ABSTRACT

The Student Management System (SMS) is a comprehensive software application designed to enhance the efficiency and accuracy of managing student information in educational institutions. By automating various administrative tasks, it reduces the manual workload on staff and improves data management. Key features of SMS include online student enrollment and registration, streamlining the process and reducing paperwork; academic records management for efficient handling of grades, attendance, and schedules; and automated fee management for billing and payment tracking.

SMS ensures data security and confidentiality through robust authentication and authorization mechanisms, safeguarding student information. Leveraging modern web technologies, SMS provides a user-friendly interface that is accessible from multiple devices, including smartphones, tablets, and computers, allowing for convenient access and management of information. The system addresses common administrative challenges faced by educational institutions, such as reducing errors in data entry, improving the accuracy of student records, and enhancing overall operational efficiency. By centralizing and digitizing student information, SMS promotes a more organized and efficient educational environment, ultimately contributing to better student outcomes and institutional performance

CONTENTS

١. INTRODUCTION

١.١ Motivation

The motivation to develop a Student Management System (SMS) stems from the need to address several pressing challenges faced by educational institutions in managing student information. Traditional methods of handling student data، often reliant on paper-based processes and disparate systems، are inefficient، error-prone، and time-consuming. These methods create significant administrative burdens، leading to delays and inaccuracies in data management، which ultimately affect the overall educational experience for students، teachers، and administrative staff.

١.٢. Objective of the project

The objectives of the Student Management System (SMS) project are to improve the efficiency and accuracy of managing student information within educational institutions. The system aims to automate key administrative tasks such as student enrollment، registration، academic record management، and fee processing، reducing manual workload and errors. It seeks to enhance data accuracy and accessibility، ensuring that student information is up-to-date and easily accessible to students، teachers، and staff. The project includes creating a communication module for sending notifications، announcements، and messages to improve interaction among all parties involved. Ensuring data security is a priority، with robust measures to protect sensitive information.

١.٣. Purpose of the project

The purpose of the Student Management System (SMS) project is to develop an efficient، reliable، and user-friendly solution for managing student information in educational institutions. The primary aim is to streamline and automate administrative processes، reducing the burden on staff and minimizing errors associated with manual data entry. By centralizing student data، the system ensures that information is accurate، up-to-date، and easily accessible to authorized users.

One of the key purposes of the SMS is to enhance administrative efficiency by automating routine tasks such as student enrollment، registration، and fee processing. This automation saves time and resource administrative staff to focus on more strategic activities. Improving data management is another critica as centralizing student records ensures data consistency and easy access for students، teachers، and administrative staff.

٢. LITERATURE SURVEY

Firstly، examining various existing SMS solutions provides insights into their features، functionalities، and implementation challenges. Studies often highlight how these systems automate tasks like enrollment، registration، academic record management، and fee processing، thereby enhancing administrative efficiency and reducing errors.

Technological frameworks play a crucial role in SMS development. Literature reviews typically explore the use of web-based applications، database management systems، and secu protocols to ensure data integrity and privacy compliance. Understanding these frameworks h designing robust، scalable systems that meet institutional needs while safeguarding sensitive information.

User experience (UX) and interface design principles are critical in SMS development. Research often focuses on usability، accessibility، and user satisfaction aspects، examining hov designed interfaces contribute to effective system adoption and user engagement. Studies m explore communication and collaboration tools integrated into SMS، such as messaging syste virtual classrooms، to facilitate seamless interaction among students، teachers، and administra

Data security remains a paramount concern in managing student information. Literature reviews typically delve into security measures and privacy concerns specific to educat environments، discussing encryption standards، access controls، and compliance with data pr regulations. Insights from these studies inform best practices for ensuring data security while maintaining system usability and accessibility.

Case studies provide valuable real-world examples of SMS implementations across different educational contexts. These studies highlight implementation challenges، user acc issues، and strategies for overcoming barriers to successful deployment. Additionally، evaluat impact of SMS on educational outcomes، such as student performance، satisfaction، and instit effectiveness، offers valuable insights into the system's effectiveness and potential areas for improvement.

٣. SYSTEM ANALYSIS

٣.١. Existing System:

Presently to maintain information about different aspects، the faculty is using manual process i.e.، using books and ledgers. Now the faculty requires a computerized environment where it is easy for storing information about student details، their attendance، marks reports، assignment work details، and schedules and so on.

Using present manualprocess it is difficult in maintaining data and moreover it is time consuming. There is wastage of stationary and more human resources effort is required. When we have computerizedenvIronment it replaces all these problem.

٣.٢. Proposed system: The mainobjective of assignment project management is to automate the assignment process of work. Using this process a student can perform assignment related work provided by faculty. This will be used by thestudents of the branch at the time of their assigning of work through the assigned Guide. Thisis designed to manage، assigning of work process entirely online .It is flexible enough toco-operate any changes or enhancements made later within the application.

٤. SOFTWARE REQUIREMENT SPECIFICATION

٤.١ HARDWARE REQUIREMENTS:

- System : Pentium IV ٢.٤ GHz.
- Hard Disk : ٤٠ GB.
- Floppy Drive : ١.٤٤ Mb.
- Monitor : ١٥ VGA Colour.
- Mouse : Logitech.
- Ram : ٥١٢ Mb.

٤.٢ SOFTWARE REQUIREMENTS:

- Operating system : Windows ١١/١٢
- Coding Language : DJANGO
- IDE : PyChram
- Database : sqlite٣

٥. SOFTWARE DESCRIPTION

Django

Django is a Python framework that makes it easier to create web sites using Python. Django takes care of the difficult stuff so that you can concentrate on building your web applications.

Django emphasizes reusability of components, also referred to as DRY (Don't Repeat Yourself), and comes with ready-to-use features like login system, database connection and CRUD operations (Create Read Update Delete). Django is especially helpful for database driven websites.

Django follows the MVT design pattern (Model View Template).

- Model – The data you want to present, usually data from a database.

  The models are usually located in a file called models.py.

- View – A request handler that returns the relevant template and content – based on the request from the user.

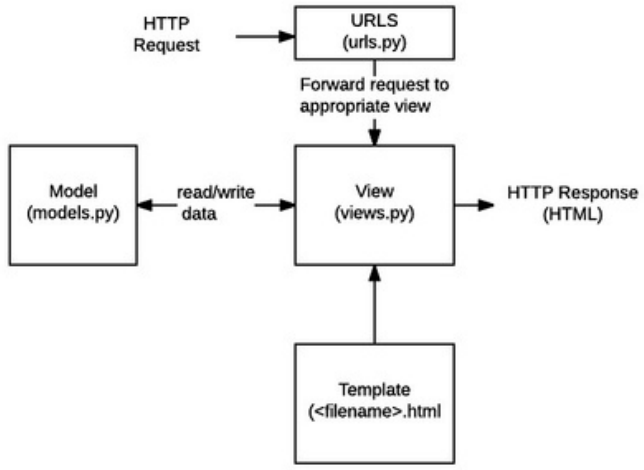
  The views are usually located in a file called views.py.

- Template – A text file (like an HTML file) containing the layout of the web page, with logic on how to display the data.

  The templates of an application is located in a folder named templates.

URLs

Django also provides a way to navigate around the different pages in a website. When a user requests a URL، Django decides which view it will send it to. It is done in file called urls.py.



Defining data models (models.py)

Django web applications manage and query data through Python objects referred to as models. Models define the structure of stored data، including the field types and possibly also their maximum size، default values، selection list options، help text for documentation، label text for forms، etc. The definition of the model is independent of the underlying database — you can choose one of several as part of your project settings. Once you've chosen what database you want to use، you don't need to talk to it directly at all — you just write your model structure and other code، and Django handles all the "dirty work" of communicating with the database for you.

The code snippet below shows a very simple Django model for a Team object. The Team class is derived from the Django class models.Model. It defines the team name and team level as character fields and specifies a maximum number of characters to be stored for each record. The team_level can be one of several values، so we define it as a choice field and provide a mapping between choices to be displayed and data to be stored، along with a default value.

EX: filename: models.py

```
from django.db import models

class Team(models.Model):
    team_name = models.CharField(max_length=٤٠)
    TEAM_LEVELS = (
        ('U٠٩', 'Under ٠٩s'),
```

...

list other team level)

team__level = models.CharField(max__length=٣, choices=TEAM__LEVELS, default='U١١')

Querying data (views.py)

The Django model provides a simple query API for searching the associated database. This can match against a number of fields at a time using different criteria (e.g. exact, case-insensitive, greater than, etc.), and can support complex statements (for example, you can specify a search on U١١ teams that have a team name that starts with "Fr" or ends with "al").

The code snippet shows a view function (resource handler) for displaying all of our U٠٩ teams. The list__teams = Team.objects.filter(team__level___exact="U٠٩") line shows how we can use the model query API to filter for all records where the team__level field has exactly the text 'U٠٩' (note how this criteria is passed to the filter() function as an argument, with the field name and match type separated by a double underscore: team__level___exact).

EX: filename: views.py

from django.shortcuts import render

from .models import Team

def index(request):

list__teams = Team.objects.filter(team__level___exact="U٠٩")

context = {'youngest__teams': list__teams}

return render(request, '/best/index.html', context)

This function uses the render() function to create the HttpResponse that is sent back to the browser. This function is a shortcut; it creates an HTML file by combining a specified HTML template and some data to insert in the template (provided in the variable named "context"). In the next section we show how the template has the data inserted in it to create the HTML.

Rendering data (HTML templates)

Template systems allow you to specify the structure of an output document, using placeholders for data that will be filled in when a page is generated. Templates are often used to create HTML, but can also

create other types of document. Django supports both its native templating system and another popula
Python library called Jinja۲ out of the box (it can also be made to support other systems if needed).

The code snippet shows what the HTML template called by the render() function in the previous section might look like. This template has been written under the assumption that it will have access to a list variable called youngest_teams when it is rendered (this is contained in the context variable inside the render() function above). Inside the HTML skeleton we have an expression that first checks if the youngest_teams variable exists, and then iterates it in a for loop. On each iteration the template displays each team's team_name value in an <li> element.

Example:

filename: best/templates/best/index.html

```
<html lang="en">
<head>
  <title>Home page</title>
</head>
<body>
  {% if youngest_teams %}
    <ul>
      {% for team in youngest_teams %}
        <li>{{ team.team_name }}</li>
      {% endfor %}
    </ul>
  {% else %}
    <p>No teams are available.</p>
  {% endif %}
```

> /body<

> /html<

Create a basic Project :

- To initiate a project of Django on Your PC , open Terminal and Enter the following command

  django-admin startproject projectName

- A New Folder with the name projectName will be created . To enter in the project using the terminal enter command

  cd projectName

- Create a new file views . py inside the project folder where settings . py , urls . py and other files are stored and save the following code in it-

  HttpResponse is used to

  pass the information

  back to view

  from django . http import HttpResponse

  Defining a function which

  will receive request and

  perform task depending

  upon function definition

  def hello__geeks (request) :

  This will return Hello Geeks

  string as HttpResponse

  return HttpResponse("Hello Geeks")

- Open urls.py inside project folder (projectName) and add your entry-

  Import hello_geeks function from views.py file.

  from projectName.views import hello_geeks

- Addanentry in url field inside url patterns-

  path('geek/', hello_geeks),

- Nowtorun the server follow these steps-

  Opencommand prompt and change directory to env_site by this command-

  <python<scripts<projectName

- Start the server- Start the server by typing following command in cmd-

  ➢ python manage.py runserver

- Checking – Open the browser and type this url-

  http://١٢٧.٠.٠.١:٨٠٠٠/

Creating an App in Django :

Let us start building an app.

Method-١

- To create a basic app in your Django project you need to go to the directory containing manage.py and from there enter the command :

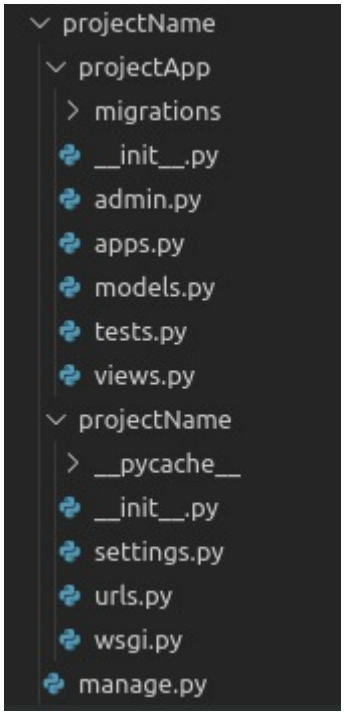
python manage.py startapp projectApp

Method-٢

- To create a basic app in your Django project you need to go to the directory containing manage.py and from there enter the command :

django-admin startapp projectApp

- Now you can see your directory structure as under :



To consider the app in yourproject you need to specify your project name in INSTALLED__APPS list as follows in settings.py:

- Python

```
Application definition

INSTALLED__APPS = (

    'django.contrib.admin',

    'django.contrib.auth',

    'django.contrib.contenttypes',

    'django.contrib.sessions',

    'django.contrib.messages',

    'django.contrib.staticfiles',

    'projectApp'

)
```

So، we have finally created an app but to render the app using URLs we need to include the app in our main project so that URLs redirected to that app can be rendered. Let us explore it.
Move to projectName -< projectName -< urls.py and add below code in the header

from django.urls import includeNow in the list of URL patterns، you need to specify the app name for including your app URLs. Here is the code for it –

- Python۳

```
from django.contrib import admin

from django.urls import path، include

 urlpatterns = ﷺ path('admin/'، admin.site.urls)،

   path(''، include("projectApp.urls"))،

ﷺ
```

- Now You can use the default MVT model to create URLs، models، views، etc. in your app and they will be automatically included in your main project.

The main feature of Django Apps is independence، every app functions as an independent unit in supporting the main project.

Now the urls.py in the project file will not access the app's url.

To run your Django Web application properly the following actions must be taken:-

۱. Create a file in the apps directory called urls.py

۲. Include the following code:

- Python۳

٦. SYSTEM DESIGN

٦.١. Introduction

A Student Management System (SMS) is a software application that assists educational institutions in managing student data efficiently. These systems are designed to streamline various administrative and academic tasks, ensuring smooth operation and easy access to information. By automating routine processes, SMS helps institutions save time and resources, enabling staff to focus more on educational quality and less on paperwork.

One of the primary features of a Student Management System is student enrollment and admissions management. The system simplifies the entire admission process, from application submission to approval. It helps manage student enrollment, re-enrollment, and transfers, ensuring that all necessary data is captured accurately and efficiently. This not only reduces administrative workload but also minimizes errors and enhances the overall experience for both staff and students.

٦.٢. System architecture

The architecture of a Student Management System (SMS) typically involves several layers and components that work together to ensure efficient data management and functionality. Below is an outline of the key components and their interactions in the system architecture.

١. Presentation Layer

This is the top layer, which interacts directly with the users (students, teachers, administrators, and parents). It includes:

- **User Interface (UI)**: The web or mobile interface through which users interact with the system. It consists of dashboards, forms, and reports.
- **Authentication and Authorization**: Manages user login, registration, and role-based access control to ensure security.

٢. Application Layer

This layer contains the core logic of the SMS. It handles the processing of user requests and implements the business rules. Key components include:

٣. Data Access Layer

This layer provides a set of interfaces for accessing and manipulating the data stored in the database. It ensures that the application layer can interact with the data without worrying about the underlying database operations. Components include:

- **Data Access Objects (DAO)**: Objects responsible for accessing the database and performing CRUD (Create, Read, Update, Delete) operations.
- **ORM (Object-Relational Mapping)**: A framework that helps in mapping objects to database tables, simplifying database interactions (e.g., Hibernate).

٤. Database Layer

This is the bottom layer، responsible for data storage and management. It includes:

- **Database Management System (DBMS)**: The actual database that stores all student-related information, such as MySQL, PostgreSQL, or MongoDB.
- **Backup and Recovery**: Ensures data integrity and availability by implementing regular backup and recovery mechanisms.

٥. Integration Layer

This layer manages interactions with external systems and services. It includes:

- **APIs (Application Programming Interfaces)**: Interfaces for integrating with external systems like Learning Management Systems (LMS), payment gateways, and other educational tools.
- **Middleware**: Software that connects different applications and ensures they can communicate and share data seamlessly.

٦.٣. Uml diagrams

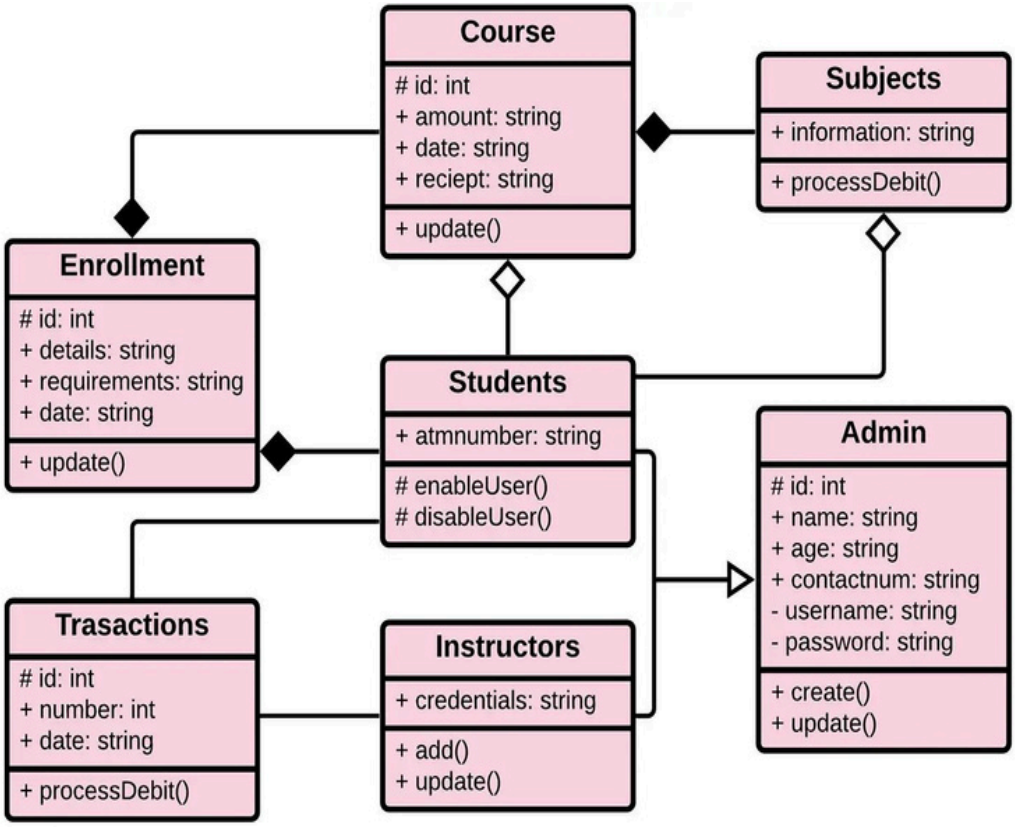
Actors:

- Administrator
- Teacher
- Student
- Parent

Use Cases:

- Manage Admissions
- Manage Student Information
- Track Attendance
- Manage Grades
- Communicate with Parents

Unified Modeling Language (UML) diagrams are essential for visualizing the design and architecture of the Student Management System (SMS). They help in understanding the system's components, their interactions, and the workflow. Below are detailed descriptions and examples of key UML diagrams for this project.



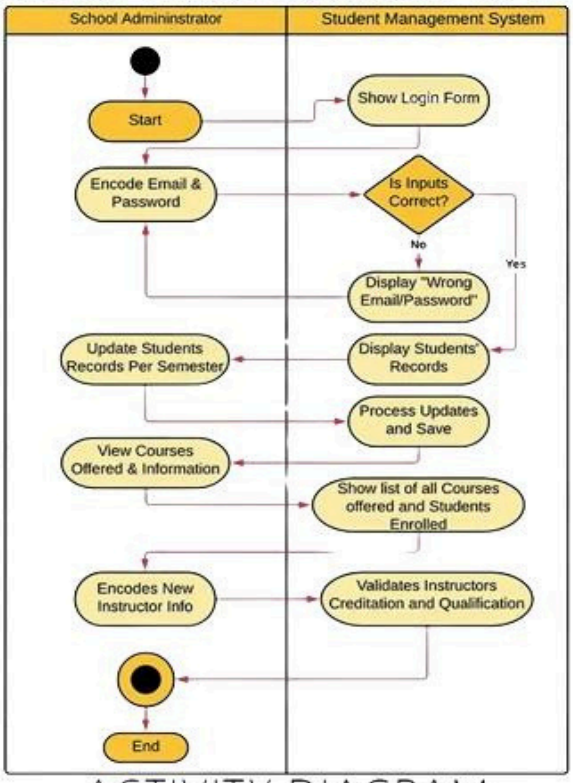
٢. Class Diagram

The Class Diagram illustrates the static structure of the system، showing the system's classes، their attributes، methods، and the relationships between them.

Classes:

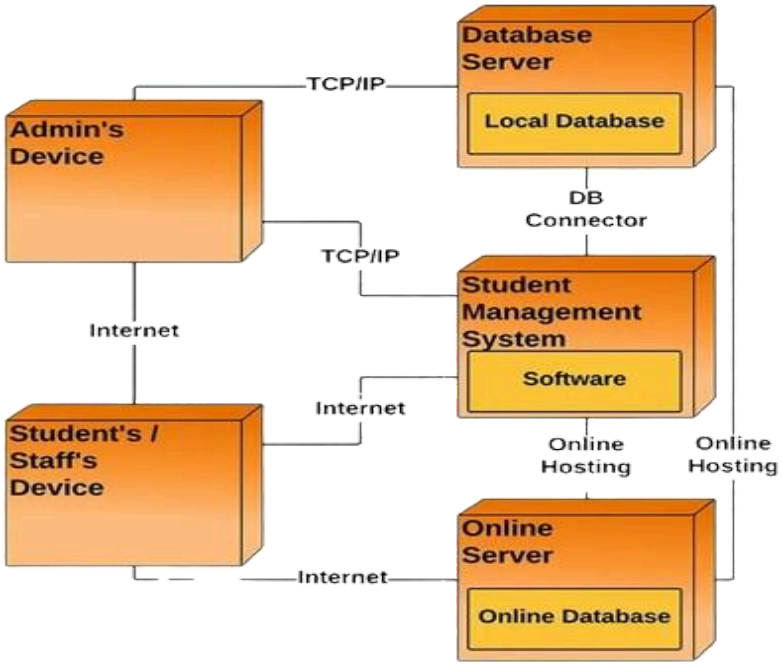
- User (attributes: userID, name, email; methods: login(), logout())
- Student (attributes: studentID, grade; methods: attendClass())
- Teacher (attributes: teacherID, subject; methods: assignGrade())
- Admin (attributes: adminID; methods: manageAdmission())
- Course (attributes: courseID, courseName; methods: enrollStudent())
- Attendance (attributes: attendanceID, date, status; methods: markAttendance())

STUDENT MANAGEMENT SYSTEM



ACTIVITY DIAGRAM

STUDENT MANAGEMENT SYSTEM



DEPLOYMENT DIAGRAM

٧. IMPLEMENTATION

٧.١ Modules of the project

١. User Management Module

2. Student Enrollment and Admissions Module

3. Attendance Management Module

٤. Grade and Exam Management Module

٧.٢. Module description

١. User Management Module

The User Management Module is essential for handling user authentication، authorization، and profile management. This module ensures that only authorized users can access the system by implementing robust login and logout mechanisms. It also manages user roles and permissions، providing role-based access control to different features and data within the system. Additionally، this module allows users to update their profiles، ensuring their information is current and accurate.

٢. Student Enrollment and Admissions Module

The Student Enrollment and Admissions Module streamlines the entire process of student admissions، from application submission to approval. This module allows administrators to manage new student applications، verify required documents، and handle the approval process efficiently. Once a student is admitted، this module manages their enrollment and re-enrollment procedures، ensuring that all necessary data is captured accurately and stored securely.

٣. Attendance Management Module

The Attendance Management Module simplifies the process of recording and tracking student attendance. Teachers can use this module to mark daily attendance، and the system automatically generates comprehensive reports that can be accessed by administrators and parents. This module also includes alert features، notifying parents and guardians of student absenteeism، thereby improving communication and ensuring students maintain regular attendance.

٤. Grade and Exam Management Module

The Grade and Exam Management Module is designed to manage the academic performance records of students. Teachers can use this module to record grades for various assignments، tests، and exams. The system can generate transcripts and report cards، providing a clear overview of a student's academic progress. Additionally، this module facilitates the scheduling and management of exams، ensuring that all necessary arrangements are handled efficiently.

۷.۳. Sample code

Manage.py

```
importos
importsys
defmain():
    os.environ.setdefault('DJANGO__SETTINGS__MODULE', 'student__management__system.settings')
    try:
        fromdjango.core.managemenimport 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)


if___name___ == '___main___':
    main()
```

Views.py

```
 from channels.auth import login, logout
from django.contrib.auth import authenticate, login, logout
from django.http import HttpResponseRedirect, HttpResponse
from django.shortcuts import render, redirect
from django.contrib import messages
from student__management__app.EmailBackEnd import EmailBackEnd
def home(request):
    return render(request, 'index.html')
def loginPage(request):
    return render(request, 'login.html')
def doLogin(request):
    if request.method != "POST":
        return HttpResponse("<h۲>Method Not Allowed</h۲>")
```

```
    else:
        user = EmailBackEnd.authenticate(request, username=request.POST.get('email'),
password=request.POST.get('password'))
        if user != None:
            login(request, user)
            user_type = user.user_type
            return HttpResponse("Email: "+request.POST.get('email')+ " Password:
"+request.POST.get('password'))
            ifuser_type== '১':
                returnredirect('admin_home')

            elifuser_type == '২':
                returnHttpResponse("Staff Login")
                returnredirect('staff_home')

            elifuser_type == '৩':
                returnHttpResponse("Student Login")
                returnredirect('student_home')
            else:
                messages.error(request, "Invalid Login!")
                returnredirect('login')
        else:
            messages.error(request, "Invalid Login Credentials!")
            returnHttpResponseRedirect("/")
            returnredirect('login')
defget_user_details(request):
    ifrequest.user!=None:
        returnHttpResponse("User: "+request.user.email+" User Type:
"+request.user.user_type)
    else:
        return HttpResponse("Please Login First")
def logout_user(request):
    logout(request)
```

```
    return HttpResponseRedirect('/')
```

URLS.py

```
from django.urls import path, include
from . import views
from .import HodViews, StaffViews, StudentViews
urlpatterns = ﷺ
    path('', views.loginPage, name="login"),
    path('accounts/', include('django.contrib.auth.urls')),
    path('doLogin/', views.doLogin, name="doLogin"),
    path('get_user_details/', views.get_user_details, name="get_user_details"),
    path('logout_user/', views.logout_user, name="logout_user"),
    path('admin_home/', HodViews.admin_home, name="admin_home"),
    path('add_staff/', HodViews.add_staff, name="add_staff"),
    path('add_staff_save/', HodViews.add_staff_save, name="add_staff_save"),
    path('manage_staff/', HodViews.manage_staff, name="manage_staff"),
    path('edit_staff/<staff_id>/', HodViews.edit_staff, name="edit_staff"),
    path('edit_staff_save/', HodViews.edit_staff_save, name="edit_staff_save"),
    path('delete_staff/<staff_id>/', HodViews.delete_staff, name="delete_staff"),
    path('add_course/', HodViews.add_course, name="add_course"),
    path('add_course_save/', HodViews.add_course_save, name="add_course_save"),
    path('manage_course/', HodViews.manage_course, name="manage_course"),
    path('edit_course/<course_id>/', HodViews.edit_course, name="edit_course"),
    path('edit_course_save/', HodViews.edit_course_save, name="edit_course_save"),
    path('delete_course/<course_id>/', HodViews.delete_course, name="delete_course"),
    path('manage_session/', HodViews.manage_session, name="manage_session"),
    path('add_session/', HodViews.add_session, name="add_session"),
    path('add_session_save/', HodViews.add_session_save, name="add_session_save"),
    path('edit_session/<session_id>', HodViews.edit_session, name="edit_session"),
    path('edit_session_save/', HodViews.edit_session_save, name="edit_session_save"),
    path('delete_session/<session_id>/', HodViews.delete_session, name="delete_session"),
```

```
path('add_student/', HodViews.add_student, name="add_student"),
    path('add_student_save/', HodViews.add_student_save, name="add_student_save"),
    path('edit_student/<student_id>', HodViews.edit_student, name="edit_student"),
    path('edit_student_save/', HodViews.edit_student_save, name="edit_student_save"),
    path('manage_student/', HodViews.manage_student, name="manage_student"),
    path('delete_student/<student_id>/', HodViews.delete_student, name="delete_student"),
    path('add_subject/', HodViews.add_subject, name="add_subject"),
    path('add_subject_save/', HodViews.add_subject_save, name="add_subject_save"),
    path('manage_subject/', HodViews.manage_subject, name="manage_subject"),
    path('edit_subject/<subject_id>/', HodViews.edit_subject, name="edit_subject"),
    path('edit_subject_save/', HodViews.edit_subject_save, name="edit_subject_save"),
    path('delete_subject/<subject_id>/', HodViews.delete_subject, name="delete_subject"),
    path('check_email_exist/', HodViews.check_email_exist, name="check_email_exist"),
    path('check_username_exist/', HodViews.check_username_exist,
name="check_username_exist"),
    path('student_feedback_message/', HodViews.student_feedback_message,
name="student_feedback_message"),
    path('student_feedback_message_reply/', HodViews.student_feedback_message_rep
name="student_feedback_message_reply"),
    path('staff_feedback_message/', HodViews.staff_feedback_message,
name="staff_feedback_message"),
    path('staff_feedback_message_reply/', HodViews.staff_feedback_message_reply,
name="staff_feedback_message_reply"),
    path('student_leave_view/', HodViews.student_leave_view, name="student_leave_view
    path('student_leave_approve/<leave_id>/', HodViews.student_leave_approve,
name="student_leave_approve"),
    path('student_leave_reject/<leave_id>/', HodViews.student_leave_reject,
name="student_leave_reject"),
    path('staff_leave_view/', HodViews.staff_leave_view, name="staff_leave_view"),
    path('staff_leave_approve/<leave_id>/', HodViews.staff_leave_approve,
name="staff_leave_approve"),
    path('staff_leave_reject/<leave_id>/', HodViews.staff_leave_reject,
name="staff_leave_reject"),
    path('admin_view_attendance/', HodViews.admin_view_attendance,
```

```
name="admin_view_attendance"),
    path('admin_get_attendance_dates/', HodViews.admin_get_attendance_dates,
name="admin_get_attendance_dates"),
    path('admin_get_attendance_student/', HodViews.admin_get_attendance_student,
name="admin_get_attendance_student"),
    path('admin_profile/', HodViews.admin_profile, name="admin_profile"),
    path('admin_profile_update/', HodViews.admin_profile_update,
name="admin_profile_update"),
```

URLS for Staff

```
    path('staff_home/', StaffViews.staff_home, name="staff_home"),
    path('staff_take_attendance/', StaffViews.staff_take_attendance,
name="staff_take_attendance"),
    path('get_students/', StaffViews.get_students, name="get_students"),
    path('save_attendance_data/', StaffViews.save_attendance_data,
name="save_attendance_data"),
    path('staff_update_attendance/', StaffViews.staff_update_attendance,
name="staff_update_attendance"),
    path('get_attendance_dates/', StaffViews.get_attendance_dates,
name="get_attendance_dates"),
    path('get_attendance_student/', StaffViews.get_attendance_student,
name="get_attendance_student"),
    path('update_attendance_data/', StaffViews.update_attendance_data,
name="update_attendance_data"),
    path('staff_apply_leave/', StaffViews.staff_apply_leave, name="staff_apply_leave"),
    path('staff_apply_leave_save/', StaffViews.staff_apply_leave_save,
name="staff_apply_leave_save"),
    path('staff_feedback/', StaffViews.staff_feedback, name="staff_feedback"),
    path('staff_feedback_save/', StaffViews.staff_feedback_save,
name="staff_feedback_save"),
    path('staff_profile/', StaffViews.staff_profile, name="staff_profile"),
    path('staff_profile_update/', StaffViews.staff_profile_update,
name="staff_profile_update"),
```

```
path('staff__add__result/', StaffViews.staff__add__result, name="staff__add__result"),
    path('staff__add__result__save/', StaffViews.staff__add__result__save,
name="staff__add__result__save"),
```

URSL for Student

```
    path('student__home/', StudentViews.student__home, name="student__home"),
    path('student__view__attendance/', StudentViews.student__view__attendance,
name="student__view__attendance"),
    path('student__view__attendance__post/', StudentViews.student__view__attendance__post,
name="student__view__attendance__post"),
    path('student__apply__leave/', StudentViews.student__apply__leave,
name="student__apply__leave"),
    path('student__apply__leave__save/', StudentViews.student__apply__leave__save,
name="student__apply__leave__save"),
    path('student__feedback/', StudentViews.student__feedback, name="student__feedback"),
    path('student__feedback__save/', StudentViews.student__feedback__save,
name="student__feedback__save"),
    path('student__profile/', StudentViews.student__profile, name="student__profile"),
    path('student__profile__update/', StudentViews.student__profile__update,
name="student__profile__update"),
    path('student__view__result/', StudentViews.student__view__result,
name="student__view__result"),
```

ﷺ

Models.py

```
from django.contrib.auth.models import AbstractUser
from django.db import models
from django.db.models.signals import post__save

from django.dispatch import receiver
class SessionYearModel(models.Model):
    id = models.AutoField(primary__key=True)
    session__start__year = models.DateField()
    session__end__year = models.DateField()
```

```
    objects = models.Manager()
Overriding the Default Django Auth User and adding One More Field (user_type)
class CustomUser(AbstractUser):
    user_type_data = ((١, "HOD"), (٢, "Staff"), (٣, "Student"))
    user_type = models.CharField(default=١, choices=user_type_data, max_length=١٠)
class AdminHOD(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class Staffs(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    address = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

classCourses(models.Model):
    id=models.AutoField(primary_key=True)
    course_name = models.CharField(max_length=٢٥٥)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

    def__str__(self):
        return self.course_name


classSubjects(models.Model):
    id=models.AutoField(primary_key=True)
    subject_name = models.CharField(max_length=٢٥٥)
    course_id = models.ForeignKey(Courses, on_delete=models.CASCADE, default=١) need
to give defauult course
```

```
    staff_id = models.ForeignKey(CustomUser, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class Students(models.Model):
    id = models.AutoField(primary_key=True)
    admin = models.OneToOneField(CustomUser, on_delete = models.CASCADE)
    gender = models.CharField(max_length=٥٠)
    profile_pic = models.FileField()
    address = models.TextField()
    course_id = models.ForeignKey(Courses, on_delete=models.DO_NOTHING, default=١)
    session_year_id = models.ForeignKey(SessionYearModel, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class Attendance(models.Model):
    Subject Attendance
    id = models.AutoField(primary_key=True)
    subject_id = models.ForeignKey(Subjects, on_delete=models.DO_NOTHING)
    attendance_date = models.DateField()
    session_year_id = models.ForeignKey(SessionYearModel, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class AttendanceReport(models.Model):
    Individual Student Attendance
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.DO_NOTHING)
    attendance_id = models.ForeignKey(Attendance, on_delete=models.CASCADE)
    status = models.BooleanField(default=False)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class LeaveReportStudent(models.Model):
```

```
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)
    leave_date = models.CharField(max_length=٢٥٥)
    leave_message = models.TextField()
    leave_status = models.IntegerField(default=٠)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class LeaveReportStaff(models.Model):
    id = models.AutoField(primary_key=True)
    staff_id = models.ForeignKey(Staffs, on_delete=models.CASCADE)
    leave_date = models.CharField(max_length=٢٥٥)
    leave_message = models.TextField()
    leave_status = models.IntegerField(default=٠)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class FeedBackStudent(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)
    feedback = models.TextField()
    feedback_reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
class FeedBackStaffs(models.Model):
    id = models.AutoField(primary_key=True)
    staff_id = models.ForeignKey(Staffs, on_delete=models.CASCADE)
    feedback = models.TextField()
    feedback_reply = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
```

```
class NotificationStudent(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

class NotificationStaffs(models.Model):
    id = models.AutoField(primary_key=True)
    stafff_id = models.ForeignKey(Staffs, on_delete=models.CASCADE)
    message = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()

class StudentResult(models.Model):
    id = models.AutoField(primary_key=True)
    student_id = models.ForeignKey(Students, on_delete=models.CASCADE)
    subject_id = models.ForeignKey(Subjects, on_delete=models.CASCADE)
    subject_exam_marks = models.FloatField(default=0)
    subject_assignment_marks = models.FloatField(default=0)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)
    objects = models.Manager()
```

Creating Django Signals

It's like trigger in database. It will run only when Data is Added in CustomUser model

```
@receiver(post_save, sender=CustomUser)
```

Now Creating a Function which will automatically insert data in HOD, Staff or Student

```
def create_user_profile(sender, instance, created, **kwargs):
    if Created is true (Means Data Inserted)
    if created:
```

```
        Check the user__type and insert the data in respective tables
        if instance.user__type == ١:
            AdminHOD.objects.create(admin=instance)
        if instance.user__type == ٢:
            Staffs.objects.create(admin=instance)
        if instance.user__type == ٣:
            Students.objects.create(admin=instance, course__id=Courses.objects.get(id=١),
session__year__id=SessionYearModel.objects.get(id=١), address="", profile__pic="",
gender="")
    @receiver(post__save, sender=CustomUser)
defsave__user__profile(sender, instance, **kwargs):
    ifinstance.user__type == ١:
        instance.adminhod.save()
    ifinstance.user__type == ٢:
        instance.staffs.save()
    ifinstance.user__type == ٣:
        instance.students.save()
```

Admin.py

```
from django.contrib import admin
from django.contrib.auth.admin import UserAdmin
from .models import CustomUser, AdminHOD, Staffs, Courses, Subjects, Students,
Attendance, AttendanceReport, LeaveReportStudent, LeaveReportStaff, FeedBackStudent,
FeedBackStaffs, NotificationStudent, NotificationStaffs
    Register your models here.
class UserModel(UserAdmin):
    pass
admin.site.register(CustomUser,
UserModel)
admin.site.register(AdminHOD)
admin.site.register(Staffs)
admin.site.register(Courses)
admin.site.register(Subjects)
admin.site.register(Students)
```

```
admin.site.register(Attendance)
admin.site.register(AttendanceReport)
admin.site.register(LeaveReportStudent)
admin.site.register(LeaveReportStaff)
admin.site.register(FeedBackStudent)
admin.site.register(FeedBackStaffs)
admin.site.register(NotificationStudent)
admin.site.register(NotificationStaffs)
```

app.py

```
from django.apps import AppConfig
class StudentManagementAppConfig(AppConfig):
    name = 'student_management_app'
```

settings.py

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'student_management_app',
]
```

forms.py

```
from django import forms
from django.forms import Form
from student_management_app.models import Courses, SessionYearModel
class DateInput(forms.DateInput):
    input_type = "date"
```

```
class AddStudentForm(forms.Form):
    email = forms.EmailField(label="Email", max_length=50,
widget=forms.EmailInput(attrs={"class":"form-control"}))
    password = forms.CharField(label="Password", max_length=50,
widget=forms.PasswordInput(attrs={"class":"form-control"}))
    first_name = forms.CharField(label="First Name", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    last_name = forms.CharField(label="Last Name", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    username = forms.CharField(label="Username", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    address = forms.CharField(label="Address", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    ﷺFor Displaying Courses
    try:
        courses = Courses.objects.all()
        course_list = ﷺ
        for course in courses:
            single_course = (course.id, course.course_name)
            course_list.append(single_course)
    except:
        course_list = ﷺ
    ﷺFor Displaying Session Years
    try:
        session_years = SessionYearModel.objects.all()
        session_year_list = ﷺ
        for session_year in session_years:
            single_session_year = (session_year.id, str(session_year.session_start_year)+"to
"+str(session_year.session_end_year))
            session_year_list.append(single_session_year)
    except:
        session_year_list = ﷺ
    gender_list = (
        ('Male','Male'),
```

```
        ('Female','Female')
    )
    course_id = forms.ChoiceField(label="Course", choices=course_list,
widget=forms.Select(attrs={"class":"form-control"}))
    gender = forms.ChoiceField(label="Gender", choices=gender_list,
widget=forms.Select(attrs={"class":"form-control"}))
    session_year_id = forms.ChoiceField(label="Session Year", choices=session_year_list,
widget=forms.Select(attrs={"class":"form-control"}))
    session_start_year = forms.DateField(label="Session Start",
widget=DateInput(attrs={"class":"form-control"}))
    session_end_year = forms.DateField(label="Session End",
widget=DateInput(attrs={"class":"form-control"}))
    profile_pic = forms.FileField(label="Profile Pic", required=False,
widget=forms.FileInput(attrs={"class":"form-control"}))
class EditStudentForm(forms.Form):
    email = forms.EmailField(label="Email", max_length=50,
widget=forms.EmailInput(attrs={"class":"form-control"}))
    first_name = forms.CharField(label="First Name", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    last_name = forms.CharField(label="Last Name", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    username = forms.CharField(label="Username", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))
    address = forms.CharField(label="Address", max_length=50,
widget=forms.TextInput(attrs={"class":"form-control"}))


    ForDisplaying Courses
    try:
        courses = Courses.objects.all()
        course_list = ﷺ
        forcourse in courses:
            single_course = (course.id, course.course_name)
            course_list.append(single_course)
```

```
    except:
        course_list = ﷺ


    ForDisplaying Session Years
    try:
        session_years = SessionYearModel.objects.all()
        session_year_list = ﷺ
        forsession_year in session_years:
            single_session_year = (session_year.id, str(session_year.session_start_year)+" to
"+str(session_year.session_end_year))
            session_year_list.append(single_session_year)
    course_id = forms.ChoiceField(label="Course", choices=course_list,
widget=forms.Select(attrs={"class":"form-control"}))
    gender = forms.ChoiceField(label="Gender", choices=gender_list,
widget=forms.Select(attrs={"class":"form-control"}))
    session_year_id = forms.ChoiceField(label="Session Year", choices=session_year_list,
widget=forms.Select(attrs={"class":"form-control"}))
    session_start_year = forms.DateField(label="Session Start",
widget=DateInput(attrs={"class":"form-control"}))
    session_end_year =
forms.DateField(label="Sessionwidget=DateInput(attrs={"class":"form-control"}))
    profile_pic = forms.FileField(label="Profile Pic", required=False,
widget=forms.FileInput(attrs={"class":"form-control"}))
```

EmailBackEnd.py

```
fromdjango.contrib.auth import get_user_model

fromdjango.contrib.auth.backends import ModelBackend

classEmailBackEnd(ModelBackend):

    defauthenticate(self, username=None, password=None, **kwargs):

        UserModel = get_user_model()
```

```
try:
        user= UserModel.objects.get(email=username)
    exceptUserModel.DoesNotExist:
        return None
    else:
        ifuser.check_password(password):
            returnuser
    return None
```

HodViews

```
from django.shortcuts import render, redirect
from django.http import HttpResponse, HttpResponseRedirect, JsonResponse
from django.contrib import messages
from django.core.files.storage import FileSystemStorage To upload Profile Picture
from django.urls import reverse
from django.views.decorators.csrf import csrf_exempt
from django.core import serializers
import json


from student_management_app.models import CustomUser, Staffs, Courses, Subjects,
Students, SessionYearModel, FeedBackStudent, FeedBackStaffs, LeaveReportStudent,
LeaveReportStaff, Attendance, AttendanceReport

from .forms import AddStudentForm, EditStudentForm
```

```
defadmin_home(request):
    all_student_count = Students.objects.all().count()
    subject_count = Subjects.objects.all().count()
    course_count = Courses.objects.all().count()
    staff_count = Staffs.objects.all().count()

    Total Subjects and students in Each Course
    course_all = Courses.objects.all()
    course_name_list = []
    subject_count_list = []
    student_count_list_in_course = []

    forcourse in course_all:
        subjects = Subjects.objects.filter(course_id=course.id).count()
        students = Students.objects.filter(course_id=course.id).count()
        course_name_list.append(course.course_name)
        subject_count_list.append(subjects)
        student_count_list_in_course.append(students)

    subject_all = Subjects.objects.all()
    subject_list = []
    student_count_list_in_subject = []
```

```
forsubject in subject__all:
    course = Courses.objects.get(id=subject.course__id.id)
    student__count = Students.objects.filter(course__id=course.id).count()
    subject__list.append(subject.subject__name)
    student__count__list__in__subject.append(student__count)
```

For Saffs

```
staff__attendance__present__list=
staff__attendance__leave__list=
staff__name__list=

staffs = Staffs.objects.all()
forstaff in staffs:
    subject__ids = Subjects.objects.filter(staff__id=staff.admin.id)
    attendance = Attendance.objects.filter(subject__id___in=subject__ids).count()
    leaves = LeaveReportStaff.objects.filter(staff__id=staff.id، leave__status=١).count()
    staff__attendance__present__list.append(attendance)
    staff__attendance__leave__list.append(leaves)
    staff__name__list.append(staff.admin.first__name)
```

For Students

```
student__attendance__present__list=
student__attendance__leave__list=
```

```
    student__name__list=ﷺ


    students = Students.objects.all()
    forstudent in students:
        attendance = AttendanceReport.objects.filter(student__id=student.id، status=True).count
        absent = AttendanceReport.objects.filter(student__id=student.id، status=False).count()
        leaves = LeaveReportStudent.objects.filter(student__id=student.id،
leave__status=١).count()
        student__attendance__present__list.append(attendance)
        student__attendance__leave__list.append(leaves+absent)
        student__name__list.append(student.admin.first__name)



    context=﴿
        "all__student__count": all__student__count،
        "subject__count": subject__count،
        "course__count": course__count،
        "staff__count": staff__count،
        "course__name__list": course__name__list،
        "subject__count__list": subject__count__list،
        "student__count__list__in__course": student__count__list__in__course،
        "subject__list": subject__list،
        "student__count__list__in__subject": student__count__list__in__subject،
```

```
        "staff__attendance__present__list": staff__attendance__present__list,
        "staff__attendance__leave__list": staff__attendance__leave__list,
        "staff__name__list": staff__name__list,
        "student__attendance__present__list": student__attendance__present__list,
        "student__attendance__leave__list": student__attendance__leave__list,
        "student__name__list": student__name__list,
    }
    return render(request, "hod__template/home__content.html", context)


def add__staff(request):
    return render(request, "hod__template/add__staff__template.html")


def add__staff__save(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method")
        return redirect('add__staff')
    else:
        first__name = request.POST.get('first__name')
        last__name = request.POST.get('last__name')
        username = request.POST.get('username')
        email = request.POST.get('email')
```

```
        password = request.POST.get('password')
        address = request.POST.get('address')

        try:
            user = CustomUser.objects.create_user(username=username, password=password, email=email, first_name=first_name, last_name=last_name, user_type=2)
            user.staffs.address = address
            user.save()
            messages.success(request, "Staff Added Successfully!")
            return redirect('add_staff')
        except:
            messages.error(request, "Failed to Add Staff!")
            return redirect('add_staff')



def manage_staff(request):
    staffs=Staffs.objects.all()
    context={
        "staffs": staffs
    }
    return render(request, "hod_template/manage_staff_template.html", context)
```

```
defedit_staff(request, staff_id):
    staff = Staffs.objects.get(admin=staff_id)

    context = {
        "staff": staff,
        "id": staff_id
    }
    return render(request, "hod_template/edit_staff_template.html", context)



defedit_staff_save(request):
    ifrequest.method != "POST":
        return HttpResponse("<h2>Method Not Allowed</h2>")
    else:
        staff_id = request.POST.get('staff_id')
        username = request.POST.get('username')
        email = request.POST.get('email')
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        address = request.POST.get('address')

        try:
```

```
        # INSERTING into Customuser Model
        user = CustomUser.objects.get(id=staff_id)
        user.first_name = first_name
        user.last_name = last_name
        user.email = email
        user.username = username
        user.save()

        # INSERTING into Staff Model
        staff_model = Staffs.objects.get(admin=staff_id)
        staff_model.address = address
        staff_model.save()

        messages.success(request, "Staff Updated Successfully.")
        return redirect('/edit_staff/'+staff_id)

    except:
        messages.error(request, "Failed to Update Staff.")
        return redirect('/edit_staff/'+staff_id)


def delete_staff(request, staff_id):
```

```
    staff=Staffs.objects.get(admin=staff_id)
    try:
        staff.delete()
        messages.success(request, "Staff Deleted Successfully.")
        returnredirect('manage_staff')
    except:
        messages.error(request, "Failed to Delete Staff.")
        returnredirect('manage_staff')




defadd_course(request):
    returnrender(request, "hod_template/add_course_template.html")



defadd_course_save(request):
    ifrequest.method != "POST":
        messages.error(request, "Invalid Method!")
        returnredirect('add_course')
    else:
        course= request.POST.get('course')
        try:
```

```
            course_model = Courses(course_name=course)
            course_model.save()
            messages.success(request, "Course Added Successfully!")
            returnredirect('add_course')
        except:
            messages.error(request, "Failed to Add Course!")
            returnredirect('add_course')


defmanage_course(request):
    courses=Courses.objects.all()
    context = {
        "courses":courses
    }
    returnrender(request, 'hod_template/manage_course_template.html', context)


defedit_course(request, course_id):
    course=Courses.objects.get(id=course_id)
    context = {
        "course":course,
        "id":course_id
    }
```

```
    returnrender(request, 'hod_template/edit_course_template.html', context)


defedit_course_save(request):
    ifrequest.method != "POST":
        HttpResponse("Invalid Method")
    else:
        course_id = request.POST.get('course_id')
        course_name = request.POST.get('course')

        try:
            course = Courses.objects.get(id=course_id)
            course.course_name = course_name
            course.save()

            messages.success(request, "Course Updated Successfully.")
            return redirect('/edit_course/'+course_id)

        except:
            messages.error(request, "Failed to Update Course.")
            return redirect('/edit_course/'+course_id)
```

```
defdelete_course(request, course_id):
    course = Courses.objects.get(id=course_id)
    try:
        course.delete()
        messages.success(request, "Course Deleted Successfully.")
        return redirect('manage_course')
    except:
        messages.error(request, "Failed to Delete Course.")
        return redirect('manage_course')



defmanage_session(request):
    session_years = SessionYearModel.objects.all()
    context = {
        "session_years": session_years
    }
    return render(request, "hod_template/manage_session_template.html", context)



defadd_session(request):
    return render(request, "hod_template/add_session_template.html")
```

```
defadd__session__save(request):
    ifrequest.method!="POST":
        messages.error(request,"Invalid Method")
        return redirect('add__course')
    else:
        session__start__year=request.POST.get('session__start__year')
        session__end__year=request.POST.get('session__end__year')


        try:
            sessionyear=SessionYearModel(session__start__year=session__start__year,
session__end__year=session__end__year)
            sessionyear.save()
            messages.success(request,"Session Year added Successfully!")
            return redirect("add__session")
        except:
            messages.error(request,"Failed to Add Session Year")
            return redirect("add__session")



defedit__session(request,session__id):
    session__year=SessionYearModel.objects.get(id=session__id)
    context={
        "session__year":session__year
```

```
    }

    returnrender(request, "hod_template/edit_session_template.html", context)



defedit_session_save(request):
    ifrequest.method != "POST":
        messages.error(request, "Invalid Method!")
        return redirect('manage_session')
    else:
        session_id = request.POST.get('session_id')
        session_start_year = request.POST.get('session_start_year')
        session_end_year = request.POST.get('session_end_year')

        try:
            session_year = SessionYearModel.objects.get(id=session_id)
            session_year.session_start_year = session_start_year
            session_year.session_end_year = session_end_year
            session_year.save()

            messages.success(request, "Session Year Updated Successfully.")
            return redirect('/edit_session/'+session_id)
        except:
            messages.error(request, "Failed to Update Session Year.")
```

```
        return redirect('/edit_session/'+session_id)


def delete_session(request, session_id):
    session = SessionYearModel.objects.get(id=session_id)
    try:
        session.delete()
        messages.success(request, "Session Deleted Successfully.")
        return redirect('manage_session')
    except:
        messages.error(request, "Failed to Delete Session.")
        return redirect('manage_session')


def add_student(request):
    form = AddStudentForm()
    context = {
        "form": form
    }
    return render(request, 'hod_template/add_student_template.html', context)
```

```
defadd__student__save(request):
    ifrequest.method!="POST":
        messages.error(request,"Invalid Method")
        return redirect('add__student')
    else:
        form=AddStudentForm(request.POST,request.FILES)

        ifform.is__valid():
            first__name=form.cleaned__data['first__name']
            last__name=form.cleaned__data['last__name']
            username=form.cleaned__data['username']
            email=form.cleaned__data['email']
            password=form.cleaned__data['password']
            address=form.cleaned__data['address']
            session__year__id=form.cleaned__data['session__year__id']
            course__id=form.cleaned__data['course__id']
            gender=form.cleaned__data['gender']

            Getting Profile Pic first
            First Check whether the file is selected or not
            Upload only if file is selected
            if len(request.FILES)!=0:
```

```
            profile_pic = request.FILES['profile_pic']
            fs=FileSystemStorage()
            filename = fs.save(profile_pic.name, profile_pic)
            profile_pic_url = fs.url(filename)
        else:
            profile_pic_url = None


        try:
            user= CustomUser.objects.create_user(username=username, password=password, email=email,first_name=first_name, last_name=last_name, user_type=3)
            user.students.address = address

            course_obj = Courses.objects.get(id=course_id)
            user.students.course_id = course_obj

            session_year_obj = SessionYearModel.objects.get(id=session_year_id)
            user.students.session_year_id = session_year_obj

            user.students.gender = gender
            user.students.profile_pic = profile_pic_url
            user.save()
            messages.success(request, "Student Added Successfully!")
```

```
            returnredirect('add__student')
        except:
            messages.error(request, "Failed to Add Student!")
            returnredirect('add__student')
    else:
        returnredirect('add__student')


defmanage__student(request):
    students=Students.objects.all()
    context = {
        "students":students
    }
    returnrender(request, 'hod__template/manage__student__template.html', context)


defedit__student(request, student__id):
    AddingStudent ID into Session Variable
    request.session['student__id'] = student__id

    student=Students.objects.get(admin=student__id)
    form=EditStudentForm()
    Fillingtheform with Data from Database
```

```
    form.fields['email'].initial = student.admin.email
    form.fields['username'].initial = student.admin.username
    form.fields['first_name'].initial = student.admin.first_name
    form.fields['last_name'].initial = student.admin.last_name
    form.fields['address'].initial = student.address
    form.fields['course_id'].initial = student.course_id.id
    form.fields['gender'].initial = student.gender
    form.fields['session_year_id'].initial = student.session_year_id.id

    context = {
        "id": student_id,
        "username": student.admin.username,
        "form": form
    }
    return render(request, "hod_template/edit_student_template.html", context)


def edit_student_save(request):
    if request.method != "POST":
        return HttpResponse("Invalid Method!")
    else:
        student_id = request.session.get('student_id')
        if student_id == None:
```

```
    return redirect('/manage_student')


form= EditStudentForm(request.POST, request.FILES)
ifform.is_valid():
    email = form.cleaned_data['email']
    username = form.cleaned_data['username']
    first_name = form.cleaned_data['first_name']
    last_name = form.cleaned_data['last_name']
    address = form.cleaned_data['address']
    course_id = form.cleaned_data['course_id']
    gender = form.cleaned_data['gender']
    session_year_id = form.cleaned_data['session_year_id']


    Getting Profile Pic first
    First Check whether the file is selected or not
    Upload only if file is selected
    iflen(request.FILES)!=0:
        profile_pic = request.FILES['profile_pic']
        fs = FileSystemStorage()
        filename = fs.save(profile_pic.name, profile_pic)
        profile_pic_url = fs.url(filename)
    else:
        profile_pic_url = None
```

```
try:
    # First Update into Custom User Model
    user = CustomUser.objects.get(id=student_id)
    user.first_name = first_name
    user.last_name = last_name
    user.email = email
    user.username = username
    user.save()

    # Then Update Students Table
    student_model = Students.objects.get(admin=student_id)
    student_model.address = address

    course = Courses.objects.get(id=course_id)
    student_model.course_id = course

    session_year_obj = SessionYearModel.objects.get(id=session_year_id)
    student_model.session_year_id = session_year_obj

    student_model.gender = gender
    ifprofile_pic_url != None:
        student_model.profile_pic = profile_pic_url
```

```
            student__model.save()
            Delete student__id SESSION after the data is updated
            delrequest.session['student__id']

            messages.success(request, "Student Updated Successfully!")
            returnredirect('/edit__student/'+student__id)
        except:
            messages.success(request, "Failed to Uupdate Student.")
            returnredirect('/edit__student/'+student__id)
    else:
        returnredirect('/edit__student/'+student__id)


def delete__student(request, student__id):
    student=Students.objects.get(admin=student__id)
    try:
        student.delete()
        messages.success(request, "Student Deleted Successfully.")
        returnredirect('manage__student')
    except:
        messages.error(request, "Failed to Delete Student.")
        returnredirect('manage__student')
```

```
    try:
        subject.delete()
        messages.success(request, "Subject Deleted Successfully.")
        return redirect('manage_subject')
    except:
        messages.error(request, "Failed to Delete Subject.")
        return redirect('manage_subject')


@csrf_exempt
defcheck_email_exist(request):
    email = request.POST.get("email")
    user_obj = CustomUser.objects.filter(email=email).exists()
    ifuser_obj:
        return HttpResponse(True)
    else:
        return HttpResponse(False)


@csrf_exempt
def check_username_exist(request):
    username = request.POST.get("username")
    user_obj = CustomUser.objects.filter(username=username).exists()
```

```
    if user_obj:
        return HttpResponse(True)
    else:
        return HttpResponse(False)




def student_feedback_message(request):
    feedbacks = FeedBackStudent.objects.all()
    context = {
        "feedbacks": feedbacks
    }
    return render(request, 'hod_template/student_feedback_template.html', context)



@csrf_exempt
def student_feedback_message_reply(request):
    feedback_id = request.POST.get('id')
    feedback_reply = request.POST.get('reply')


    try:
        feedback = FeedBackStudent.objects.get(id=feedback_id)
        feedback.feedback_reply = feedback_reply
```

```
        feedback.save()
        return HttpResponse("True")

    except:
        return HttpResponse("False")



def staff_feedback_message(request):
    feedbacks = FeedBackStaffs.objects.all()
    context = {
        "feedbacks": feedbacks
    }
    return render(request, 'hod_template/staff_feedback_template.html', context)



@csrf_exempt
def staff_feedback_message_reply(request):
    feedback_id = request.POST.get('id')
    feedback_reply = request.POST.get('reply')

    try:
        feedback = FeedBackStaffs.objects.get(id=feedback_id)
        feedback.feedback_reply = feedback_reply
```

```
        feedback.save()
        return HttpResponse("True")

    except:
        return HttpResponse("False")


def student_leave_view(request):
    leaves = LeaveReportStudent.objects.all()
    context = {
        "leaves": leaves
    }
    return render(request, 'hod_template/student_leave_view.html', context)

def student_leave_approve(request, leave_id):
    leave = LeaveReportStudent.objects.get(id=leave_id)
    leave.leave_status = 1
    leave.save()
    return redirect('student_leave_view')


def student_leave_reject(request, leave_id):
    leave = LeaveReportStudent.objects.get(id=leave_id)
```

```
    leave.leave_status = ٢
    leave.save()
    return redirect('student_leave_view')


def staff_leave_view(request):
    leaves = LeaveReportStaff.objects.all()
    context = {
        "leaves": leaves
    }
    return render(request, 'hod_template/staff_leave_view.html', context)


def staff_leave_approve(request, leave_id):
    leave = LeaveReportStaff.objects.get(id=leave_id)
    leave.leave_status = ١
    leave.save()
    return redirect('staff_leave_view')


def staff_leave_reject(request, leave_id):
    leave = LeaveReportStaff.objects.get(id=leave_id)
    leave.leave_status = ٢
```

```
    leave.save()
    return redirect('staff_leave_view')


def admin_view_attendance(request):
    subjects = Subjects.objects.all()
    session_years = SessionYearModel.objects.all()
    context = {
        "subjects": subjects,
        "session_years": session_years
    }
    return render(request, "hod_template/admin_view_attendance.html", context)


@csrf_exempt
def admin_get_attendance_dates(request):
    # Getting Values from Ajax POST 'Fetch Student'
    subject_id = request.POST.get("subject")
    session_year = request.POST.get("session_year_id")

    # Students enroll to Course, Course has Subjects
    # Getting all data from subject model based on subject_id
    subject_model = Subjects.objects.get(id=subject_id)
```

```
    session__model = SessionYearModel.objects.get(id=session__year)


    students = Students.objects.filter(course__id=subject__model.course__id,
session__year__id=session__model)

attendance = Attendance.objects.filter(subject__id=subject__model,
session__year__id=session__model)

Only Passing Student Id and Student Name Only

    list__data = ﷺ

for attendance__single in attendance:

        data__small={"id":attendance__single.id,
"attendance__date":str(attendance__single.attendance__date),
"session__year__id":attendance__single.session__year__id.id}

        list__data.append(data__small)


    return JsonResponse(json.dumps(list__data), content__type="application/json", safe=False)



@csrf__exempt

defadmin__get__attendance__student(request):

    Getting Values from Ajax POST 'Fetch Student'

    attendance__date = request.POST.get('attendance__date')

    attendance = Attendance.objects.get(id=attendance__date)
```

```
    attendance__data = AttendanceReport.objects.filter(attendance__id=attendance)
    Only Passing Student Id and Student Name Only
    list__data = ﷺ
for student in attendance__data:
        data__small={"id":student.student__id.admin.id,
"name":student.student__id.admin.first__name+" "+student.student__id.admin.last__name,
"status":student.status}
        list__data.append(data__small)

    return JsonResponse(json.dumps(list__data), content__type="application/json", safe=False)



defadmin__profile(request):
    user = CustomUser.objects.get(id=request.user.id)
context={
        "user": user
    }
    return render(request, 'hod__template/admin__profile.html', context)
def admin__profile__update(request):
    if request.method != "POST":
        messages.error(request, "Invalid Method!")
        return redirect('admin__profile')
    else:
```

```
        first_name = request.POST.get('first_name')
        last_name = request.POST.get('last_name')
        password = request.POST.get('password')
    try:
            customuser = CustomUser.objects.get(id=request.user.id)
            customuser.first_name = first_name
            customuser.last_name = last_name
            ifpassword != None and password != "":
                customuser.set_password(password)
            customuser.save()
            messages.success(request, "Profile Updated Successfully")
            return redirect('admin_profile')
        except:
            messages.error(request, "Failed to Update Profile")
            return redirect('admin_profile')


defstaff_profile(request):
    pass
defstudent_profile(requtest):
    pass
```

٨. TESTING &VALIDATION

١. Unit Testing

Purpose: To verify that individual components or modules of the SMS work correctly in isolation.

Approach:

- Write test cases for each module, such as User Management, Student Enrollment, Attendance Management, etc.
- Use a testing framework like JUnit (for Java) or pytest (for Python) to automate the execution of these test cases.
- Ensure that each function and method within the modules returns the expected results for given inputs.

٢. Integration Testing

Purpose: To ensure that different modules of the SMS interact correctly and work together as expected.

Approach:

- Identify interactions between modules, such as how the Attendance Management Module interacts with the Student Information Management Module.
- Develop test cases that cover these interactions and use integration testing tools to execute them.
- Validate data flow between modules and ensure that integrated components function as a cohesive unit.

3. System Testing

Purpose: To validate the complete and integrated SMS to ensure that it meets the specified requirements.

Approach:

- Conduct end-to-end testing of the entire system.
- Use real-world scenarios to test the system's functionality.
- Ensure that all modules and their interactions are functioning as intended

٤. Performance Testing

Purpose:To ensure that the SMS performs well under expected and peak load conditions.

Approach:

- Conduct load testing to evaluate how the system handles a high number of concurrent users.
- Perform stress testing to determine the system's breaking point and identify any performance bottlenecks.
- Use performance testing tools like JMeter or LoadRunner to automate these tests.

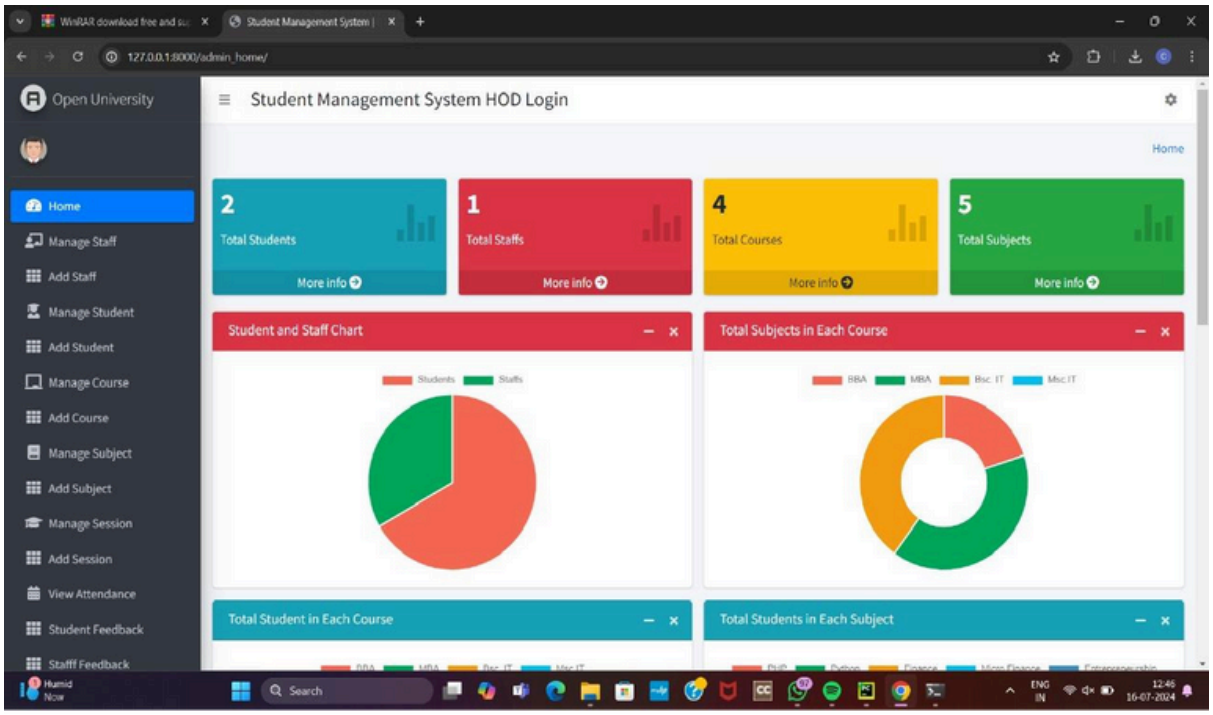
٥. Security Testing

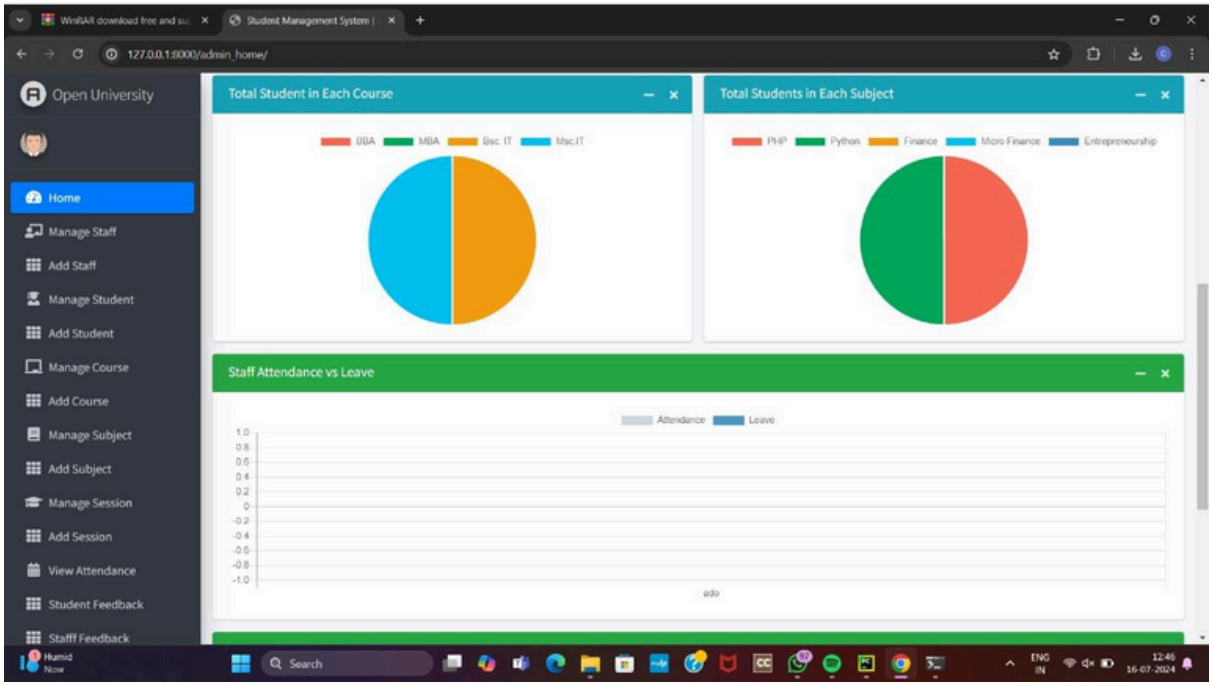
Purpose: To identify and mitigate vulnerabilities in the SMS to protect against threats and ensure data security.

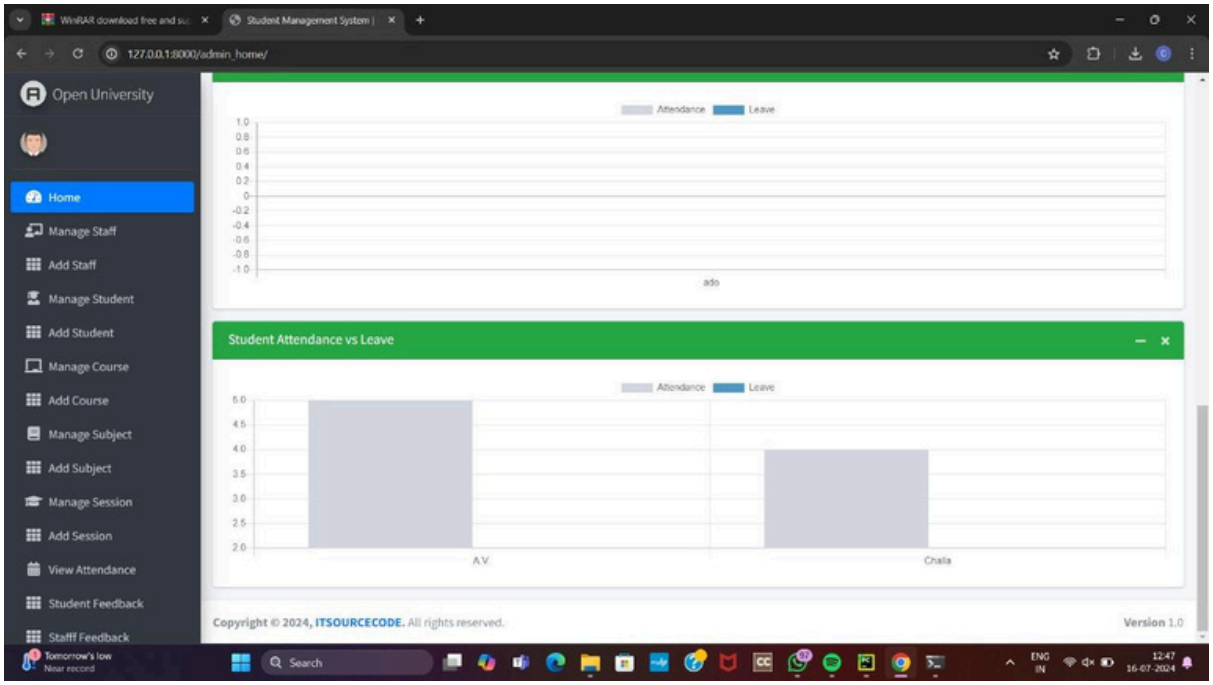
Approach:

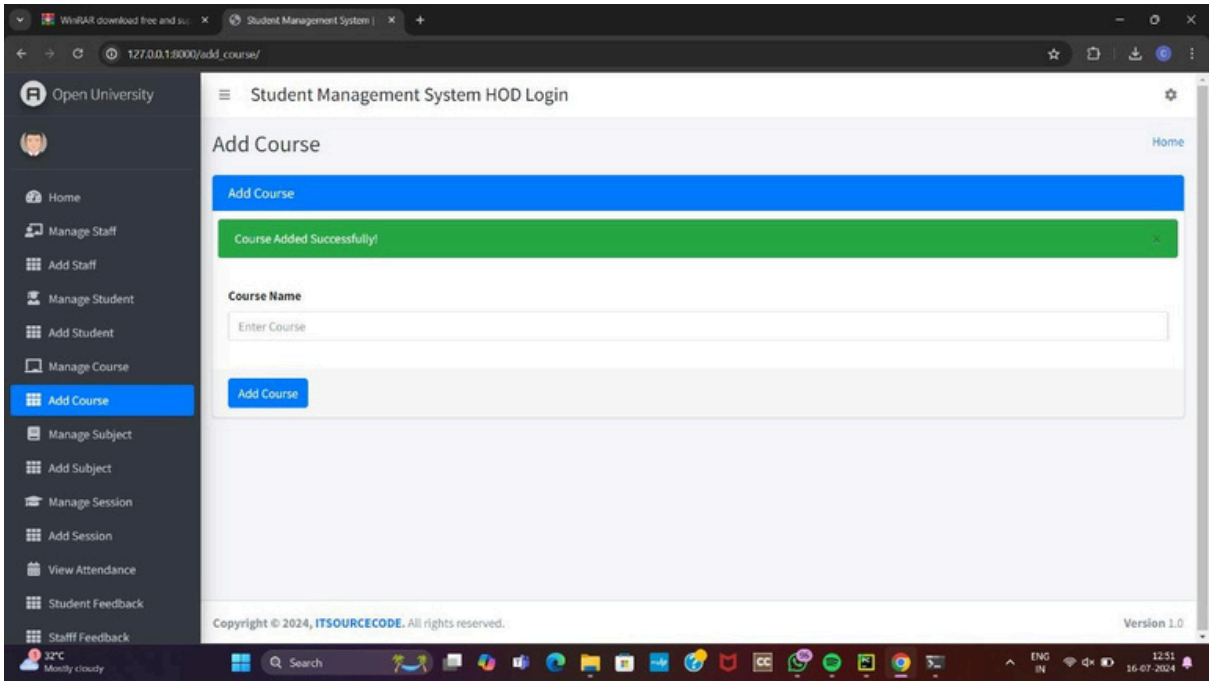
- Conduct vulnerability scanning to identify common security issues such as SQL injection, XSS, and CSRF.
- Perform penetration testing to simulate attacks and identify potential security flaws.
- Ensure compliance with security standards and best practices.

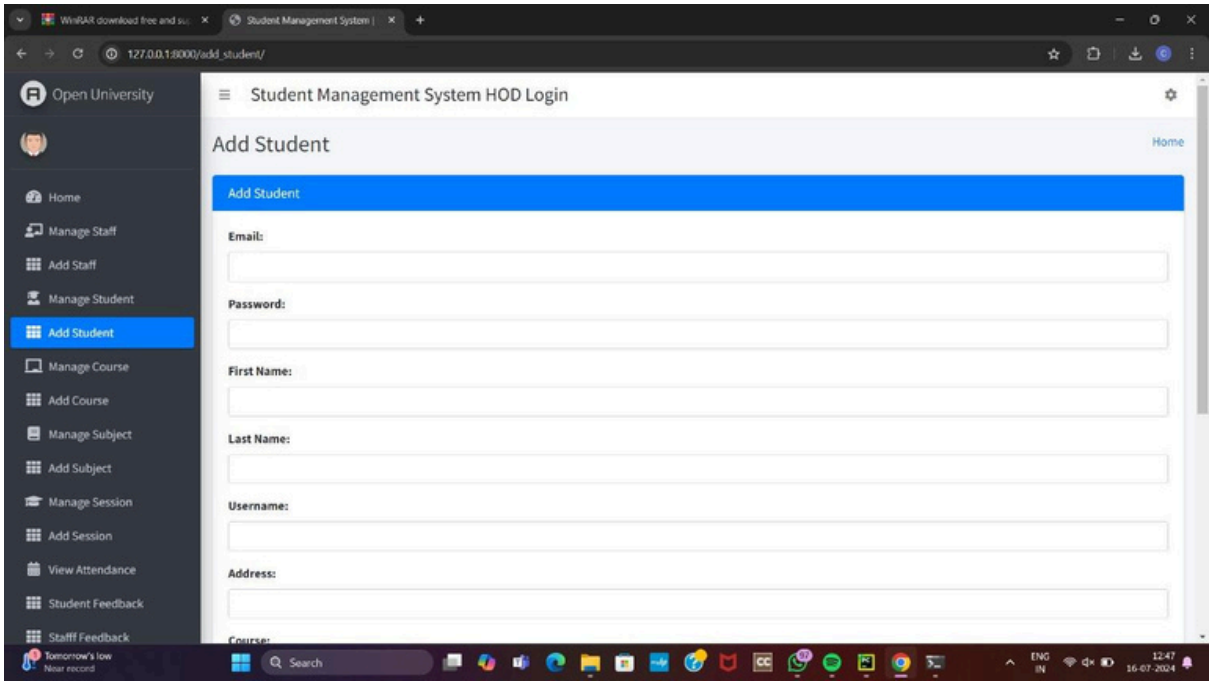
OUTPUTS:

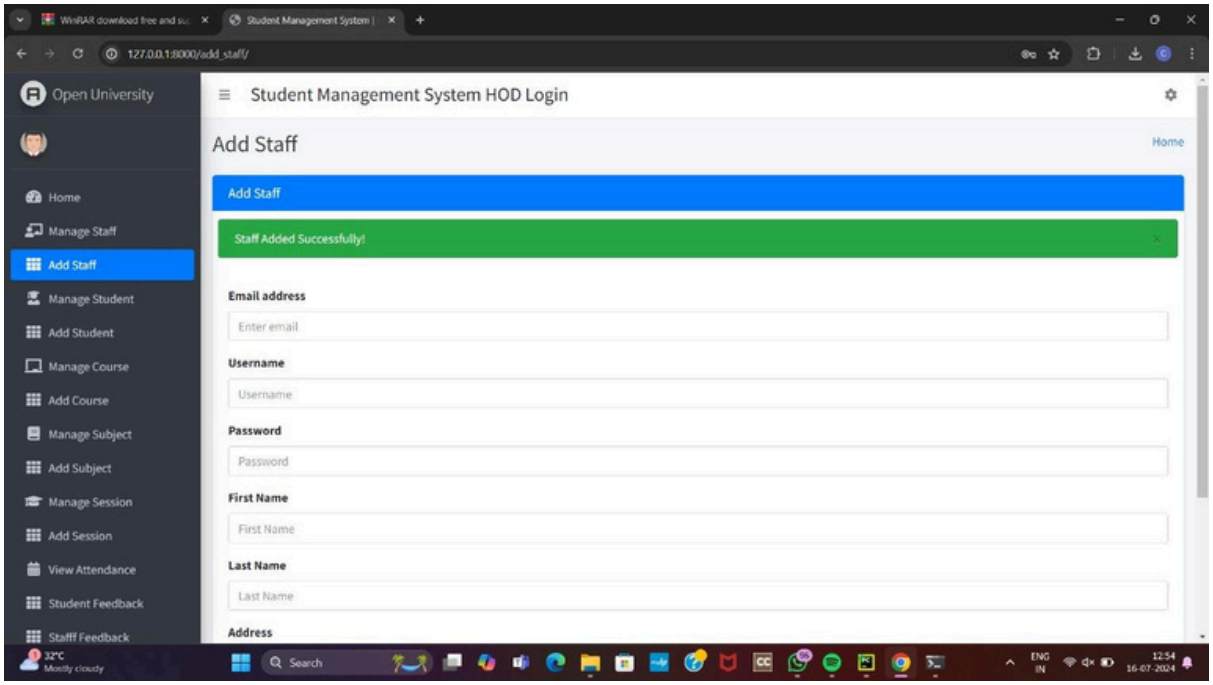

Open University
Student Management System HOD Login
Home
2
Total Students
More info
1
Total Staffs
More info
4
Total Courses
More info
5
Total Subjects
More info
Student and Staff Chart
Students
Staffs
Total Subjects in Each Course
BBA
MBA
Bsc. IT
Msc.IT
Total Student in Each Course
Total Students in Each Subject
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback

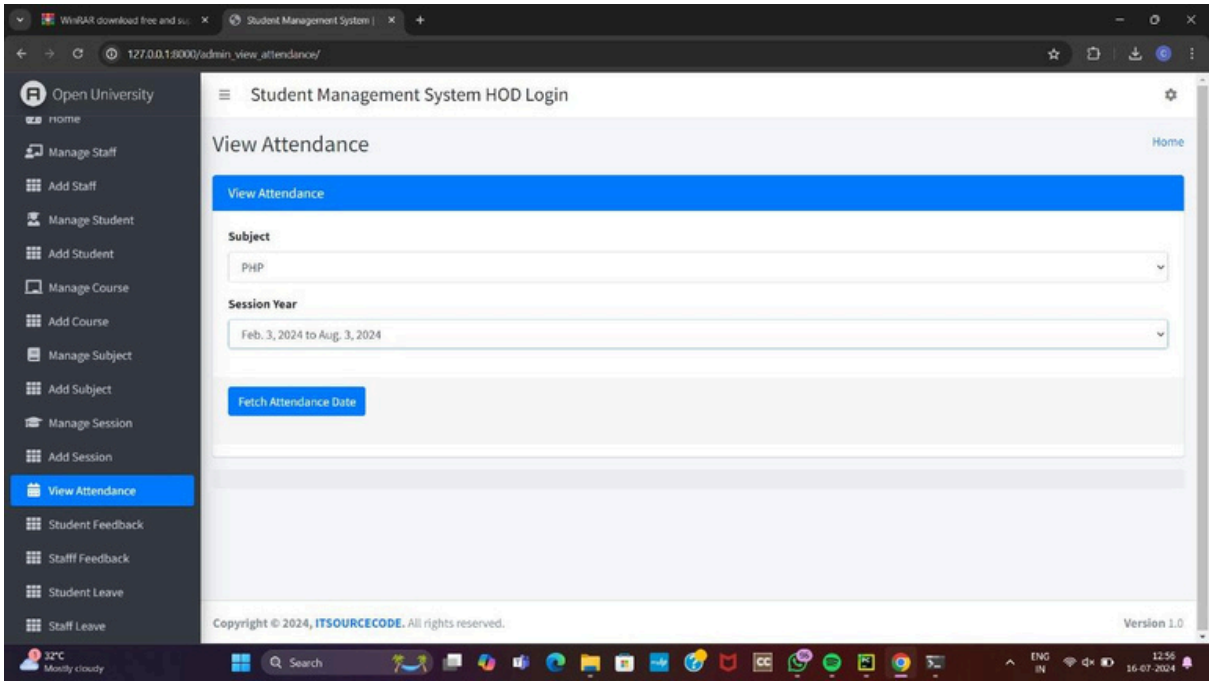

Open University
Total Student in Each Course
BBA
MBA
Bsc. IT
Msc.IT
Total Students in Each Subject
PHP
Python
Finance
Micro Finance
Entrepreneurship
Staff Attendance vs Leave
Attendance
Leave
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback

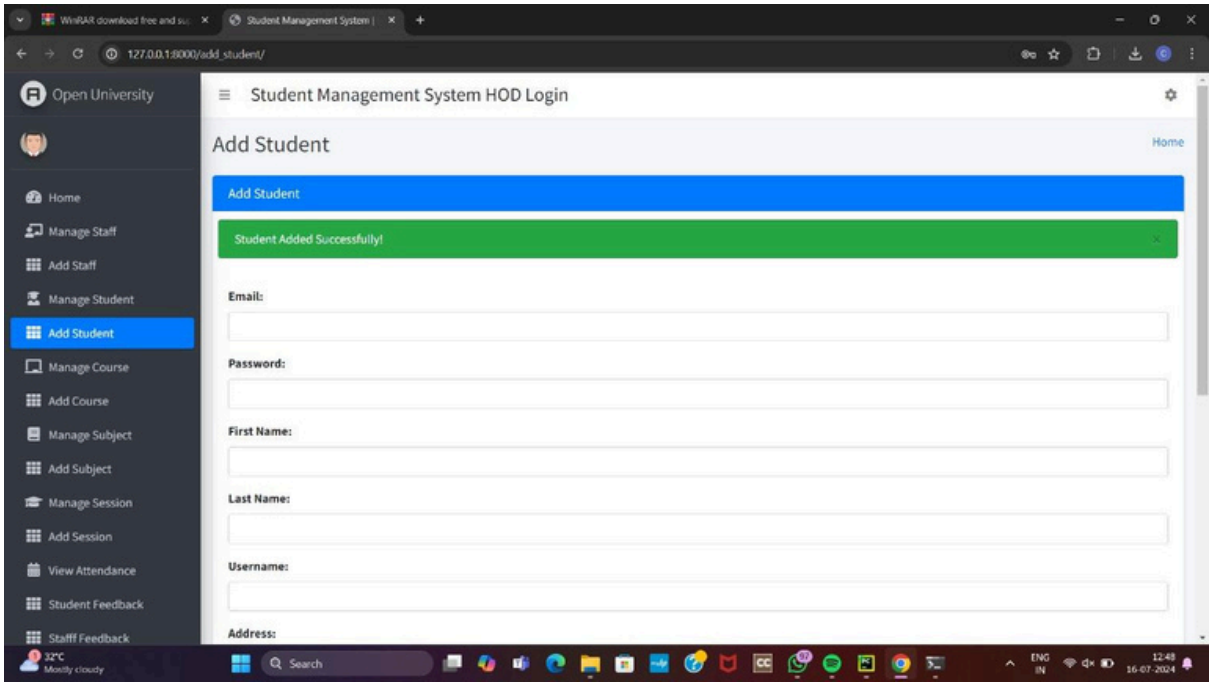

Open University
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback
Attendance
Leave
ado
Student Attendance vs Leave
A.V
Challa
Copyright © 2024, ITSOURCECODE. All rights reserved.
Version 1.0

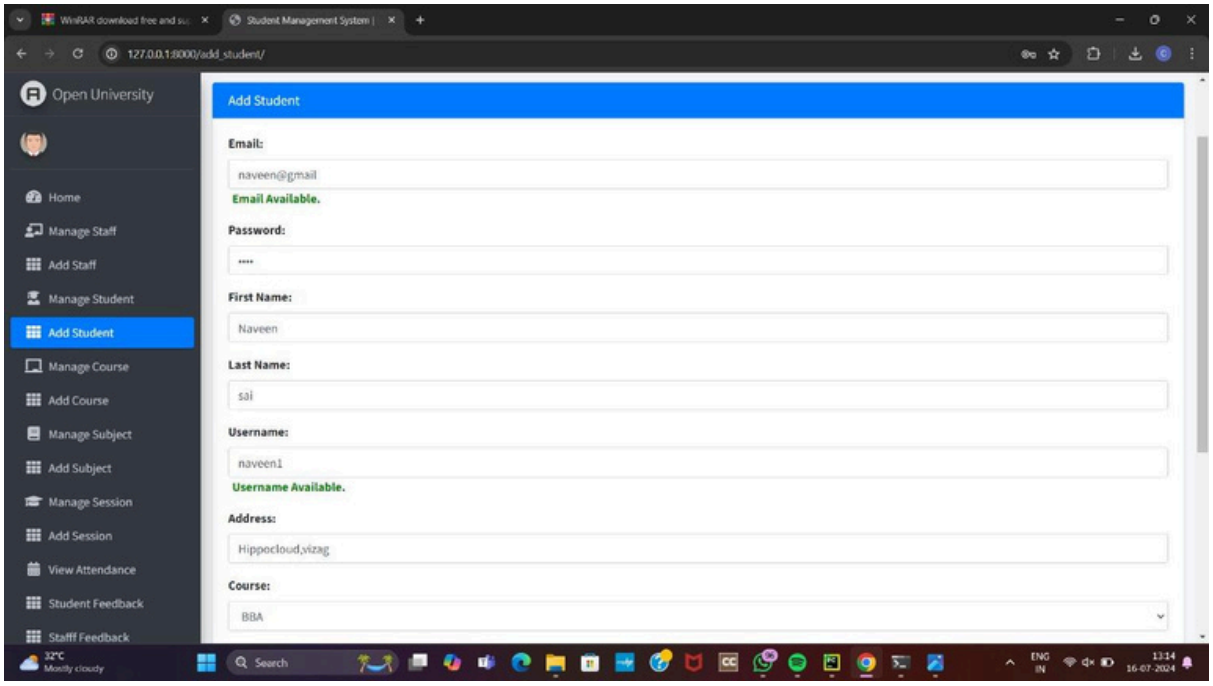

Open University
Student Management System HOD Login
Add Course
Home
Add Course
Course Added Successfully!
Course Name
Enter Course
Add Course
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback
Copyright © 2024, ITSOURCECODE. All rights reserved.
Version 1.0

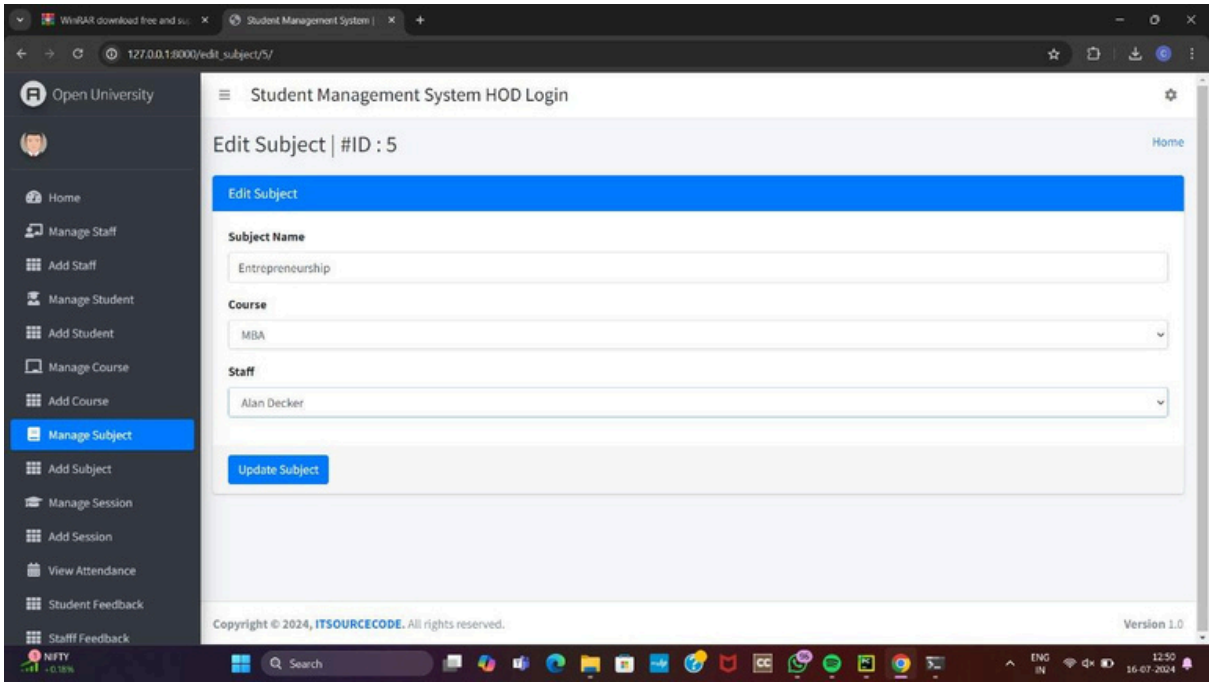

Student Management System
127.0.0.1:8000/add_student/
Open University
Student Management System HOD Login
Add Student
Home
Add Student
Email:
Password:
First Name:
Last Name:
Username:
Address:
Course:
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Staff Feedback
Search


Student Management System
127.0.0.1:8000/add_staff/
Open University
Student Management System HOD Login
Add Staff
Home
Add Staff
Staff Added Successfully!
Email address
Enter email
Username
Username
Password
Password
First Name
First Name
Last Name
Last Name
Address
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Staff Feedback
Search


Open University
Student Management System HOD Login
View Attendance
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback
Student Leave
Staff Leave
View Attendance
Subject
PHP
Session Year
Feb. 3, 2024 to Aug. 3, 2024
Fetch Attendance Date
Copyright © 2024, ITSOURCECODE. All rights reserved.
Version 1.0


Open University
Student Management System HOD Login
Add Student
Home
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback
Add Student
Student Added Successfully!
Email:
Password:
First Name:
Last Name:
Username:
Address:


Open University
Add Student
Email:
naveen@gmail
Email Available.
Password:
Naveen
Last Name:
sai
Username:
naveen1
Username Available.
Address:
Hippocloud,vizag
Course:
BBA
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback


Open University
Student Management System HOD Login
Edit Subject | #ID : 5
Home
Edit Subject
Subject Name
Entrepreneurship
Course
MBA
Staff
Alan Decker
Update Subject
Copyright © 2024, ITSOURCECODE. All rights reserved.
Version 1.0
Home
Manage Staff
Add Staff
Manage Student
Add Student
Manage Course
Add Course
Manage Subject
Add Subject
Manage Session
Add Session
View Attendance
Student Feedback
Stafff Feedback

10.Conclusion:

The development and implementation of a Student Management System (SMS) represent a significant step toward enhancing the efficiency and effectiveness of managing educational institutions. By automating and streamlining various administrative and academic processes, this system provides a robust solution for handling student data, improving communication, and ensuring accurate and timely access to information.

Throughout the project, a structured approach was employed to design and develop key modules, including User Management, Student Enrollment and Admissions, Attendance Management, Grade and Exam Management, and Communication. Each module was carefully crafted to address specific requirements, ensuring that the system is comprehensive and user-friendly.

The system architecture was designed to be scalable, secure, and maintainable, incorporating essential layers such as the Presentation Layer, Application Layer, Data Access Layer, Database Layer, Integration Layer, and Security Layer. This architecture ensures that the system can handle the complex needs of educational institutions while maintaining data integrity and security.

Extensive testing and validation processes were conducted to ensure the system's functionality, performance, security, and usability. Unit testing, integration testing, system testing, performance testing, security testing, and usability testing were all employed to identify and resolve potential issues, ensuring a robust and reliable system.

By providing features such as centralized student information management, efficient attendance tracking, comprehensive grade management, seamless communication, and more, the Student Management System significantly reduces the administrative burden on staff and enhances the overall educational experience for students and parents.

In conclusion, the successful completion of the Student Management System project marks a transformative advancement in educational administration. It not only optimizes operational efficiency but also fosters a more organized, transparent, and responsive educational environment. This system is poised to play a crucial role in the digital transformation of