SRI VENKATESHWARA UNIVERSITY-TIRUPATI

## **A MAJOR PROJECT REPORT**

ON

Advanced Online Learning Platform for Personalized Education And Enhanced Student Engagement

Submitted in partial fulfillment of the requirements for the IV Semester of

**MASTER OF COMPUTER APPLICATIONS**



Submitted By

K. ROHIT KUMAR 2723188

Under the valuable Guidance of

**Mr. K. CHANDRA SEKHAR, M.Sc., M.Phil. (PhD)**

**Associate Professor**

****

## **MOTHER THERESA INSTITUTE OF COMPUTER APPLICATIONS**

(Affiliated to S. V. University, Tirupati) **PALAMANER-517408.CHITTOOR**

**Abstract**

This project presents **An Online Learning Platform with Intelligent Tutoring**, developed using Django, designed to provide a comprehensive and interactive learning experience for users. The platform facilitates user registration, where each user is assigned a unique account number generated based on their chosen username and password.

Upon successful login, users are directed to their personalized homepage, which showcases an array of courses available for enrolment. Users can browse these courses, add them to their cart, and proceed to purchase video content associated with the selected courses. The platform supports a seamless video playback feature for purchased courses and provides a detailed payment history for users to track their transactions.

In addition, an integrated Chabot enhances user interaction by providing instant responses to inquiries, such as course pricing. Users are encouraged to send feedback regarding their learning experience, which can be directly addressed by administrators.

The **Admin Portal** offers robust management capabilities, allowing administrators to view and manage user details, including options to edit or delete accounts. Admins can also create and manage courses, including adding new courses, editing existing ones, or removing courses as necessary. Furthermore, administrators can create assignments for users and respond to user feedback to foster an improved learning environment.

This platform leverages AI components to enhance the tutoring experience, ensuring that users receive tailored guidance and support throughout their educational journey. By integrating features such as real-time feedback mechanisms, a comprehensive course catalog, and intelligent chat support, this online learning platform stands out as a versatile tool for both learners and educators.

**Introduction**

* In today's fast-paced digital world, online education has become an essential medium for knowledge acquisition and skill development. With the proliferation of learning resources available on the internet, learners require a platform that not only provides access to quality content but also delivers personalized learning experiences tailored to individual needs. **An Online Learning Platform with Intelligent Tutoring** is designed to address these challenges by integrating advanced features and AI components to enhance the learning process.
* This platform enables users to create accounts easily, after which they receive a unique account number linked to their profile. This identification allows users to log in securely and access a range of features designed to facilitate their educational journey. Upon logging in, users are greeted with a customized homepage displaying available courses, making navigation intuitive and user-friendly.
* The platform provides a rich catalog of courses across various subjects, allowing users to explore, enrol, and add courses to their cart seamlessly. Once a course is purchased, users can engage with multimedia content, including video lectures, which can be played directly within the platform. This feature not only enhances user engagement but also caters to various learning styles.
* In addition to course access, the platform incorporates a Chabot feature that assists users in obtaining information such as course pricing and general inquiries, thereby enhancing the user experience through interactive support. Users are also encouraged to provide feedback on courses, which can be addressed by the admin, fostering a responsive and supportive learning environment.

**Existing System**

The existing systems for online learning primarily focus on providing a repository of courses and resources, but they often lack the interactive and personalized features necessary to enhance the learning experience effectively. Many traditional online learning platforms share common limitations, including:

### 1. Limited Personalization

Most existing systems deliver a one-size-fits-all approach, where learners receive the same content without consideration for their individual learning styles, preferences, or progress. This generic approach can lead to disengagement and reduced effectiveness in knowledge retention.

### 2. Static Content Delivery

### Current platforms often rely on pre-recorded video lectures and static content, which can become monotonous over time. Learners may find it challenging to stay motivated without interactive elements or engaging methods to assess their understanding of the material.

### 3. Ineffective Feedback Mechanisms

### Many platforms lack efficient feedback systems that allow learners to communicate their concerns or suggestions to instructors or administrators. This gap can result in unresolved issues that affect the overall user experience and course quality.

**Disadvantages of Existing System**

The existing online learning systems present several disadvantages that hinder their effectiveness and user satisfaction. These drawbacks can significantly impact the learning experience and overall educational outcomes.

### 1. Lack of Personalization

* **Generic Learning Paths**: Many systems offer standardized learning paths, failing to accommodate individual learning styles, pace, and preferences. This often results in disengagement and ineffective learning.
* **Limited Adaptability**: Existing platforms typically do not adjust content based on user performance or feedback, preventing tailored educational experiences that cater to students' unique needs.

### 2. Static and Unengaging Content

* **Monotonous Learning Materials**: Most platforms rely on pre-recorded lectures and static text, which can become tedious over time. The absence of interactive elements may lead to a decline in learner motivation.
* **Minimal Engagement Strategies**: Many platforms do not incorporate gamification, quizzes, or interactive assignments, reducing student involvement and making learning less enjoyable.

### 3. Inadequate Communication Channels

* **Poor Feedback Mechanisms**: Existing systems often lack efficient channels for students to communicate their concerns or feedback to instructors or administrators, resulting in unresolved issues and diminished user satisfaction.

**Proposed System**

The proposed online learning platform aims to address the limitations of existing systems by integrating intelligent tutoring components powered by AI. This innovative approach will enhance user engagement, personalize learning experiences, and streamline administrative tasks.

**1. User Registration and Account Management**

* **Automated Account Generation**: Upon registration, users will receive a unique account number generated based on their username and password. This enhances user identification and simplifies account management.
* **User Profiles**: Each user will have a personalized profile page displaying their details, enrolled courses, and progress, allowing them to track their learning journey effectively.

**2. Personalized Learning Experience**

* **Intelligent Tutoring System**: The platform will utilize AI algorithms to analyze user performance, learning styles, and preferences. This data will inform personalized course recommendations and learning paths, fostering more effective learning experiences.
* **Adaptive Content Delivery**: The system will adapt course materials in real-time based on user interactions, ensuring that learners receive content suited to their pace and comprehension level.

**Advantages of Proposed System**

The proposed online learning platform offers numerous advantages that enhance both user experience and administrative efficiency. These benefits address the limitations of existing systems while promoting an effective learning environment.

**1. Enhanced Learning Experience**

* **Personalized Learning Paths**: The intelligent tutoring system tailors course recommendations and content delivery based on individual user preferences, learning styles, and performance. This personalization helps learners absorb material more effectively and at their own pace.
* **Adaptive Content Delivery**: By analysing user interactions, the platform can modify content in real time, ensuring that users receive the most relevant materials to support their learning objectives.

**2. Comprehensive Course Management**

* **Diverse Course Offerings**: Users have access to a wide range of courses across various subjects, allowing them to choose and explore topics that interest them. This diversity encourages continuous learning and skill development.
* **User-Friendly Interface**: The intuitive design of the platform makes it easy for users to navigate through courses, assignments, and their profiles, leading to a more efficient learning experience.

**3. Efficient Administrative Processes**

**Streamlined Course Management**: Admins can easily add, edit, and delete courses, manage user accounts, and oversee assignments through a centralized dashboard.

**Feasibility Solution**

1. **Technical Feasibility:**   
   The system uses Django, a powerful and popular web framework, which supports secure user login, course management, video playback, and catboat features. It can be easily hosted on any server with basic technical setup.
2. **Economic Feasibility:**   
   The platform is cost-effective to build and maintain using open-source tools. No need for high-cost software or hardware, making it suitable even for small educational institutions or start-ups.
3. **Operational Feasibility:**   
   The platform is simple to use for both students and admins. With clear navigation and automated features like course purchase, video access, and catboat help, it requires minimal training.
4. **Legal Feasibility:**   
   The system follows basic legal standards like secure login, privacy protection, and safe online transactions. Any user data collected is handled carefully as per data protection rules.

**Frontend Technologies**

* **HTML** – To structure the web pages.
* **CSS** – For styling and layout design.
* **JavaScript** – To add interactivity and client-side functionality.
* **Bootstrap** – To create responsive and mobile-first web designs.

**Backend Technologies**

* **Python** – As the primary programming language.
* **Django** – A high-level Python web framework used for rapid development and clean design.

**Database**

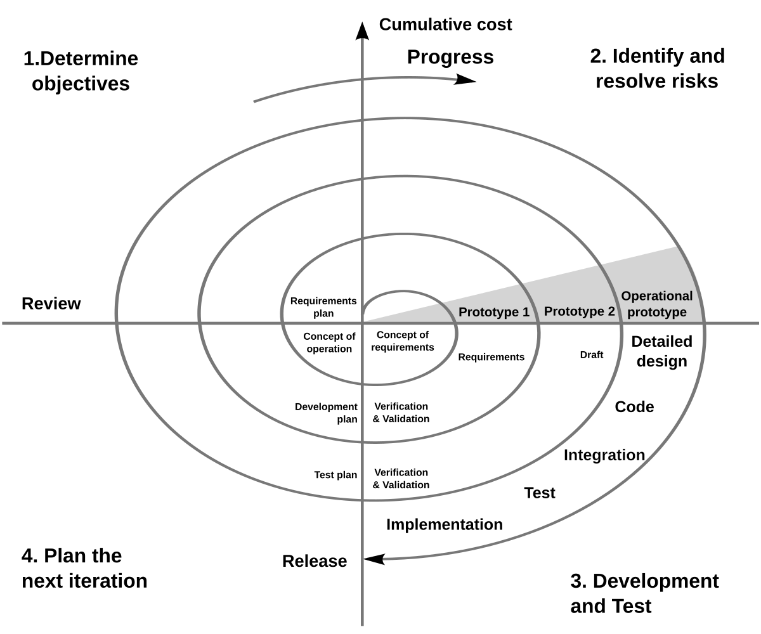
* **PostgreSQL** – A powerful, open-source object-relational database system used for storing and managing project data.

**Hardware Requirements**

**Operating** **system**: Windows 10 or 11.  
**Processor**: Intel I5 or I7 Processor.  
**RAM**: 4 GB or 8 GB.  
**Hard disk**: 1 TB or 256 SSD.

**Process Model**

For this Online Learning Platform, the  **Spirals Model** is best suited because it follows a clear and step-by-step approach. Here's how it works in simple terms:

1. **Requirement Gathering:**   
   Understand what the users and admins need from the platform.
2. **System Design:**   
   Plan how the website will look and work, including course pages, login system, Chabot, etc.
3. **Implementation:**   
   Develop the platform using Django, with all the required features like user registration, video playback, and admin controls.
4. **Testing:**   
   Check the system for any errors and make sure all features work smoothly.
5. **Deployment and Maintenance:**   
   Launch the platform for real users and keep updating it to fix bugs and add improvements.****

**Module Description**

 **User Registration and Login Module:**   
Allows users to sign up with their details and get a unique account. Users can log in securely using their username and password.

 **Course Management Module:**   
Displays a list of available courses. Users can browse, add courses to the cart, and purchase them to access video content.

 **Video Playback Module:**   
Lets users watch the videos of the courses they have purchased. The videos are streamed smoothly within the platform.

 **Payment and History Module:**   
Handles course payments securely. Users can view their past transactions and payment history.

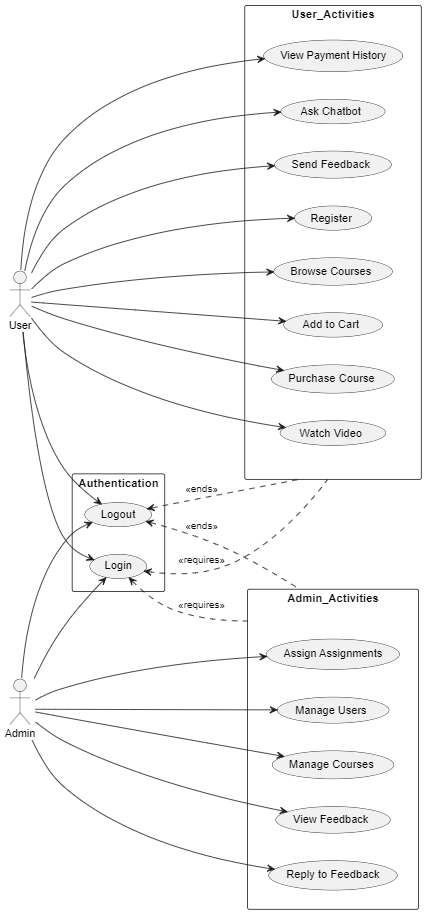
 **Chabot Module:**   
Provides instant replies to user queries, such as course pricing and general help, using AI.

 **Feedback Module:**   
Users can submit feedback about their learning experience. Admins can view and respond to the feedback.

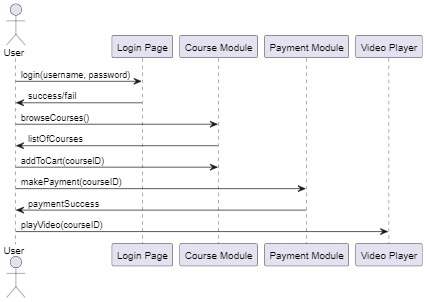
 **Admin Portal Module:**   
Admins can manage user accounts, edit or delete users, add/edit/remove courses, assign assignments, and reply to feedback.

**Pictorial (UML-as pre case study)**

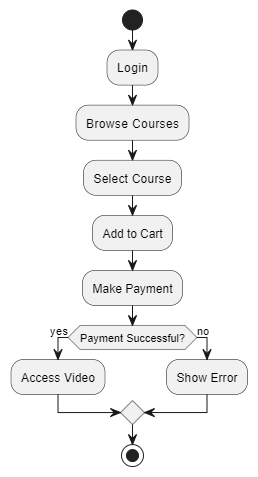
**Use-case study for every module**

****

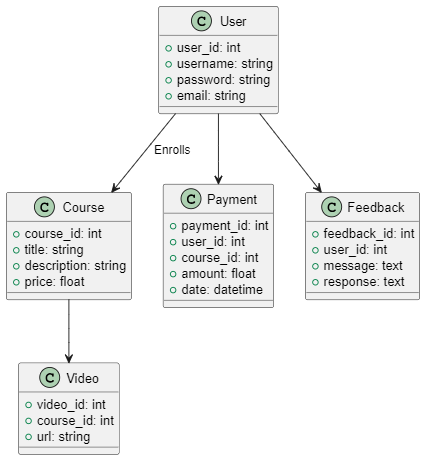
**Sequence Diagram**

****

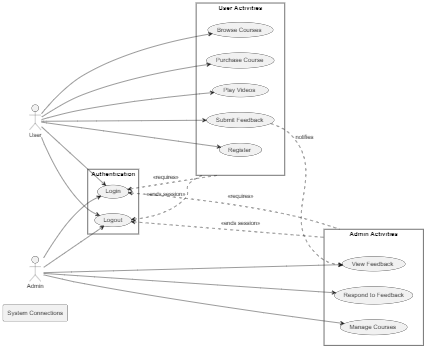
**Activity Diagram**

****

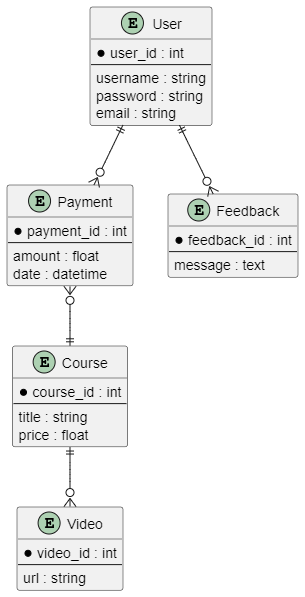
**Class Diagram**

****

**DFD Diagram**

****

**ER Diagram**

****

**Class Methods Diagram**

****

**Match to Process models.**

The Online Learning Platform with Intelligent Tutoring, developed using Django, provides a seamless and interactive learning experience through a structured process flow beginning with user registration and authentication, where unique account IDs are generated based on usernames and passwords. Upon login, users access personalized dashboards to browse and purchase courses, with integrated cart functionality and secure payment processing, while the system maintains detailed transaction records for reference. Video content from purchased courses is delivered via a dedicated player, ensuring smooth playback, and users can interact with an AI-powered chatbot for instant support on queries like pricing or course details. Feedback submission mechanisms allow users to share their experiences, which administrators monitor and address through a dedicated portal, enabling dynamic course and user management, including CRUD operations for accounts, course curation, assignment creation, and feedback resolution. The platform’s AI components further enhance engagement through personalized tutoring and automated responses, creating a closed-loop system where user input drives continuous improvement, resulting in a cohesive, scalable, and adaptive e-learning environment that benefits both learners and educators**.**

**Data Base Dictionary**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S.No** | **Field\_Name** | **Data\_Type** | **Keys** | **Constraints** | **Functionality / Description** |
| 1 | user\_id | INTEGER | PK | Auto Increment, Not Null | Unique ID for user |
|  | Username | VARCHAR(11) | UNIQUE | Not Null | User login name |
|  | Password | VARCHAR(7) |  | Not Null | Encrypted password |
|  | Email | VARCHAR(30) |  | Not Null | Email of the user |
| 2 | course\_id | INTEGER | PK | Auto Increment, Not Null | Unique course ID |
|  | Title | VARCHAR(30) |  | Not Null | Title of the course |
|  | Description | TEXT |  |  | Course details |
|  | Price | FLOAT |  | Not Null | Cost of the course |
| 3 | payment\_id | INTEGER | PK | Auto Increment, Not Null | Unique payment record |
|  | user\_id | INTEGER | FK | References User(user\_id) | ID of the user making payment |
|  | course\_id | INTEGER | FK | References Course(course\_id) | ID of course being purchased |
|  | Amount | FLOAT |  | Not Null | Amount paid |
|  | Date | TIMESTAMP |  | Default current\_timestamp | Payment date |
| 4 | video\_id | INTEGER | PK | Auto Increment, Not Null | Unique video ID |
|  | course\_id | INTEGER | FK | References Course(course\_id) | Course to which video belongs |
|  | url | VARCHAR(255) |  | Not Null | Path to video |
| 5 | feedback\_id | INTEGER | PK | Auto Increment, Not Null | Unique ID for feedback |
|  | user\_id | INTEGER | FK | References User(user\_id) | Feedback given by which user |
|  | Message | TEXT |  | Not Null | Feedback content |
|  | Response | TEXT |  | Nullable | Admin reply |

**Source Code**

from django.http import JsonResponse

from django.shortcuts import render

 # Create your views here.

# =====================================================

# outpage

def index(request):

    return render(request,'index.html')

# user sign up and login

# views.py

from django.contrib.auth.models import User

from django.shortcuts import render, redirect

from django.contrib.auth import authenticate, login

from django.contrib import messages

from .models import Cart, Payment, Purchase, Student

from django.utils import timezone

from datetime import datetime

import re

 def signup(request):

  if request.method == 'POST':

    username = request.POST['username']

     password = request.POST['password']

date\_of\_birth = request.POST['date\_of\_birth']

phone\_number = request.POST['phone\_number']

# Validate age

age = calculate\_age(date\_of\_birth)

if age < 15:

messages.error(request, "You must be at least 15 years old to sign up.")

return redirect('signup')

account\_number = generate\_account\_number()

# Create the user

user = User.objects.create\_user(username=username, password=password)

Student.objects.create(

user=user,

account\_number=account\_number,

date\_of\_birth=date\_of\_birth,

phone\_number=phone\_number

)

messages.success(request, "Account created successfully. Please log in.")

return redirect('login')

return render(request, 'signup.html')

# views.py

def login\_view(request):

if request.method == 'POST':

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

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

account\_number = request.POST.get('account\_number')

# Authenticate the user using username and password

user = authenticate(request, username=username, password=password)

login(request, user)

return redirect('profile')  # Redirect to profile page on successful logi

return render(request, 'login.html')

def calculate\_age(birth\_date):

today = timezone.now().date()

birth\_date = datetime.strptime(birth\_date, '%Y-%m-%d').date()

return today.year - birth\_date.year - ((today.month, today.day) < (birth\_date.month, birth\_date.day))

def generate\_account\_number():

last\_student = Student.objects.last()

if last\_student:

last\_number = int(last\_student.account\_number[-4:]) + 1

else:

last\_number = 1

return f"LPITAIC{last\_number:04d}"

# user profile

from django.shortcuts import render, redirect

from django.contrib.auth.decorators import login\_required

# ... (existing imports and code)

@login\_required

def profile(request):

return render(request, './user/profile.html')

from django.shortcuts import render, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from .models import Student

@login\_required

def userprofile(request):

student = request.user.student

return render(request, './user/userprofile.html', {'student': student})

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from django.shortcuts import render, redirect

from django.contrib.auth.models import User

@login\_required

def edit\_profile(request):

student = request.user.student  # Assuming you have a related Student model

if request.method == 'POST':

# Get updated values from the form

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

date\_of\_birth = request.POST.get('date\_of\_birth')

phone\_number = request.POST.get('phone\_number')

# Update the student object

student.date\_of\_birth = date\_of\_birth

student.phone\_number = phone\_number

# Update the User model username

if username and username != student.user.username:

try:

user = User.objects.get(username=username)

messages.error(request, 'This username is already taken. Please choose a different one.')

return render(request, './user/edit\_profile.html', {'student': student})

except User.DoesNotExist:

student.user.username = username

try:

student.save()  # Save the changes to the database

student.user.save()  # Save changes to the User model

messages.success(request, 'Profile updated successfully.')

except Exception as e:

messages.error(request, f"Error updating profile: {e}")

return redirect('userprofile')  # Redirect back to the profile page

return render(request, './user/edit\_profile.html', {'student': student})

# =========================================================================

# user logout

# views.py

from django.contrib.auth import logout

def logout\_view(request):

logout(request)

return redirect('login')

# ==========================================================================

# admin portal

from django.shortcuts import render, redirect

from django.contrib import messages

# View for admin login

def admin\_login(request):

if request.method == 'POST':

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

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

# Simple authentication check for hardcoded admin credentials

if username == 'rohit' and password == 'rohit':

request.session['is\_admin'] = True  # Set session variable

return redirect('admin\_portal')  # Redirect to admin portal after login

else:

messages.error(request, 'Invalid admin credentials.')

return render(request, './admin/admin\_login.html')

# View for admin portal

from django.contrib.auth.models import User  # Import User model if not already imported

def admin\_portal(request):

# Check if admin is logged in by verifying session variable

if not request.session.get('is\_admin'):

return redirect('admin\_login')  # Redirect to login if not logged in

courses = Course.objects.all()

course\_count = courses.count()  # Count the number of courses

user\_count = User.objects.count()

return render(request, 'admin/admin\_portal.html', {

'courses': courses,

'course\_count': course\_count,

'user\_count': user\_count

})

# Logout view

def admin\_logout(request):

request.session.flush()  # Clear the session

return redirect('admin\_login')

# edit and delete by admin

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.contrib.auth.decorators import user\_passes\_test

from django.contrib.auth.models import User

from django.contrib import messages

from .models import Student

# Check if the user is an admin

def admin\_check(user):

return user.is\_superuser

# Admin view to display all users

def admin\_user\_list(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')  # Redirect to login if not logged in

users = User.objects.all()

return render(request, 'admin/admin\_user\_list.html', {'users': users})

# Admin view to edit user profiles

def admin\_edit\_user(request, user\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')  # Redirect to login if not logged i

user = get\_object\_or\_404(User, id=user\_id)

student = user.student  # Assuming the user has a related Student object

if request.method == 'POST':

username = request.POST['username']

phone\_number = request.POST['phone\_number'

user.username = username

student.phone\_number = phone\_number

user.save()

student.save()

# Notify the user that their profile has been edited

messages.success(request, f"{user.username}, your profile has been updated by the admin.")

messages.success(request, f"User {user.username}'s profile updated successfully.")

return redirect('admin\_user\_list')

return render(request, 'admin/admin\_edit\_user.html', {'user': user, 'student': student})

# Admin view to delete a user

def admin\_delete\_user(request, user\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')  # Redirect to login if not logged in

user = get\_object\_or\_404(User, id=user\_id)

user.delete()

messages.success(request, f"User {user.username} deleted successfully.")

return redirect('admin\_user\_list')

# admin views.py

from django.shortcuts import render, redirect

from django.contrib import messages

from .models import Course

def admin\_add\_course(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')  # Redirect to login if not logged in

if request.method == 'POST':

title = request.POST['title']

description = request.POST['description']

price = request.POST['price']

video\_file = request.FILES.get('video\_file', None)  # Handle video file upload

# Create new course without video\_url

course = Course.objects.create(

title=title,

description=description,

price=price,

video\_file=video\_file  # Only keep video\_file

)

messages.success(request, f"Course '{title}' added successfully.")

return redirect('admin\_portal')

return render(request, 'admin/admin\_add\_course.html')

# admin views.py (continued)

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.contrib import messages

from .models import Course  # Assuming Course model is imported

# Admin view to list all courses

def admin\_course\_list(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

courses = Course.objects.all()

return render(request, 'admin/admin\_course\_list.html', {'courses': courses})

# Admin view to edit a course

def admin\_edit\_course(request, course\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login'

course = get\_object\_or\_404(Course, id=course\_id)

if request.method == 'POST':

course.title = request.POST['title']

course.description = request.POST['description']

course.price = request.POST['price']

course.save()

messages.success(request, f"Course '{course.title}' updated successfully.")

return redirect('admin\_course\_list')

return render(request, 'admin/admin\_edit\_course.html', {'course': course})

# Admin view to delete a course

def admin\_delete\_course(request, course\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

course = get\_object\_or\_404(Course, id=course\_id)

course.delete()

messages.success(request, f"Course '{course.title}' deleted successfully.")

return redirect('admin\_course\_list')

# User view to list all available courses

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from .models import Course, Cart, Purchase

# List all available courses

@login\_required

def course\_list(request):

courses = Course.objects.all()

return render(request, 'user/course\_list.html', {'courses': courses})

# Add a course to the cart

@login\_required

def add\_to\_cart(request, course\_id):

course = get\_object\_or\_404(Course, id=course\_id)

cart\_item, created = Cart.objects.get\_or\_create(user=request.user, course=course)

if created:

messages.success(request, f"Course '{course.title}' added to cart.")

else:

messages.info(request, f"Course '{course.title}' is already in your cart.")

return redirect('course\_list')

from django.contrib.auth.decorators import login\_required

from django.shortcuts import get\_object\_or\_404, redirect

from django.contrib import messages

from .models import Cart  # Import your Cart model

@login\_required

def remove\_from\_cart(request, cart\_item\_id):

cart\_item = get\_object\_or\_404(Cart, id=cart\_item\_id, user=request.user)

cart\_item.delete()  # Remove the cart item from the database

messages.success(request, "Course removed from cart.")

return redirect('view\_cart')  # Replace with the name of your cart view

@login\_required

def view\_cart(request):

# Get all cart items for the user

cart\_items = Cart.objects.filter(user=request.user)

# Exclude courses that the user has already purchased

purchased\_courses = Purchase.objects.filter(user=request.user).values\_list('course\_id', flat=True)

cart\_items = cart\_items.exclude(course\_id\_\_in=purchased\_courses)

return render(request, 'user/cart.html', {'cart\_items': cart\_items})

# Purchase a course

@login\_required

def purchase\_course(request, course\_id):

course = get\_object\_or\_404(Course, id=course\_id)

# Simulate successful payment process (replace this with real payment logic)

if process\_payment(request.user, course):

Purchase.objects.create(user=request.user, course=course)

messages.success(request, f"Payment successful! You now have access to the course '{course.title}'.")

return redirect('view\_course\_video', course\_id=course.id)

else:

messages.error(request, "Payment failed. Please try again.")

return redirect('view\_cart')

@login\_required

def view\_course\_video(request, course\_id):

course = get\_object\_or\_404(Course, id=course\_id)

# Check if the user has purchased the course

if not Purchase.objects.filter(user=request.user, course=course).exists():

messages.error(request, "You do not have access to this course. Please purchase it first.")

return redirect('course\_list')

return render(request, 'user/course\_video.html', {'course': course})

# Simulated payment process function (replace this with real payment logic)

def process\_payment(user, course):

# Simulate a successful payment

return True

@login\_required

def checkout(request, course\_id):

course = get\_object\_or\_404(Course, id=course\_id)

if request.method == 'POST':

# Extracting payment details from the request

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

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

account\_number = request.POST.get('account\_number')

card\_number = request.POST.get('card\_number')

expiry\_date = request.POST.get('expiry\_date')

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

# Convert the expiry date from MM-YY to YYYY-MM-DD

if expiry\_date:

year, month = map(int, expiry\_date.split('-'))

expiry\_date = f"{year}-{month:02d}-01"  # Setting day to 01

# Process the payment and check if it was successful

if process\_payment(request.user, course):  # Ensure process\_payment is defined

# Create Purchase

purchase = Purchase.objects.create(user=request.user, course=course)

# Save Payment details

Payment.objects.create(

purchase=purchase,

name=name,

age=age,

account\_number=account\_number,

card\_number=card\_number,  # Ideally, this should be encrypted

expiry\_date=expiry\_date,

cvv=cvv,  # Ideally, this should be encrypted

is\_successful=True

)

messages.success(request, f"Payment successful! You now have access to the course '{course.title}'.")

return redirect('view\_course\_video', course\_id=course.id)

else:

messages.error(request, "Payment failed. Please try again.")

return redirect('view\_course\_video', course\_id=course.id)

return render(request, 'user/checkout.html', {'course': course})

from django.shortcuts import render, get\_object\_or\_404

from .models import Purchase, Course

@login\_required

def purchased\_videos(request):

# Get all purchases for the logged-in user

purchases = Purchase.objects.filter(user=request.user).select\_related('course')

return render(request, 'user/purchased\_videos.html', {'purchases': purchases})

@login\_required

def payment\_history(request):

purchases = Purchase.objects.filter(user=request.user)

return render(request, 'user/payment\_history.html', {'purchases': purchases})

# views.py

from django.shortcuts import render, redirect, get\_object\_or\_404

from django.contrib import messages

from .models import Assignment, Submission

# views.py

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib import messages

from .models import Course, Assignment, AssignmentQuestion

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib import messages

from .models import Course, Assignment, AssignmentQuestion

def add\_assignment(request, course\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

course = get\_object\_or\_404(Course, id=course\_id)

if request.method == 'POST':

# Create a new Assignment

assignment = Assignment.objects.create(course=course)

# Loop through 5 questions and answers

for i in range(1, 6):

question = request.POST.get(f'question\_{i}')

answer = request.POST.get(f'answer\_{i}')

if question and answer:

# Create an AssignmentQuestion for each question and answer

AssignmentQuestion.objects.create(assignment=assignment, question=question, answer=answer)

messages.success(request, "Assignment with questions added successfully.")

return redirect('admin\_portal')

# Pass the range 1 to 5 for the template to iterate over

context = {

'course': course,

'question\_range': range(1, 6)  # Passing range to the template

}

return render(request, 'admin/add\_assignment.html', context)

def admin\_assignments\_history(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

assignments = Assignment.objects.prefetch\_related('submission\_set').all()

return render(request, 'admin/assignments\_history.html', {

'assignments': assignments

})

# ===========================

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from .models import Assignment, Submission

from django.shortcuts import render, get\_object\_or\_404, redirect

from django.contrib.auth.decorators import login\_required

from django.contrib import messages

from .models import Assignment, Submission, Purchase

@login\_required

def user\_assignments(request):

# Fetch assignments for courses purchased by the user

purchased\_courses = Purchase.objects.filter(user=request.user).values\_list('course', flat=True)

assignments = Assignment.objects.filter(course\_id\_\_in=purchased\_courses).prefetch\_related('questions')

# Loop through assignments and check if the user has submitted answers

for assignment in assignments:

assignment.submissions = Submission.objects.filter(assignment=assignment, user=request.user)

assignment.user\_has\_submitted = assignment.submissions.exists()

return render(request, 'user/user\_assignments.html', {

'assignments': assignments,  # Pass assignments with submission info

})

@login\_required

def submit\_assignment(request, assignment\_id):

assignment = get\_object\_or\_404(Assignment, id=assignment\_id)

# Check if the user has purchased the course for the assignment

purchased\_courses = Purchase.objects.filter(user=request.user).values\_list('course', flat=True)

if assignment.course.id not in purchased\_courses:

messages.error(request, "You need to purchase the course to access this assignment.")

return redirect('user\_assignments')

if request.method == 'POST':

results = []  # To hold results for each question

for question in assignment.questions.all():

user\_answer = request.POST.get(f'answer\_{question.id}')

correct\_answer = question.answer.strip().lower() if question.answer else ""

is\_correct = user\_answer.strip().lower() == correct\_answer

existing\_submission = Submission.objects.filter(user=request.user, assignment=assignment, question=question).first()

if existing\_submission:

existing\_submission.answer = user\_answer

existing\_submission.is\_correct = is\_correct

existing\_submission.save()

else:

Submission.objects.create(

assignment=assignment,

user=request.user,

question=question,

answer=user\_answer,

is\_correct=is\_correct

)

results.append({

'question': question,

'user\_answer': user\_answer,

'correct\_answer': correct\_answer,

'is\_correct': is\_correct

})

messages.success(request, "Your answers have been submitted successfully.")

return render(request, 'user/assignment\_results.html', {

'assignment': assignment,

'results': results,

})

return render(request, 'user/submit\_assignment.html', {

'assignment': assignment,

})

# views.py

from django.shortcuts import render

from .models import Payment

def admin\_payment\_history(request):

payments = Payment.objects.all()  # Retrieve all payments

return render(request, 'admin/payment\_history.html', {'payments': payments})

# ===============

# views.py

from django.shortcuts import render, redirect

from .models import Feedback, FeedbackReply

from django.contrib.auth.decorators import login\_required

@login\_required

def submit\_feedback(request):

if request.method == 'POST':

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

feedback = Feedback.objects.create(user=request.user, content=content)

return redirect('feedback\_success')  # Redirect to a success page or similar

return render(request, 'user/submit\_feedback.html')

@login\_required

def feedback\_success(request):

return render(request, 'user/feedback\_success.html')

def admin\_feedback\_list(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

feedbacks = Feedback.objects.all().prefetch\_related('replies')

return render(request, 'admin/feedback\_list.html', {'feedbacks': feedbacks})

def reply\_to\_feedback(request, feedback\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

feedback = Feedback.objects.get(id=feedback\_id)

if request.method == 'POST':

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

FeedbackReply.objects.create(feedback=feedback, admin=request.user, content=content)

feedback.is\_replied = True

feedback.save()

return redirect('admin\_feedback\_list')  # Redirect back to feedback list

return render(request, 'admin/reply\_feedback.html', {'feedback': feedback})

# views.py

from django.shortcuts import render

from .models import Feedback

from django.contrib.auth.decorators import login\_required

@login\_required

def submit\_feedback(request):

if request.method == 'POST':

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

feedback = Feedback.objects.create(user=request.user, content=content)

return redirect('feedback\_success')  # Redirect to a success page or similar

return render(request, 'user/submit\_feedback.html')

@login\_required

def feedback\_success(request):

return render(request, 'user/feedback\_success.html')

def admin\_feedback\_list(request):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

feedbacks = Feedback.objects.all().prefetch\_related('replies')

return render(request, 'admin/feedback\_list.html', {'feedbacks': feedbacks})

def reply\_to\_feedback(request, feedback\_id):

if not request.session.get('is\_admin'):

return redirect('admin\_login')

feedback = Feedback.objects.get(id=feedback\_id)

if request.method == 'POST':

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

FeedbackReply.objects.create(feedback=feedback, admin=request.user, content=content)

feedback.is\_replied = True

feedback.save()

return redirect('admin\_feedback\_list')  # Redirect back to feedback list

return render(request, 'admin/reply\_feedback.html', {'feedback': feedback})

# views.py

from django.shortcuts import render

from .models import Feedback

from django.contrib.auth.decorators import login\_required

@login\_required

def view\_feedback(request):

# Get all feedback submitted by the logged-in user

feedbacks = Feedback.objects.filter(user=request.user).order\_by('-created\_at')

return render(request, 'user/view\_feedback.html', {'feedbacks': feedbacks})

from django.http import JsonResponse

from .models import Course

@login\_required

def chatbot\_view(request):

if request.method == 'POST':

user\_input = request.POST.get('message')

username = request.user.username  # Get the logged-in username

user\_input\_lower = user\_input.lower().strip()  # Convert the user input to lowercase and strip spaces

# Handle greetings

if 'hello' in user\_input\_lower or 'hi' in user\_input\_lower:

response = f"Hello, {username}! How can I assist you today?"

# Handle course inquiries (individual course details)

elif 'course' in user\_input\_lower and 'show' not in user\_input\_lower:

# Extract course name from the user input

course\_name = user\_input\_lower.replace('course', '').strip()

# Try to find the course in the database

try:

course = Course.objects.get(title\_\_icontains=course\_name)

response = (f"Here are the details for the '{course.title}' course:\n"

f"Price: RS {course.price}\n"

f"Description: {course.description}\n"

f"Instructor: {course.instructor.name}\n"

f"Duration: {course.duration} weeks\n"

f"Start Date: {course.start\_date}\n"

f"Syllabus: {course.syllabus}")

except Course.DoesNotExist:

response = "I couldn't find the course you're asking for. Please try again or check the available courses

# Handle requests to show all available courses

elif 'show courses' in user\_input\_lower or 'available courses' in user\_input\_lower:

courses = Course.objects.all()

course\_list = '\n'.join([f"{course.title}: RS {course.price}" for course in courses])

response = f"Here are the available courses and their prices:\n{course\_list}"

# Handle requests specifically about course prices

elif 'price' in user\_input\_lower:

# Extract course name from the user input

course\_name = user\_input\_lower.replace('price', '').replace('of', '').strip()

# Check if any course matches the user's inquiry for price

try:

course = Course.objects.get(title\_\_icontains=course\_name)

response = (f"The price of the '{course.title}' course is RS {course.price}.\n"

f"Description: {course.description}")

except Course.DoesNotExist:

response = "I couldn't find the course you're asking for. Please try again."

# Handle requests about course content or syllabus

elif 'content' in user\_input\_lower or 'syllabus' in user\_input\_lower:

# Extract course name from the user input

course\_name = user\_input\_lower.replace('content', '').replace('syllabus', '').strip()

# Check if any course matches the user's inquiry for content or syllabus

try:

course = Course.objects.get(title\_\_icontains=course\_name)

response = f"The syllabus for '{course.title}' is as follows:\n{course.syllabus}"

except Course.DoesNotExist:

response = "I couldn't find the course you're asking for. Please try again."

# Handle chatbot farewell

elif 'bye' in user\_input\_lower or 'exit' in user\_input\_lower:

response = f"Goodbye, {username}! Have a great day!"

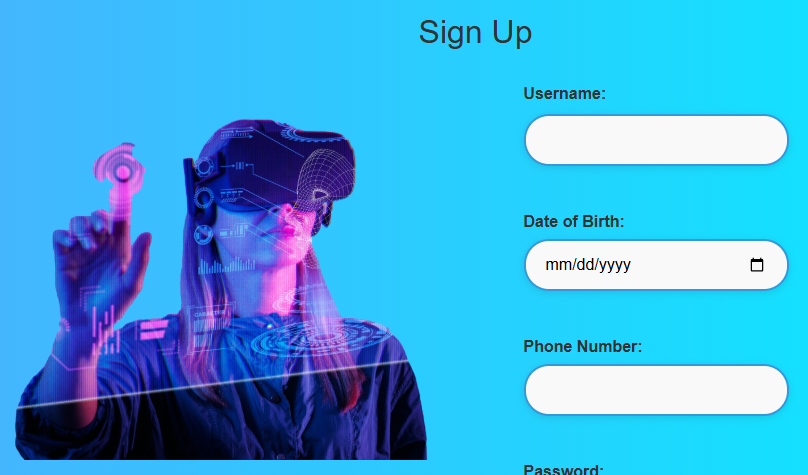
# Fallback response for unrecognized questions

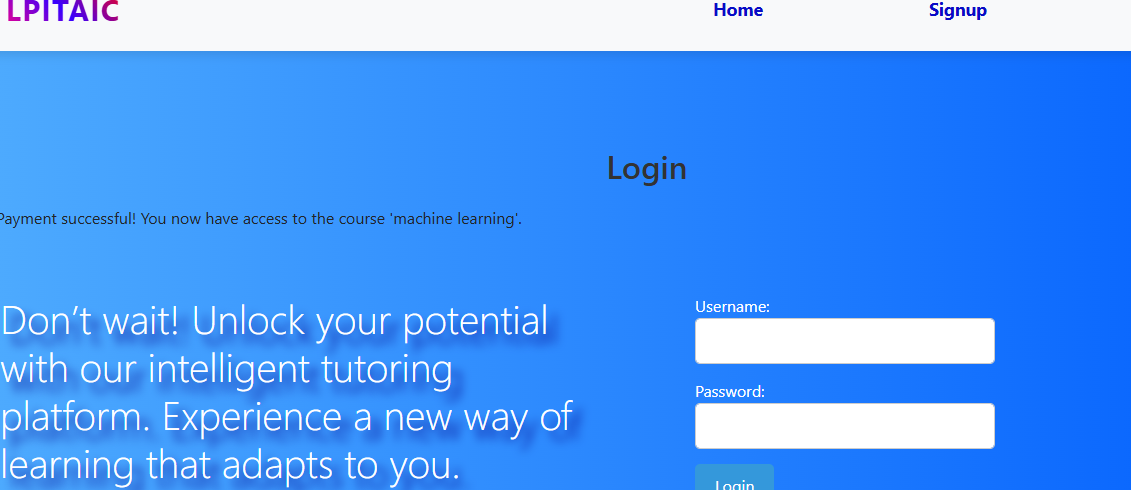
else:

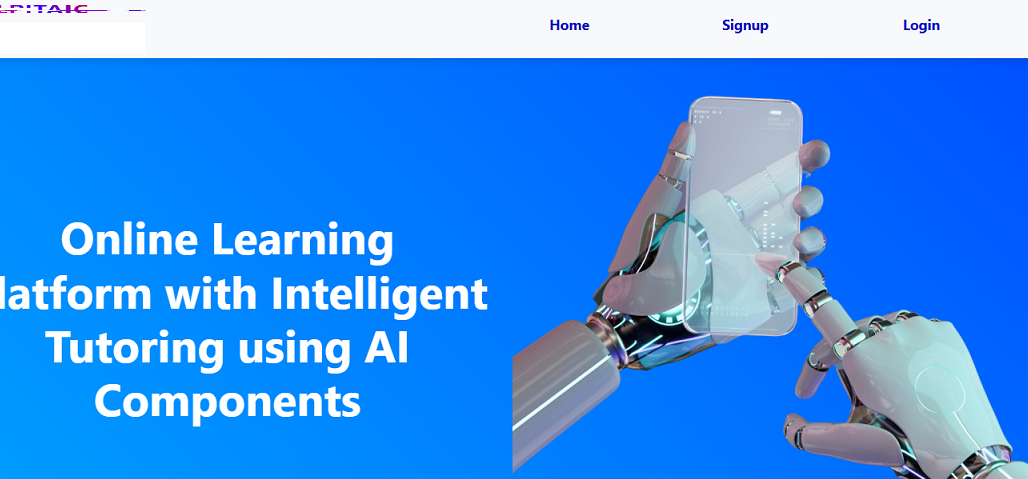
response = "I'm here to help! You can ask me about available courses, their prices, or any other questions."

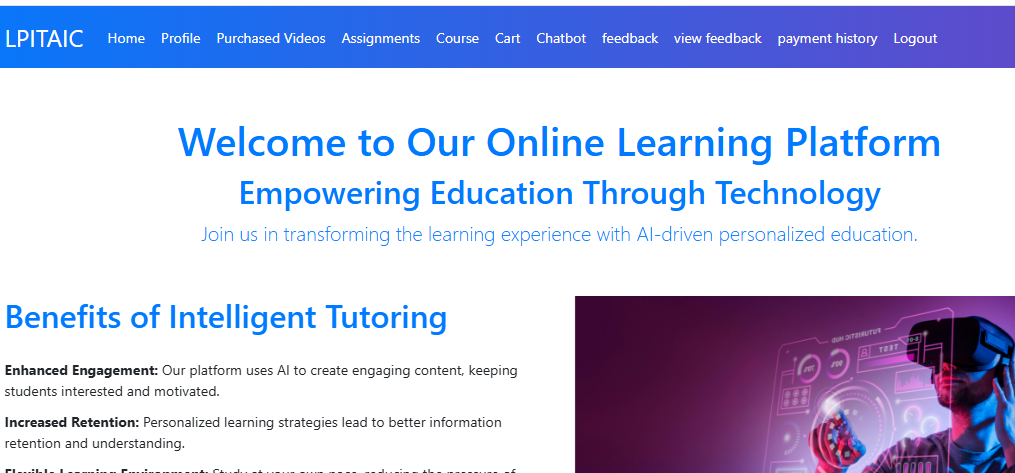
return JsonResponse({'response': response}

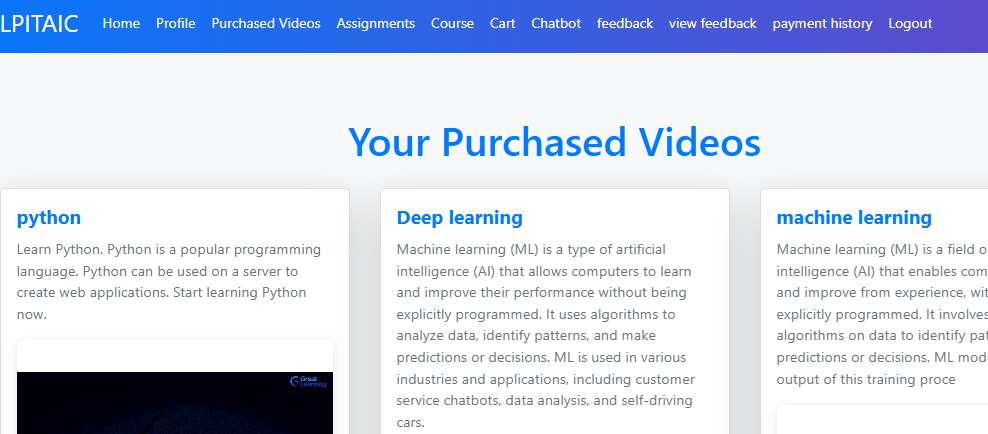
**Input and output Screens**

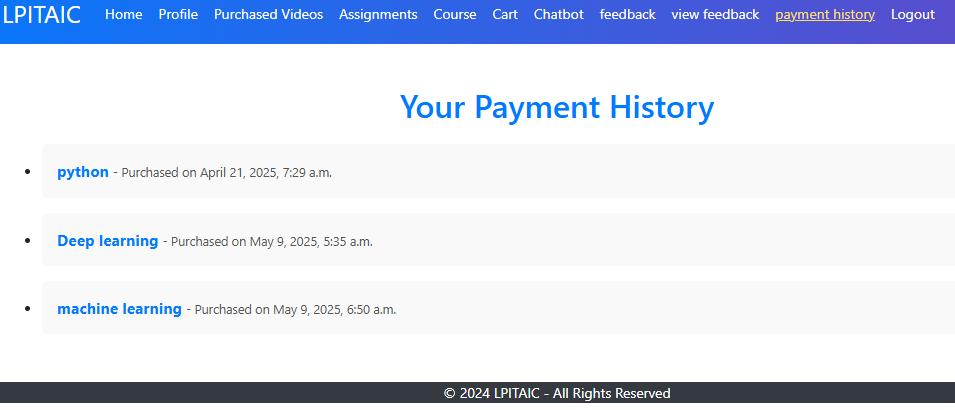
****

****

****

****

****

****