MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

Mini Project Report

of

Internet Technologies Lab (CSE 3262)

TITLE

Online Learning Management System for Student-Teacher Interaction

**SUBMITTED
BY**

**Soumya Sahu
Reg no: 210905196
Roll no. :35
Section: A**

**Department of Computer Science and Engineering
Manipal Institute of Technology, Manipal.**

April 2024

MANIPAL INSTITUTE OF TECHNOLOGY
MANIPAL
(A constituent unit of MAHE, Manipal)

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Manipal
05/04/2023

CERTIFICATE

This is to certify that the project titled **Online Learning Management System for Student-Teacher Interaction** is a record of the bonafide work done by **Student (Reg. No. 210905195)** submitted in partial fulfilment of the requirements for the award of the Degree of Bachelor of Technology (B.Tech.) in COMPUTER SCIENCE & ENGINEERING of Manipal Institute of Technology, Manipal, Karnataka, (A Constituent Institute of Manipal Academy of Higher Education), during the academic year 2022-2023.

Name and Signature of Examiners:

1. **Dr. Roopalakshmi R, Associate Professor, CSE Dept.**

2. **Prof.   , Assistant Professor, CSE Dept.**

TABLE OF CONTENTS

Chapter 1: Introduction

In today's rapidly evolving digital landscape, where information is readily accessible at our fingertips, the educational sector must adapt to meet the changing needs of students and educators alike. Online Learning Management Systems (LMS) stand at the forefront of this transformation, revolutionizing traditional teaching methods and opening doors to new avenues of learning. By seamlessly integrating technology into the educational process, these systems empower students to take control of their learning journey while providing educators with powerful tools to facilitate effective instruction. This project endeavors to harness the potential of modern technologies, including Django, HTML, CSS, jQuery, and JavaScript, to create an innovative LMS platform tailored specifically to the unique dynamics of university education. With a focus on user-centric design and intuitive functionality, this platform seeks to redefine the educational experience, fostering collaboration, engagement, and achievement in the digital age. Through the synergy of cutting-edge technology and pedagogical expertise, we aim to build a bridge that connects students and teachers in a vibrant ecosystem of knowledge exchange and growth.

Chapter 2: Problem Statement & Objectives

2.1 Problem Statement:

Traditional methods of managing student-teacher interactions, such as manual record-keeping and communication through physical mediums, are outdated and inefficient. There is a need for a centralized platform that streamlines processes such as course enrollment, attendance tracking, grade management, and communication between students and teachers. The absence of such a system leads to disorganization, miscommunication, and potential academic discrepancies. Moreover, with the increasing adoption of online and hybrid learning models, the demand for robust digital solutions to support remote education has never been more pressing. The lack of a comprehensive LMS exacerbates challenges related to accessibility, inclusivity, and accountability in the educational ecosystem. Additionally, without a centralized platform, students and teachers may struggle to effectively collaborate, share resources, and provide timely feedback, hindering the overall quality of education. Addressing these challenges requires the development of an innovative LMS platform that not only meets the basic requirements of course management but also incorporates advanced features to enhance engagement, personalized learning, and academic success.

2.2 Objectives:

- To develop a user-friendly interface for students to easily access course information, manage their academic progress, and communicate with their instructors.
- To create a platform for teachers to efficiently manage their course, track students attendance, record grades, and disseminate announcements.
- To implement secure authentication mechanisms for both students and teachers to ensure data confidentiality and integrity.
- To provide a centralized database for storing student and course information, enabling seamless data retrieval and management.
- To enhance collaboration and communication between students and teachers through features such as announcements and messaging.

Chapter 3: Methodology

1. Requirement Analysis: Conduct surveys and interviews to gather requirements from students and teachers regarding their needs and expectations from the LMS.

2. System Design: Design the architecture of the LMS, including database schema, user interface wireframes, and system functionalities.

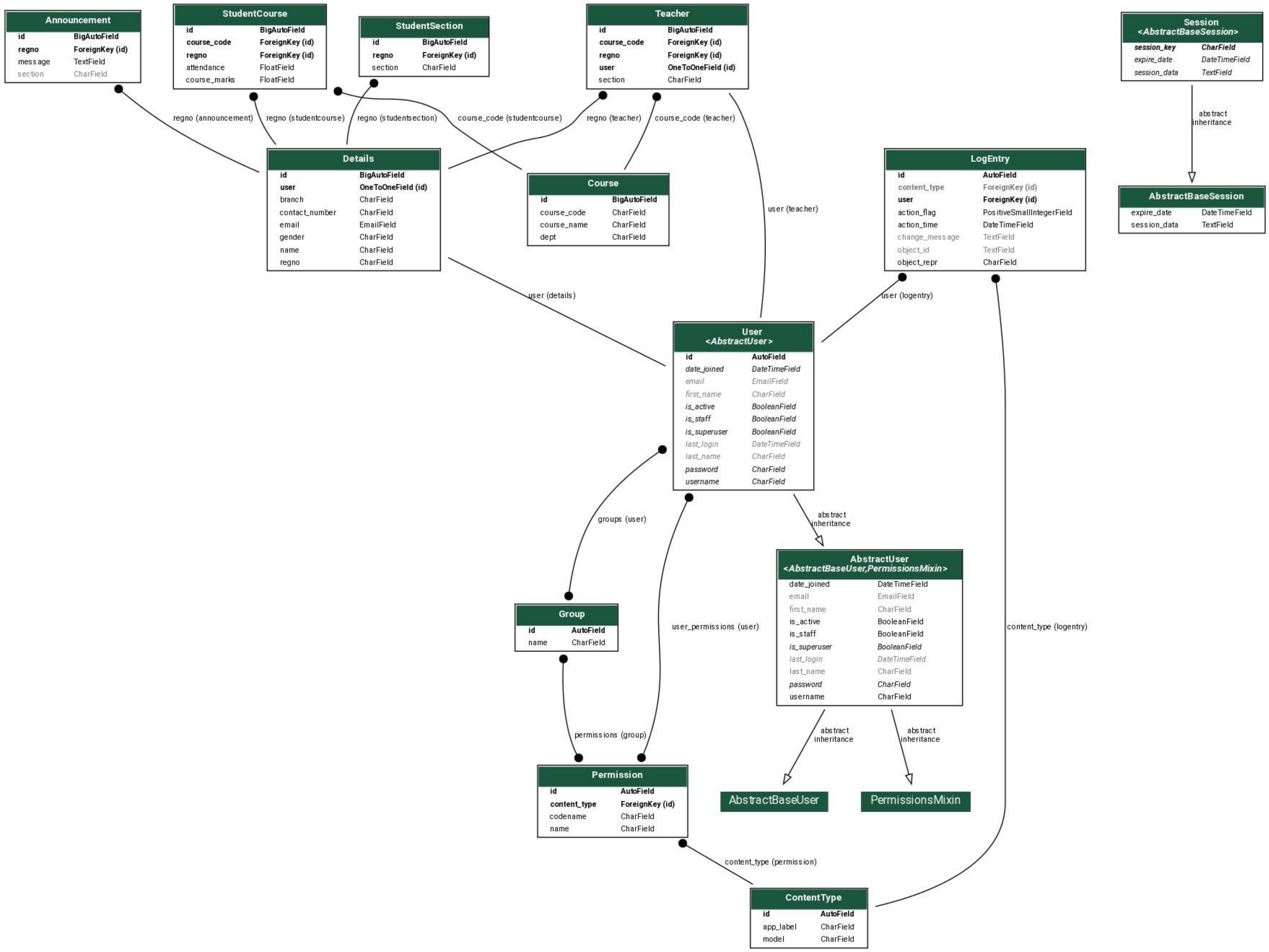
3. Development: Implement the LMS using appropriate technologies and frameworks, ensuring scalability, reliability, and security.

4. Testing: Conduct thorough testing to identify and rectify any bugs or issues in the system, ensuring optimal performance.

5. Deployment: Deploy the LMS on a server, configure access controls, and provide necessary training to users.

6. Evaluation: Solicit feedback from students and teachers regarding their experience with the LMS and make necessary improvements based on their suggestions.

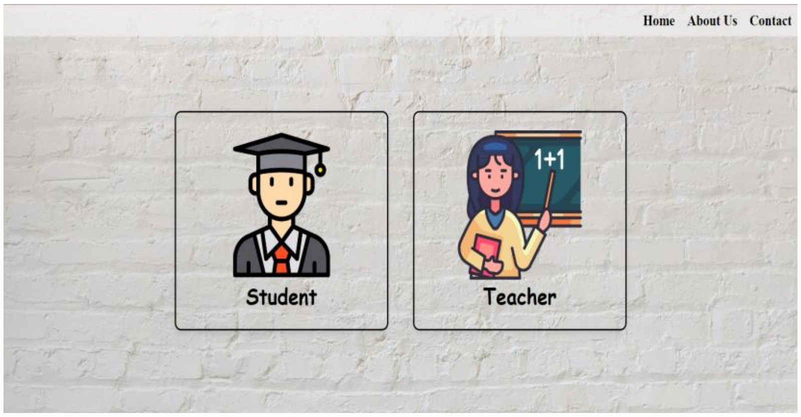
7. By addressing these objectives through a systematic methodology, this project aims to develop an efficient and reliable Online Learning Management System that fosters seamless interaction between students and teachers, ultimately enhancing the overall learning experience.


Announcement
id BigAutoField
regno ForeignKey (id)
message TextField
section CharField
StudentCourse
id BigAutoField
course_code ForeignKey (id)
regno ForeignKey (id)
attendance FloatField
course_marks FloatField
StudentSection
id BigAutoField
regno ForeignKey (id)
section CharField
Teacher
id BigAutoField
course_code ForeignKey (id)
regno ForeignKey (id)
user OneToOneField (id)
section CharField
Session
<AbstractBaseSession>
session_key CharField
expire_date DateTimeField
session_data TextField
abstract
inheritance
AbstractBaseSession
expire_date DateTimeField
session_data TextField
regno (announcement)
regno (studentcourse)
regno (studentsection)
course_code (studentcourse)
regno (teacher)
course_code (teacher)
Details
id BigAutoField
user OneToOneField (id)
branch CharField
contact_number CharField
email EmailField
gender CharField
name CharField
regno CharField
Course
id BigAutoField
course_code CharField
course_name CharField
dept CharField
user (teacher)
LogEntry
id AutoField
content_type ForeignKey (id)
user ForeignKey (id)
action_flag PositiveSmallIntegerField
action_time DateTimeField
change_message TextField
object_id TextField
object_repr CharField
user (details)
user (logentry)
User
<AbstractUser>
id AutoField
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField
groups (user)
abstract
inheritance
Group
id AutoField
name CharField
user_permissions (user)
AbstractUser
<AbstractBaseUser,PermissionsMixin>
date_joined DateTimeField
email EmailField
first_name CharField
is_active BooleanField
is_staff BooleanField
is_superuser BooleanField
last_login DateTimeField
last_name CharField
password CharField
username CharField
content_type (logentry)
permissions (group)
abstract
inheritance
abstract
inheritance
Permission
id AutoField
content_type ForeignKey (id)
codename CharField
name CharField
AbstractBaseUser
PermissionsMixin
content_type (permission)
ContentType
id AutoField
app_label CharField
model CharField

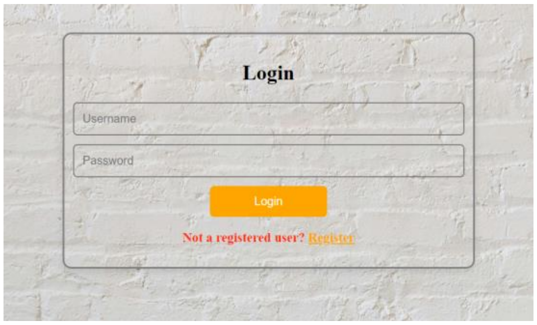
Chapter 4: Results & Snapshots

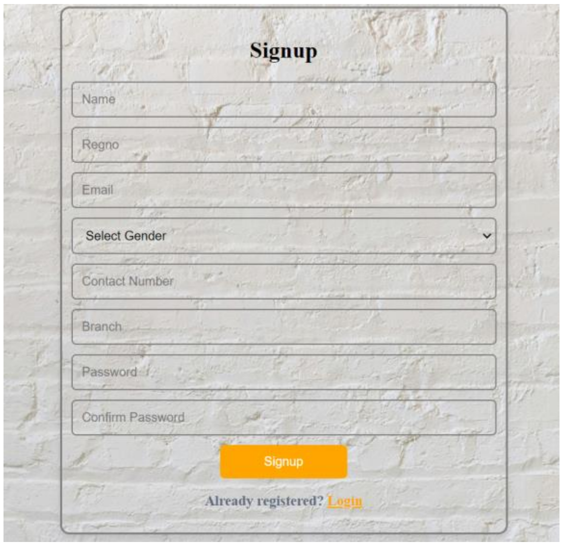
The development of the Online Learning Management System (LMS) has yielded promising results in addressing the identified objectives. The system provides a user-friendly interface for both students and teachers, allowing easy access to course information, management of academic progress, and efficient communication. Teachers can effectively manage courses, track attendance, record grades, and disseminate announcements. The implementation of secure authentication mechanisms ensures data confidentiality and integrity, while the centralized database facilitates seamless data management. Collaboration and communication between students and teachers have been enhanced through features such as announcements and messaging.

First Page:

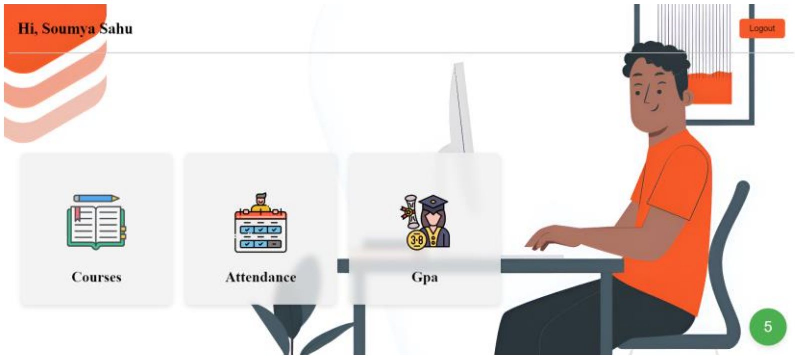


Student login and signup:



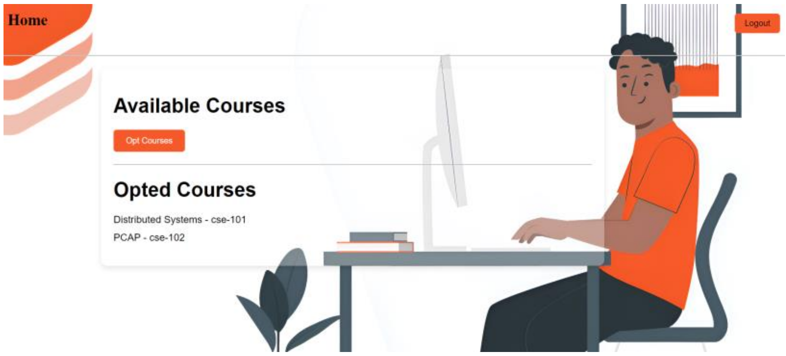


Student Landing page:



Student

1. Course card



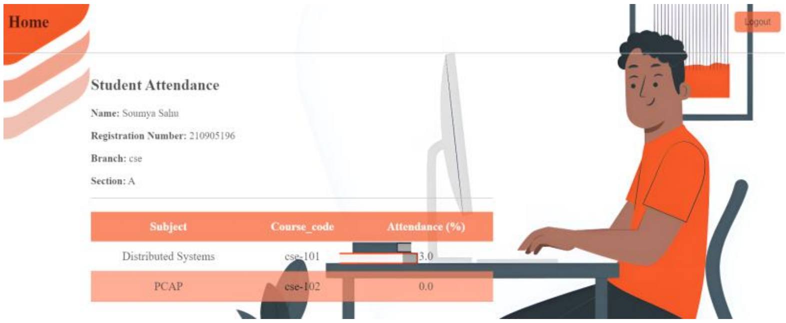
Available Courses

Opted Courses

Distributed Systems - cse-101

PCAP - cse-102

2. Attendance card



Student Attendance

Name: Soumya Sahu

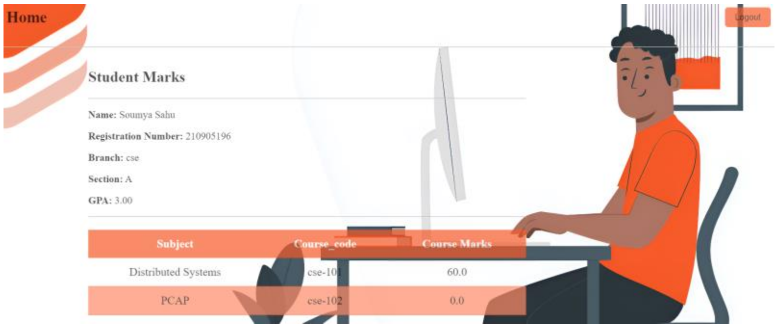
Registration Number: 210905196

Branch: cse

Section: A

Subject	Course_code	Attendance (%)
Distributed Systems	cse-101	3.0
PCAP	cse-102	0.0

3.GPA card



Student Marks

Name: Soumya Sahu

Registration Number: 210905196

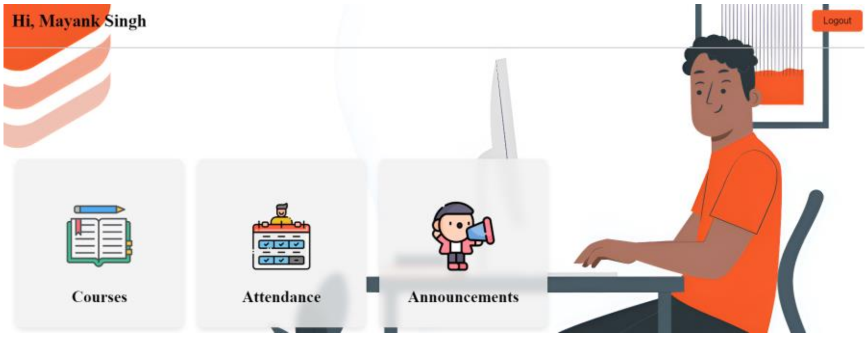
Branch: cse

Section: A

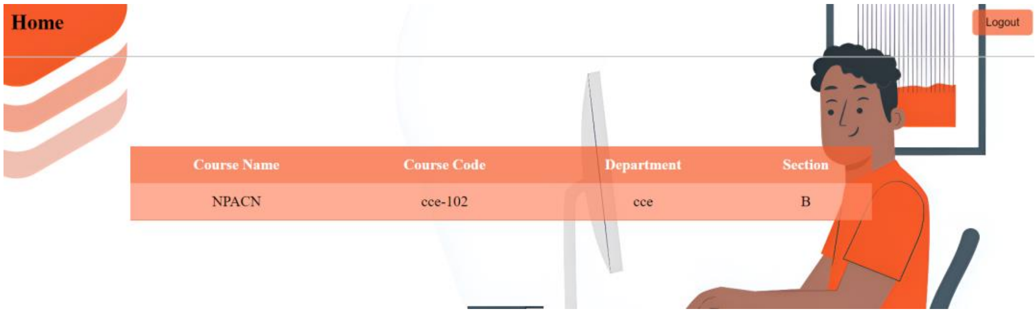
GPA: 3.00

Subject	Course_code	Course Marks
Distributed Systems	cse-101	60.0
PCAP	cse-102	0.0

Teacher Landing page:

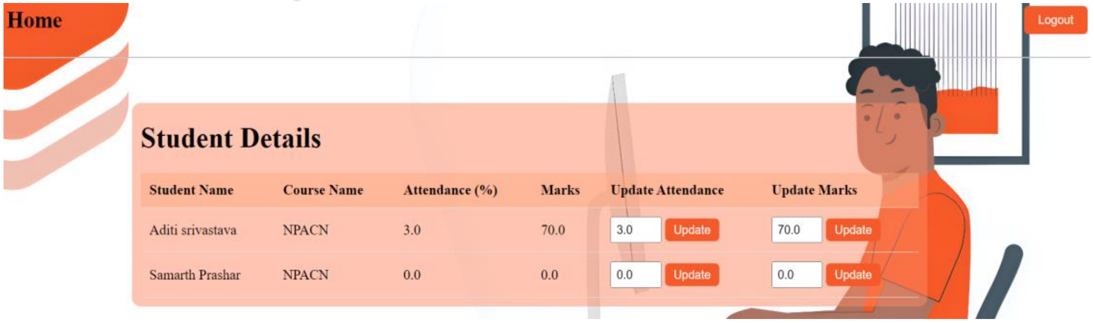


1.Course card:



Course Name	Course Code	Department	Section
NPACN	cce-102	cce	B

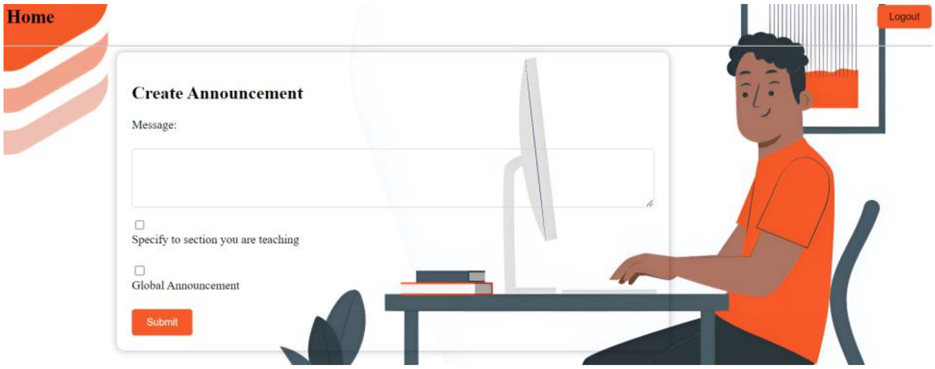
2. Attendance and mark update:



Student Details

Student Name	Course Name	Attendance (%)	Marks	Update Attendance	Update Marks
Aditi srivastava	NPACN	3.0	70.0	3.0 Update	70.0 Update
Samarth Prashar	NPACN	0.0	0.0	0.0 Update	0.0 Update

3. Announcement:



Create Announcement

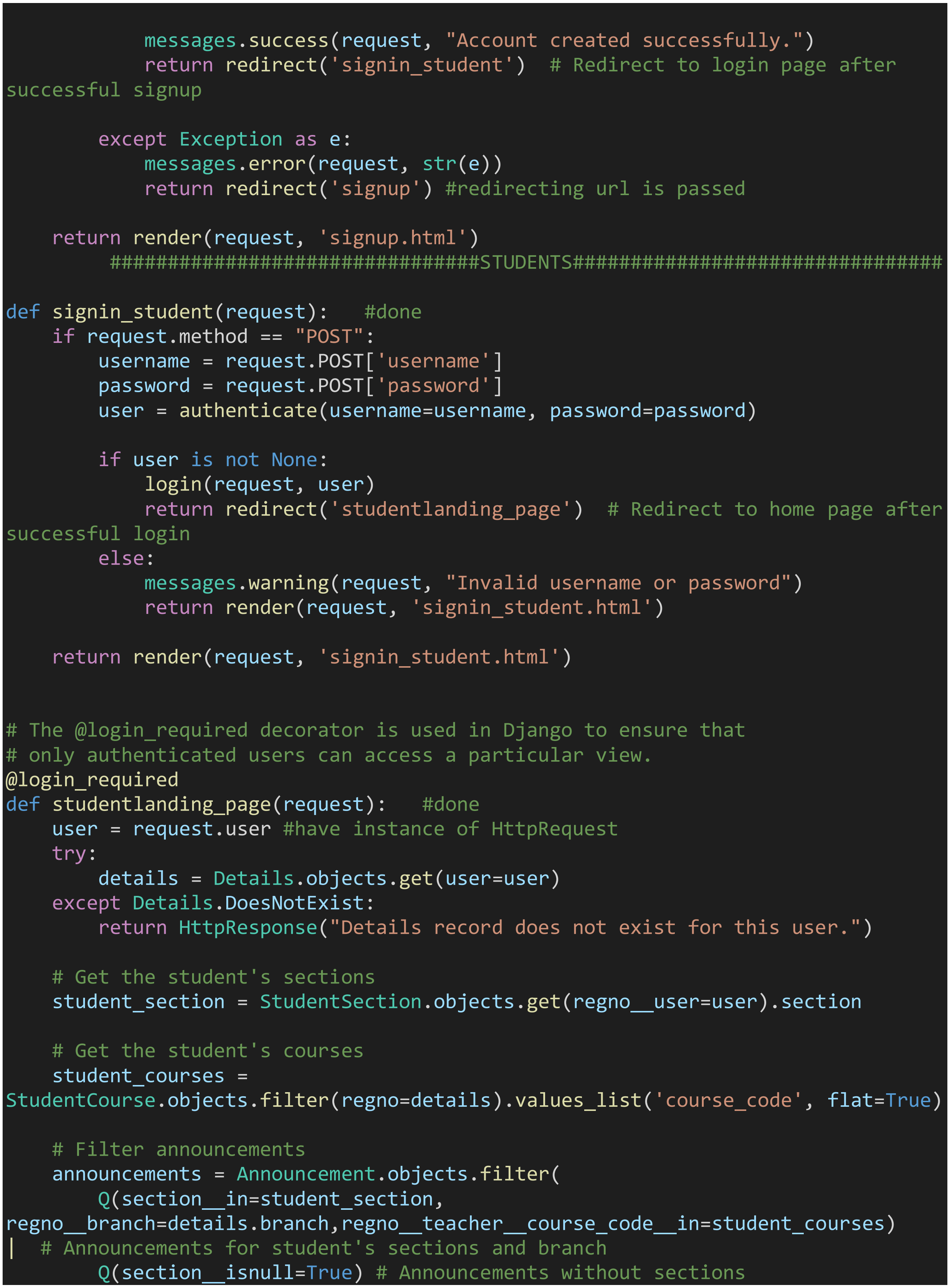
Message:

□ Specify to section you are teaching

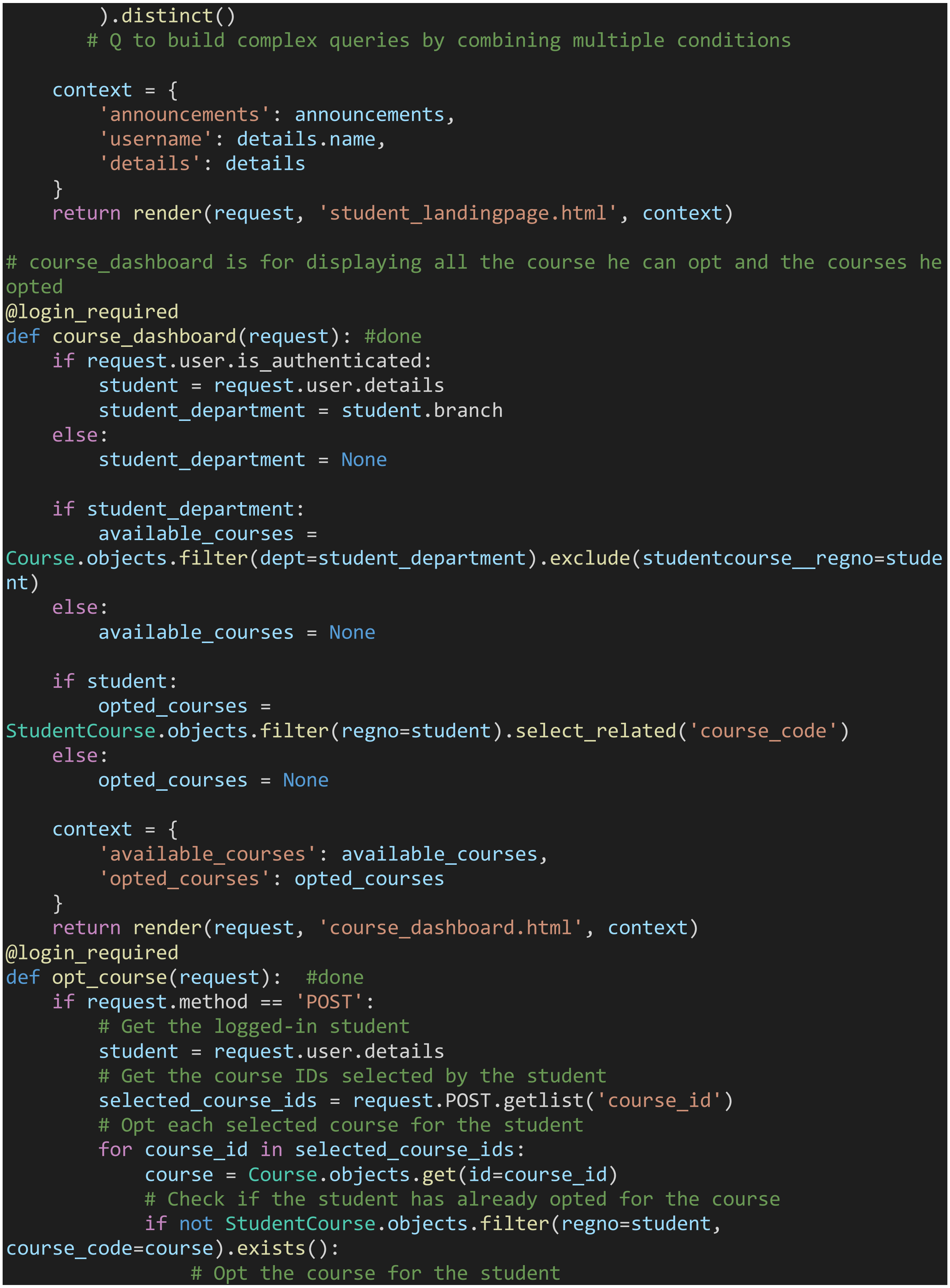
□ Global Announcement

Views.py

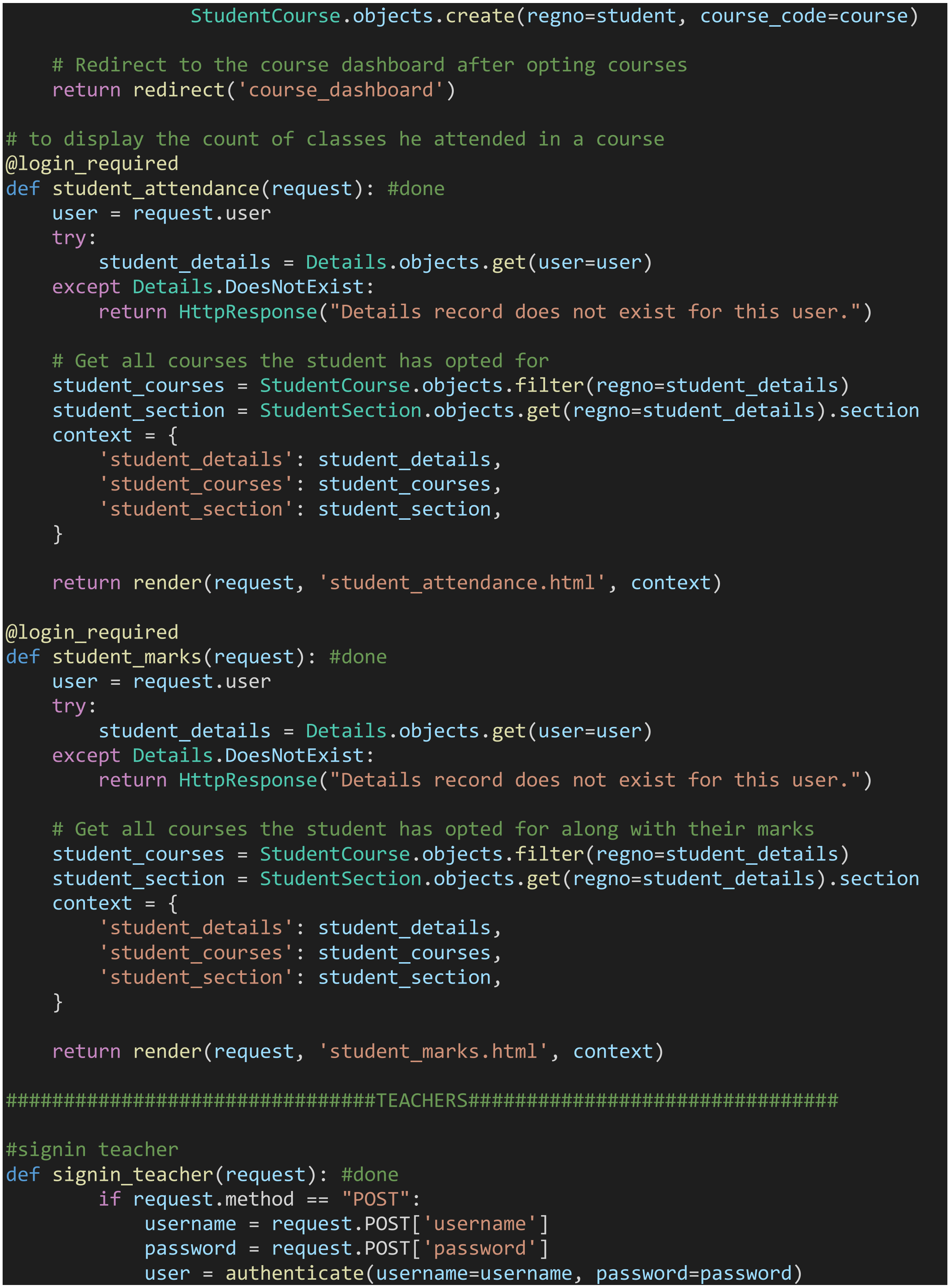
```
from django.shortcuts import render,redirect
from django.contrib.auth import authenticate, login, logout
from django.contrib.auth.models import User
from django.shortcuts import render, redirect
from django.contrib import messages
from django.http import HttpResponse
from django.contrib.auth.decorators import login_required #to ensure that only
authenticated users can access certain views
from django.core.exceptions import ObjectDoesNotExist
from django.db import connection  # for default database connection in Django
application.
from django.db.models import Q
from .models import Details,Course, StudentCourse,Teacher, StudentSection,
Announcement

def index(request):
    return render(request,'index.html')

def signup(request):   #done
    if request.method == "POST":
        name = request.POST['name']
        regno = request.POST['regno']
        email = request.POST['email']
        gender = request.POST['gender']
        contact_number = request.POST['contact_number']
        branch = request.POST['branch']
        password = request.POST['password']
        confirm_password = request.POST['confirm_password']

        # Check if passwords match
        if password != confirm_password:
            messages.warning(request, "Passwords do not match.")
            return redirect('signup')  # Redirect to signup page if passwords
don't match

        # Check if the username (regno) already exists in the database
        if User.objects.filter(username=regno).exists():
            messages.warning(request, "Username already exists.")
            return redirect('signup')  # Redirect to signup page if username
exists

        try:
            # Create the user with username as email and password
            user = User.objects.create_user(username=regno, email=email,
password=password)

            # Create the associated details record
            details = Details.objects.create(user=user, regno=regno, name=name,
email=email, gender=gender,
                                             contact_number=contact_number,
branch=branch)
```



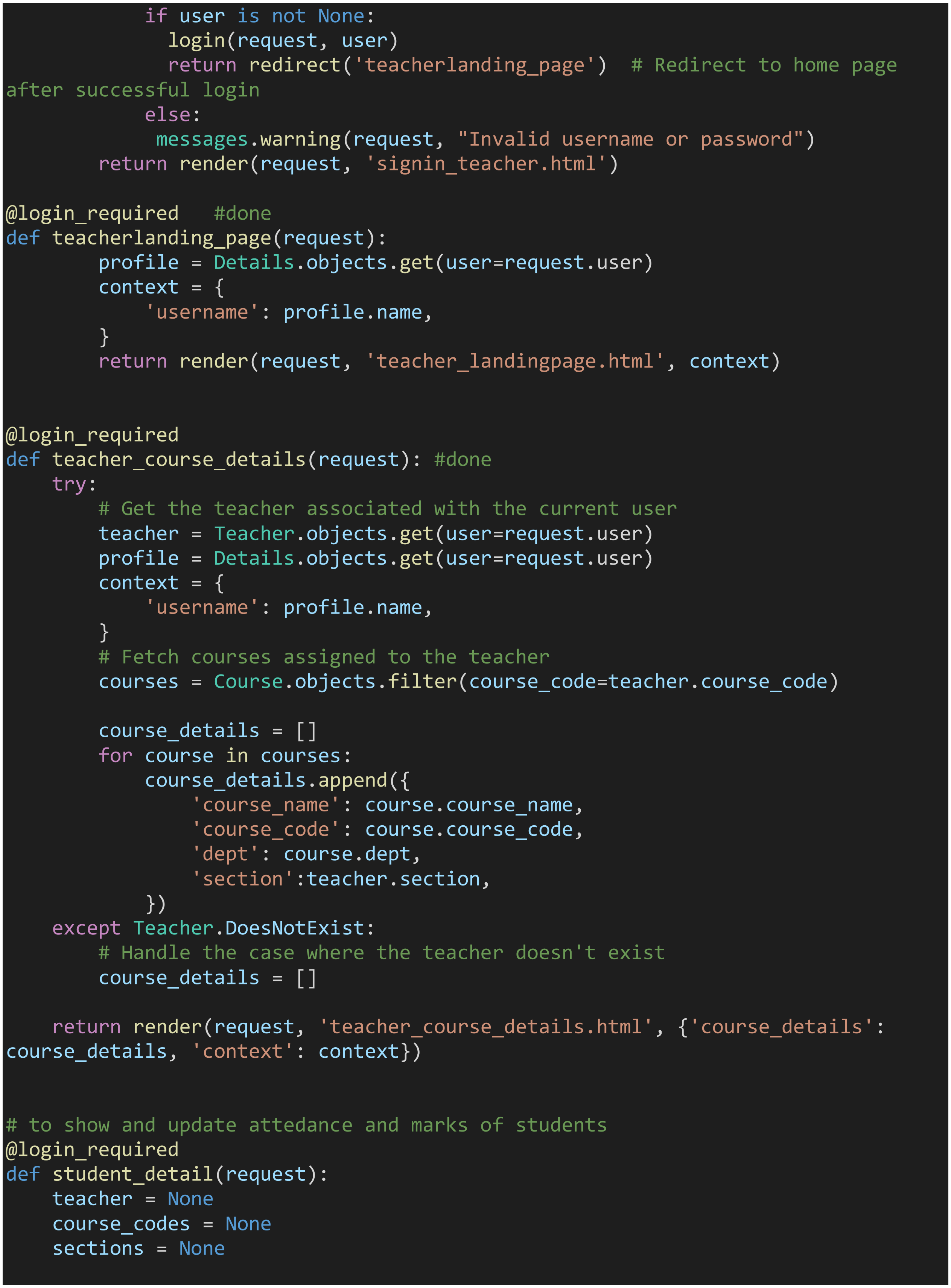
```
            messages.success(request, "Account created successfully.")
            return redirect('signin_student')  # Redirect to login page after
successful signup

        except Exception as e:
            messages.error(request, str(e))
            return redirect('signup') #redirecting url is passed

    return render(request, 'signup.html')
         ################################STUDENTS################################

def signin_student(request):   #done
    if request.method == "POST":
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('studentlanding_page')  # Redirect to home page after
successful login
        else:
            messages.warning(request, "Invalid username or password")
            return render(request, 'signin_student.html')

    return render(request, 'signin_student.html')


# The @login_required decorator is used in Django to ensure that
# only authenticated users can access a particular view.
@login_required
def studentlanding_page(request):   #done
    user = request.user #have instance of HttpRequest
    try:
        details = Details.objects.get(user=user)
    except Details.DoesNotExist:
        return HttpResponse("Details record does not exist for this user.")

    # Get the student's sections
    student_section = StudentSection.objects.get(regno__user=user).section

    # Get the student's courses
    student_courses =
StudentCourse.objects.filter(regno=details).values_list('course_code', flat=True)

    # Filter announcements
    announcements = Announcement.objects.filter(
        Q(section__in=student_section,
regno__branch=details.branch,regno__teacher__course_code__in=student_courses)
|  # Announcements for student's sections and branch
        Q(section__isnull=True) # Announcements without sections
```



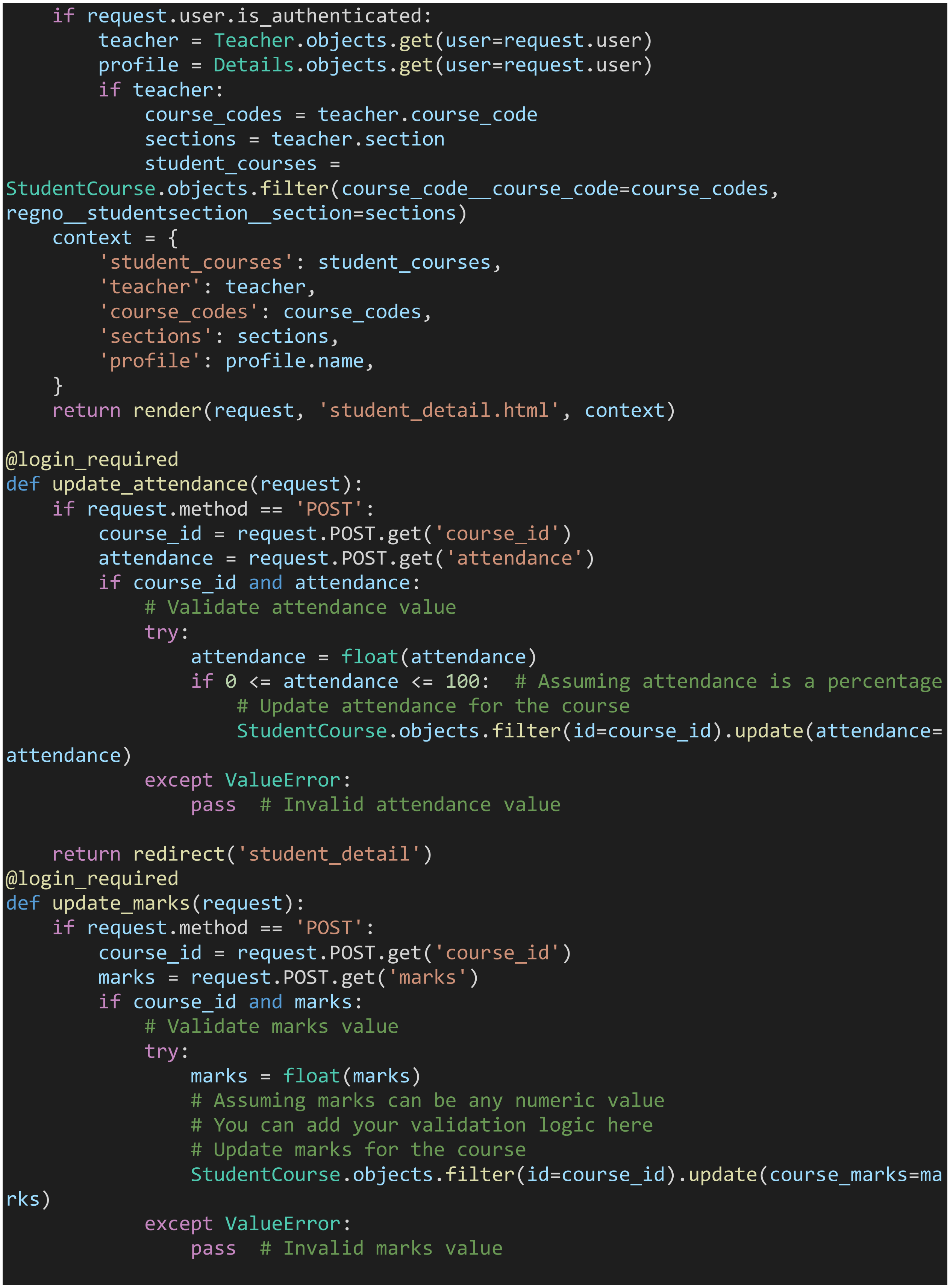
```
        ).distinct()
       # Q to build complex queries by combining multiple conditions

    context = {
        'announcements': announcements,
        'username': details.name,
        'details': details
    }
    return render(request, 'student_landingpage.html', context)

# course_dashboard is for displaying all the course he can opt and the courses he
opted
@login_required
def course_dashboard(request): #done
    if request.user.is_authenticated:
        student = request.user.details
        student_department = student.branch
    else:
        student_department = None

    if student_department:
        available_courses =
Course.objects.filter(dept=student_department).exclude(studentcourse__regno=stude
nt)
    else:
        available_courses = None

    if student:
        opted_courses =
StudentCourse.objects.filter(regno=student).select_related('course_code')
    else:
        opted_courses = None

    context = {
        'available_courses': available_courses,
        'opted_courses': opted_courses
    }
    return render(request, 'course_dashboard.html', context)
@login_required
def opt_course(request):  #done
    if request.method == 'POST':
        # Get the logged-in student
        student = request.user.details
        # Get the course IDs selected by the student
        selected_course_ids = request.POST.getlist('course_id')
        # Opt each selected course for the student
        for course_id in selected_course_ids:
            course = Course.objects.get(id=course_id)
            # Check if the student has already opted for the course
            if not StudentCourse.objects.filter(regno=student,
course_code=course).exists():
                # Opt the course for the student
```



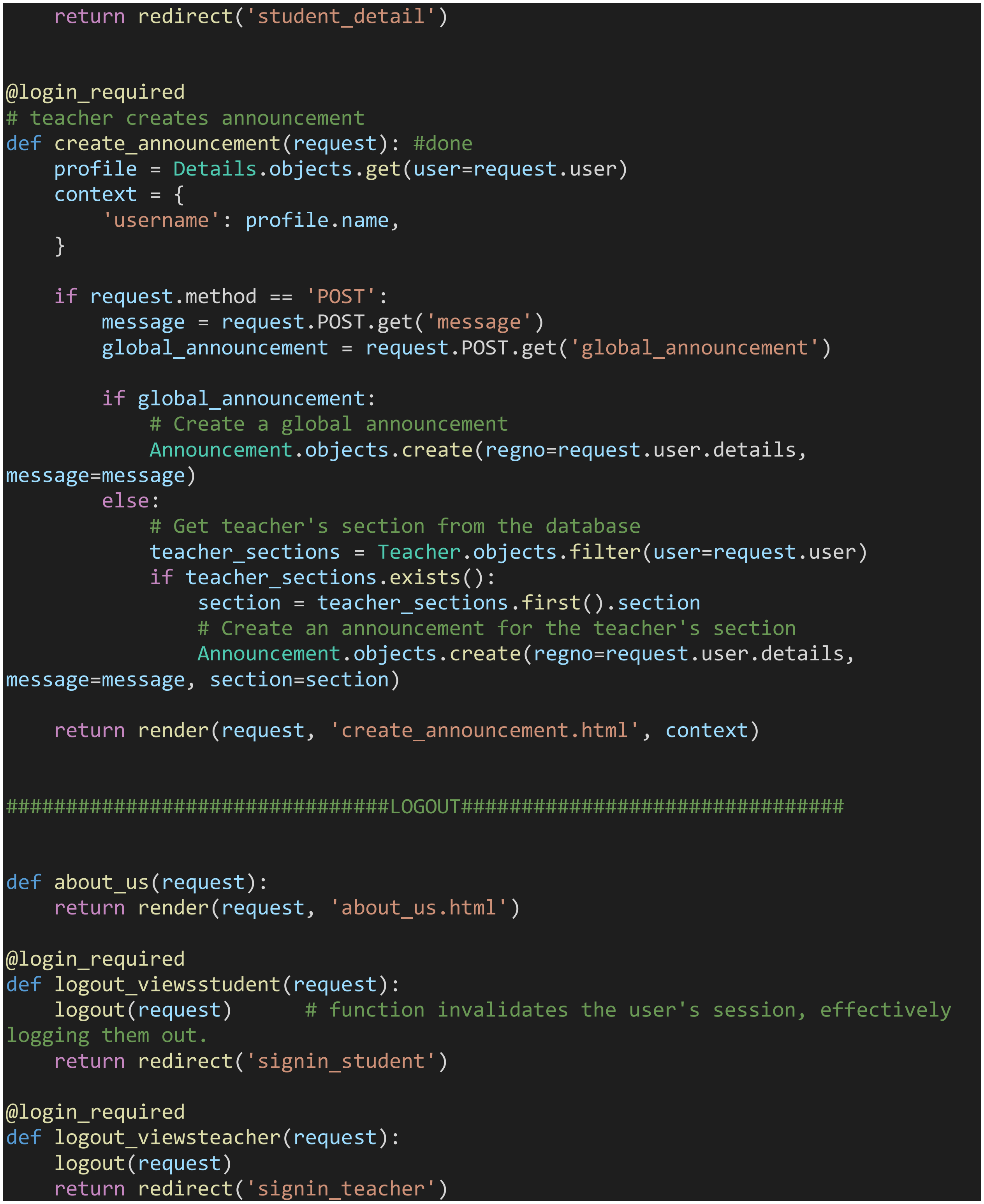
```
                StudentCourse.objects.create(regno=student, course_code=course)

    # Redirect to the course dashboard after opting courses
    return redirect('course_dashboard')

# to display the count of classes he attended in a course
@login_required
def student_attendance(request): #done
    user = request.user
    try:
        student_details = Details.objects.get(user=user)
    except Details.DoesNotExist:
        return HttpResponse("Details record does not exist for this user.")

    # Get all courses the student has opted for
    student_courses = StudentCourse.objects.filter(regno=student_details)
    student_section = StudentSection.objects.get(regno=student_details).section
    context = {
        'student_details': student_details,
        'student_courses': student_courses,
        'student_section': student_section,
    }

    return render(request, 'student_attendance.html', context)

@login_required
def student_marks(request): #done
    user = request.user
    try:
        student_details = Details.objects.get(user=user)
    except Details.DoesNotExist:
        return HttpResponse("Details record does not exist for this user.")

    # Get all courses the student has opted for along with their marks
    student_courses = StudentCourse.objects.filter(regno=student_details)
    student_section = StudentSection.objects.get(regno=student_details).section
    context = {
        'student_details': student_details,
        'student_courses': student_courses,
        'student_section': student_section,
    }

    return render(request, 'student_marks.html', context)

################################TEACHERS################################

#signin teacher
def signin_teacher(request): #done
        if request.method == "POST":
            username = request.POST['username']
            password = request.POST['password']
            user = authenticate(username=username, password=password)
```



```
            if user is not None:
              login(request, user)
              return redirect('teacherlanding_page')  # Redirect to home page
after successful login
            else:
             messages.warning(request, "Invalid username or password")
        return render(request, 'signin_teacher.html')

@login_required   #done
def teacherlanding_page(request):
        profile = Details.objects.get(user=request.user)
        context = {
            'username': profile.name,
        }
        return render(request, 'teacher_landingpage.html', context)


@login_required
def teacher_course_details(request): #done
    try:
        # Get the teacher associated with the current user
        teacher = Teacher.objects.get(user=request.user)
        profile = Details.objects.get(user=request.user)
        context = {
            'username': profile.name,
        }
        # Fetch courses assigned to the teacher
        courses = Course.objects.filter(course_code=teacher.course_code)

        course_details = []
        for course in courses:
            course_details.append({
                'course_name': course.course_name,
                'course_code': course.course_code,
                'dept': course.dept,
                'section':teacher.section,
            })
    except Teacher.DoesNotExist:
        # Handle the case where the teacher doesn't exist
        course_details = []

    return render(request, 'teacher_course_details.html', {'course_details':
course_details, 'context': context})


# to show and update attedance and marks of students
@login_required
def student_detail(request):
    teacher = None
    course_codes = None
    sections = None
```



```
    if request.user.is_authenticated:
        teacher = Teacher.objects.get(user=request.user)
        profile = Details.objects.get(user=request.user)
        if teacher:
            course_codes = teacher.course_code
            sections = teacher.section
            student_courses =
StudentCourse.objects.filter(course_code__course_code=course_codes,
regno__studentsection__section=sections)
    context = {
        'student_courses': student_courses,
        'teacher': teacher,
        'course_codes': course_codes,
        'sections': sections,
        'profile': profile.name,
    }
    return render(request, 'student_detail.html', context)

@login_required
def update_attendance(request):
    if request.method == 'POST':
        course_id = request.POST.get('course_id')
        attendance = request.POST.get('attendance')
        if course_id and attendance:
            # Validate attendance value
            try:
                attendance = float(attendance)
                if 0 <= attendance <= 100:  # Assuming attendance is a percentage
                    # Update attendance for the course
                    StudentCourse.objects.filter(id=course_id).update(attendance=
attendance)
            except ValueError:
                pass  # Invalid attendance value

    return redirect('student_detail')
@login_required
def update_marks(request):
    if request.method == 'POST':
        course_id = request.POST.get('course_id')
        marks = request.POST.get('marks')
        if course_id and marks:
            # Validate marks value
            try:
                marks = float(marks)
                # Assuming marks can be any numeric value
                # You can add your validation logic here
                # Update marks for the course
                StudentCourse.objects.filter(id=course_id).update(course_marks=ma
rks)
            except ValueError:
                pass  # Invalid marks value
```



```
    return redirect('student_detail')


@login_required
# teacher creates announcement
def create_announcement(request): #done
    profile = Details.objects.get(user=request.user)
    context = {
        'username': profile.name,
    }

    if request.method == 'POST':
        message = request.POST.get('message')
        global_announcement = request.POST.get('global_announcement')

        if global_announcement:
            # Create a global announcement
            Announcement.objects.create(regno=request.user.details,
message=message)
        else:
            # Get teacher's section from the database
            teacher_sections = Teacher.objects.filter(user=request.user)
            if teacher_sections.exists():
                section = teacher_sections.first().section
                # Create an announcement for the teacher's section
                Announcement.objects.create(regno=request.user.details,
message=message, section=section)

    return render(request, 'create_announcement.html', context)


################################LOGOUT################################


def about_us(request):
    return render(request, 'about_us.html')

@login_required
def logout_viewsstudent(request):
    logout(request)      # function invalidates the user's session, effectively
logging them out.
    return redirect('signin_student')

@login_required
def logout_viewsteacher(request):
    logout(request)
    return redirect('signin_teacher')
```

Chapter 5: Conclusion

In conclusion, the design and development of the Online Learning Management System represents a significant step towards modernizing educational practices. By leveraging technology to create an intuitive platform tailored to the needs of university education, we have addressed key challenges in student-teacher interactions and course management. The system's user-centric design and comprehensive functionalities have the potential to improve engagement, collaboration, and academic success in the digital age. Moving forward, it is imperative to continue refining the system based on user feedback and technological advancements to ensure its relevance and effectiveness in the ever-evolving educational landscape.

Chapter 6: Limitations & Future works

- **Limitations:**

1. **Technical Constraints**: The system may encounter technical issues or limitations related to scalability, performance, and compatibility with different devices and browsers.
2. **User Adoption:** Resistance to change or unfamiliarity with technology among users could hinder the widespread adoption and utilization of the system.]
3. **Resource Constraints:** Limited resources such as time, budget, and expertise may impact the development, maintenance, and support of the system.
4. **User Adoption:** Ensuring accessibility for users with disabilities and diverse learning needs may require additional efforts and resources.

- **Future works**:

1. **Enhanced Features:** Continuously adding new features and functionalities to improve the user experience and meet evolving educational needs.
2. **Integration:** Integrating the LMS with other educational tools and systems to enhance interoperability and streamline workflows.
3. **Personalization:** Implementing personalized learning pathways and adaptive technologies to cater to individual student needs and preferences.
4. **Analytics:** Incorporating data analytics and machine learning techniques to gain insights into student performance, engagement, and learning outcomes.
5. **Mobile Support:** Developing dedicated mobile applications or ensuring full compatibility with mobile devices to enhance accessibility and convenience for users on the go.

Chapter 7: References

1. https://www.w3schools.com/django/
2. https://stackoverflow.com/
3. https://youtu.be/rHux0gMZ3Eg?si=szKCVR_deDqIWAm8
4. https://www.geeksforgeeks.org/django-tutorial/