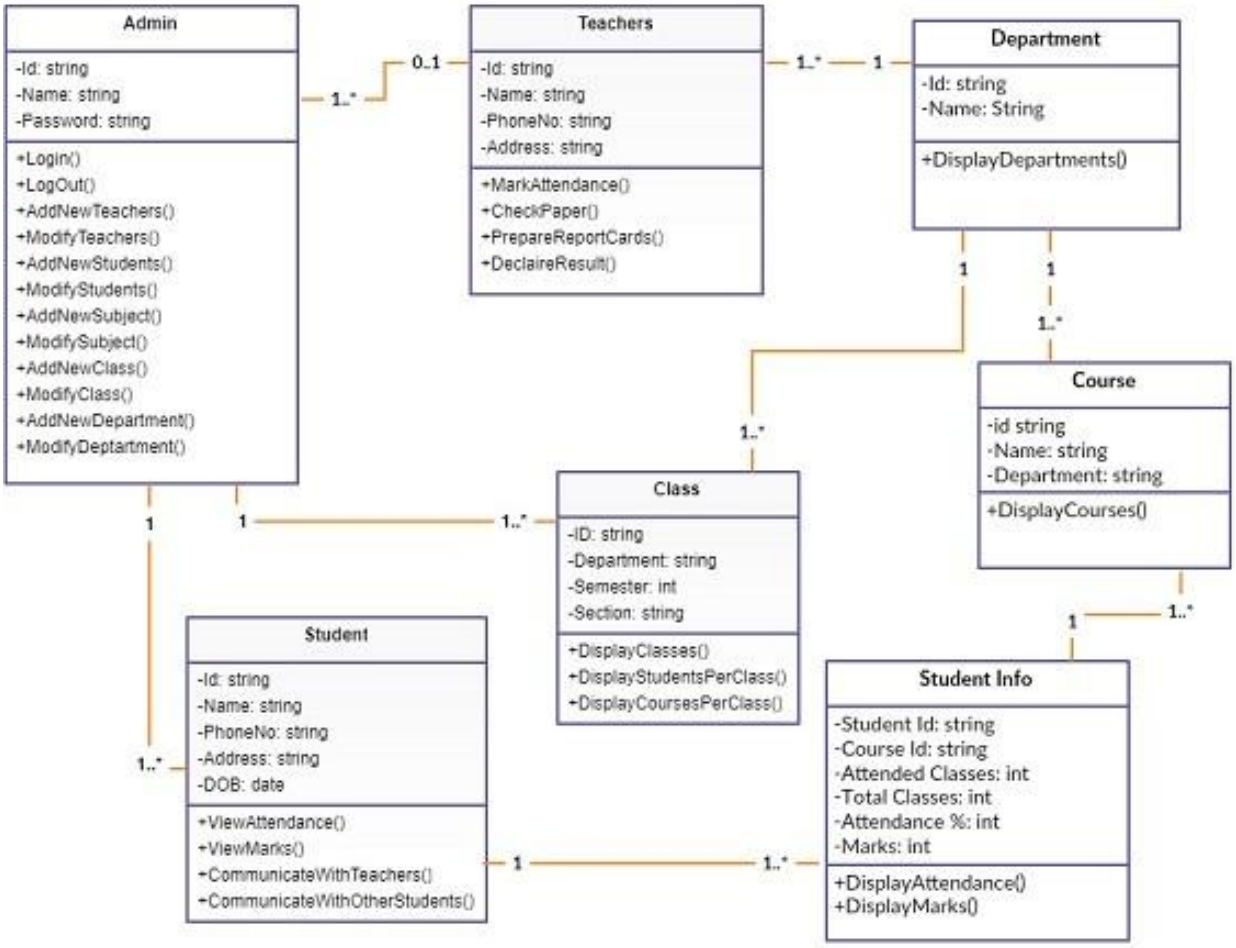


Fig: Class diagram of college ERP

ER - Diagram

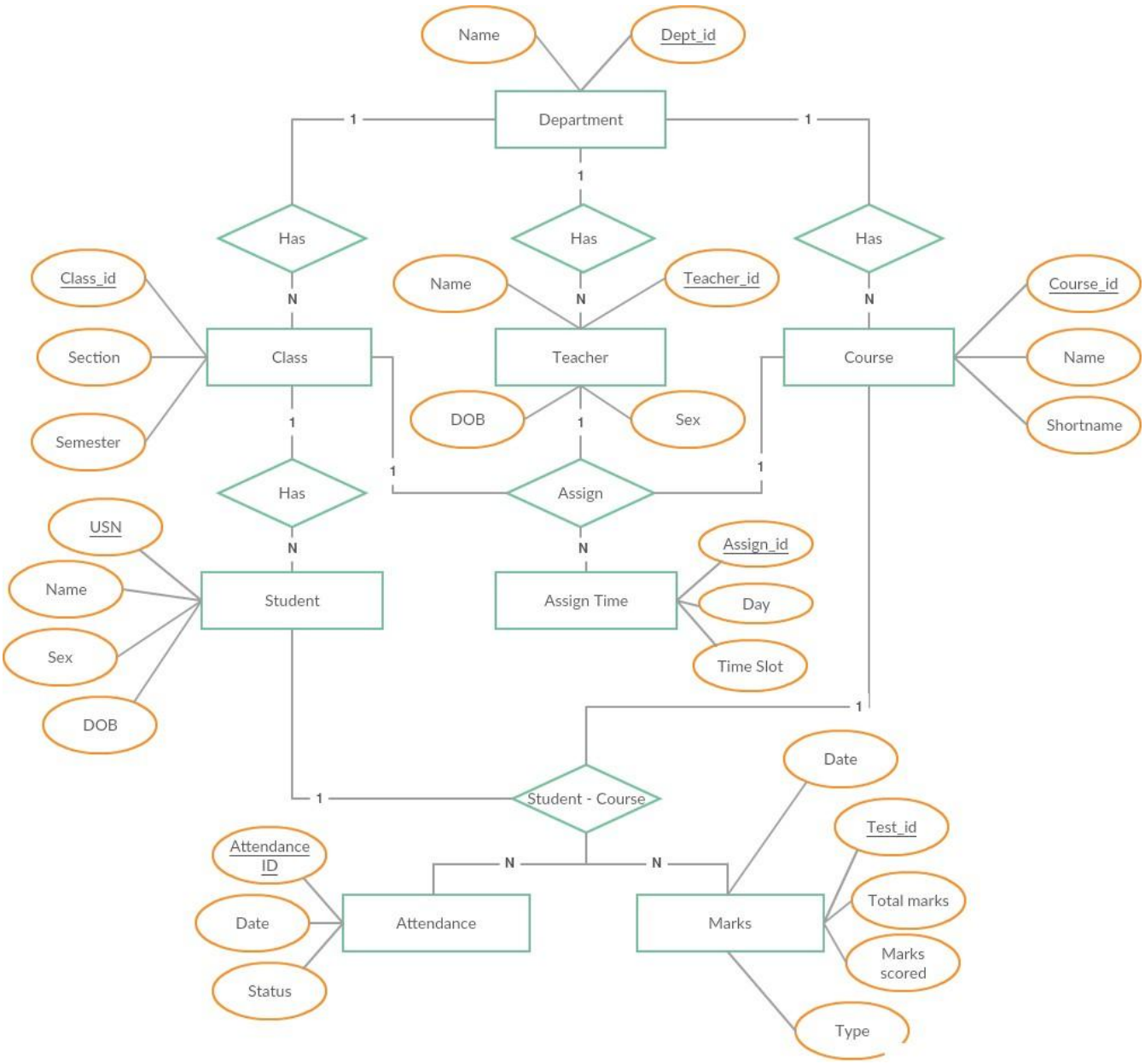


Fig: Entity Relationship diagram of college ERP

Data Dictionary

Add User

S. No.	Field Name	Data Type	Constraints	Description
1	Name	String	Not null	Name of User (F&L)
2	Email	String	Not null	Email of User
3	Password	String	Not null	User Password
4	Id	String	Primary key	Unique Id
5	Role	String	Not null	Teacher Or Student

Department

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Name	String	Not null	Name

Course

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Unique Id
2	Dept.	String	Foreign key	Name
3	Name	String	Not null	Course Name
4	Shortname	String	Not null	Short name

Classes

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Dept.	String	Foreign key	Name
3	Section	String	Not null	Section
4	Sem.	String	Not null	Semester

Marks

S. No.	Field Name	Data Type	Constraints	Description
1	Student	String	Foreign key	Name & ID
2	Course	String	Foreign key	Name & ID

Assignments

S. No.	Field Name	Data Type	Constraints	Description
1	Id	String	Primary key	Id
2	Course	String	Foreign key	Name & Id
3	Teacher	String	Foreign Key	Name & Id

Testing Tool

Once source code has been generated, software must be tested to uncover as many errors as possible before it can be used or delivered. Our goal is to design a series of test cases that have a high likelihood of finding errors. Tests are conducted from three different perspectives:

1. **Black Box testing:** It is used for validation. Here, we ignore internal working mechanisms and focus on what the output is.
2. **White box testing:** It is used for verification. The focus is on internal mechanisms, i.e. how is the output produced?
3. **Unit Testing:** It focuses on the smallest unit of software design. We test an individual unit or a group of interconnected units. This is typically done by the programmer using sample inputs and observing the corresponding outputs.

References

In preparing this synopsis, I have read some books and visited some websites :

1. Elmasri and Navathe: Fundamentals of Database Systems, 7th Edition, Pearson Education, 2016.
2. Ian Sommerville: Software Engineering, 10th edition, Person Education Ltd, 2015.
3. Roger S Pressman: Software Engineering- A Practitioners approach,8th edition, McGraw-Hill Publication, 2015.
4. https://en.wikipedia.org/wiki/Requirements-engineering
5. https://web.cs.dal.ca/ hawkey/3130/srs-template-ieee.doc
6. http://www.ntu.edu.sg/home/cfcavallaro/Reports/Report%20writing.htmTop
7. https://en.wikipedia.org/wiki/Class diagram
8. https://www.djangoproject.com/
9. https://getbootstrap.com/
10. https://www.tutorialspoint.com/
11. https://creately.com/
12. https://www.overleaf.com/project