AI-Skilled-Navigator

Alhaan

2025-05-06 19:23:30

Contents

Project Overview

No project description provided.

Codebase Statistics

- **Total files:** 47
- **Total lines of code:** 1996
- **Languages used:** plaintext (1), python (46)
- **Generated by:** code2pdf v0.2

Directory Structure

```
AI-SkillNavigator/
    └── .gitignore
    └── manage.py
    ├── accounts/
        └── admin.py
        └── apps.py
        └── models.py
        └── tests.py
        └── urls.py
        └── views.py
        ├── templates/
            ├── accounts/
    ├── batch_allocation/
        └── admin.py
        └── apps.py
        └── batch_allocation_logic.py
        └── collaborative_filtering.py
        └── models.py
        └── tests.py
        └── urls.py
        └── utils.py
        └── views.py
        ├── templates/
            ├── batch_allocation/
    ├── core/
        └── asgi.py
        └── settings.py
        └── urls.py
    ├── feedback/
        └── admin.py
        └── apps.py
        └── forms.py
        └── models.py
        └── tests.py
        └── urls.py
        └── views.py
        ├── templates/
            ├── feedback/
    ├── profiles/
        └── admin.py
        └── apps.py
        └── forms.py
        └── models.py
        └── tests.py
        └── urls.py
        └── views.py
        ├── templates/
            ├── profiles/
    ├── test_/
        └── admin.py
        └── apps.py
        └── models.py
        └── tests.py
        └── urls.py
        └── utils.py
        └── views.py
        ├── templates/
            ├── test_/
    ├── visual/
        └── admin.py
        └── apps.py
        └── models.py
        └── tests.py
        └── urls.py
        └── views.py
        ├── templates/
```

```
            ├── visual/
```

D:/Programming/AI-SkillNavigator/.gitignore

```
# Ignore virtual environment
venv/

# Ignore Python cache files
__pycache__/
*.py[cod]

# Ignore Django migrations
migrations/
*/migrations/

# Env file
.env

templates/
__init__.py
code
db.sqlite3
README.md
requirements.txt
wsgi.py
.vscode
.venv
```

D:/Programming/AI-SkillNavigator/manage.py

```python
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys


def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'core.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)


if __name__ == '__main__':
    main()
```

D:/Programming/AI-SkillNavigator/accounts/admin.py

```
from django.contrib import admin

# Register your models here.
```

D:/Programming/AI-SkillNavigator/accounts/apps.py

```
from django.apps import AppConfig


class AccountsConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'accounts'
```

D:/Programming/AI-SkillNavigator/accounts/models.py

```
from django.db import models

# Create your models here.
```

D:/Programming/AI-SkillNavigator/accounts/tests.py

```
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/accounts/urls.py

```python
# accounts/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('', views.home_view, name='home'),
    path('signup/', views.signup_view, name='signup'),
    path('login/', views.login_view, name='login'),
    path('logout/', views.logout_view, name='logout'),
]
```

D:/Programming/AI-SkillNavigator/accounts/views.py

```python
# accounts/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.contrib import messages
from django.contrib.auth.decorators import login_required
from batch_allocation.models import Batch

def signup_view(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        confirm_password = request.POST['confirm_password']

        if password != confirm_password:
            messages.error(request, 'Passwords do not match')
            return render(request, 'accounts/signup.html')

        # Check if the username already exists
        if User.objects.filter(username=username).exists():
            messages.error(request, 'Username already exists, please choose a
↪   different one')
            return render(request, 'accounts/signup.html')

        # Create the user if the username is unique
        user = User.objects.create_user(username=username, password=password)
        user.save()
        messages.success(request, 'Account created successfully')
        return redirect('login')

    return render(request, 'accounts/signup.html')

def login_view(request):
    if request.method == 'POST':
        username = request.POST['username']
        password = request.POST['password']
        user = authenticate(request, username=username, password=password)

        if user is not None:
            login(request, user)
            return redirect('home')  # redirect to home page or dashboard
        else:
            messages.error(request, 'Invalid credentials')
    return render(request, 'accounts/login.html')

def logout_view(request):
    logout(request)
    return redirect('login')

@login_required
def home_view(request):
    # Get the user's batches
    user_batches = request.user.batches.all()  # This gets all batches associated
 ↪  with the user

    # If the user has any batches, select the first one (or apply your own logic)
    if user_batches.exists():
        batch = user_batches.first()  # You can modify this logic as needed
    else:
        batch = None  # Handle the case where the user has no batches

    return render(request, 'accounts/home.html', {'batch': batch})
```

D:/Programming/AI-SkillNavigator/batch_allocation/admin.py

```
# batch_allocation/admin.py
from django.contrib import admin
from .models import Batch

@admin.register(Batch)
class BatchAdmin(admin.ModelAdmin):
    list_display = ('name', 'programming_languages', 'min_candidates',
↪   'max_candidates', 'current_candidates')
    search_fields = ('name', 'programming_languages')
    list_filter = ('min_candidates', 'max_candidates')
    ordering = ('name',)  # Order by name

    def candidate_count(self, obj):
        return obj.candidates.count()
    candidate_count.short_description = 'Number of Candidates'
```

D:/Programming/AI-SkillNavigator/batch_allocation/apps.py

```
from django.apps import AppConfig


class BatchAllocationConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'batch_allocation'
```

D:/Programming/AI-SkillNavigator/batch_allocation/batch_allocation_logic.py

```
from collections import namedtuple

Batch = namedtuple("Batch", ["name", "max_candidates", "current_candidates",
↪   "candidates"])

batches = [
    Batch(name="Java Jedi Academy", max_candidates=5, current_candidates=2,
↪   candidates=set()),
    Batch(name=".NET Ninja Clan", max_candidates=3, current_candidates=1,
↪   candidates=set()),
    Batch(name="Python Wizardry Guild", max_candidates=4, current_candidates=3,
↪   candidates=set()),
]

def allocate_batch(user_id, programming_languages):
    languages = [lang.strip().lower() for lang in programming_languages.split(",")]
    selected_batch = None

    if "java" in languages:
        selected_batch = next((b for b in batches if b.name == "Java Jedi
↪   Academy"), None)
    elif ".net" in languages:
        selected_batch = next((b for b in batches if b.name == ".NET Ninja Clan"),
↪   None)
    elif "python" in languages:
        selected_batch = next((b for b in batches if b.name == "Python Wizardry
↪   Guild"), None)

    if selected_batch:
        if user_id in selected_batch.candidates:
            return f"User {user_id} is already enrolled in {selected_batch.name}."
        if selected_batch.current_candidates < selected_batch.max_candidates:
            selected_batch.candidates.add(user_id)
            selected_batch.current_candidates += 1
           return f"User {user_id} successfully enrolled in {selected_batch.name}."
        else:
            return f"{selected_batch.name} is full. Try another batch."

    return "No suitable batch found for the user."

if __name__ == "__main__":
    users = [
        {"id": 1, "languages": "Java, Python"},
        {"id": 2, "languages": ".NET, Java"},
        {"id": 3, "languages": "Python"},
        {"id": 4, "languages": "Ruby, Go"},
    ]

    for user in users:
        result = allocate_batch(user["id"], user["languages"])
        print(result)
```

D:/Programming/AI-SkillNavigator/batch_allocation/collaborative_filtering.py

```
from sklearn.metrics.pairwise import cosine_similarity
import numpy as np

def collaborative_filtering(user_vector, candidate_vectors):
    user_vector = np.array(user_vector).reshape(1, -1)
    candidate_vectors = np.array(candidate_vectors)
```

```python
    similarities = cosine_similarity(user_vector, candidate_vectors)
    recommended_indices = np.argsort(-similarities[0])
    return recommended_indices.tolist()

if __name__ == "__main__":
    user_vector = [1, 0, 1, 0, 1]
    courses = [
        {"name": "Java Jedi Mastery", "vector": [1, 0, 0, 0, 1]},
        {"name": "Python for Data Wizards", "vector": [0, 1, 1, 0, 1]},
        {"name": "C# .NET Ninja Training", "vector": [1, 1, 0, 0, 0]},
        {"name": "Full-Stack Sorcery", "vector": [0, 0, 1, 1, 1]},
        {"name": "AI Alchemy 101", "vector": [1, 0, 1, 1, 0]},
        {"name": "Cloud Computing Chronicles", "vector": [0, 1, 0, 1, 1]},
        {"name": "Cybersecurity Secrets", "vector": [1, 1, 1, 0, 0]},
        {"name": "DevOps Dungeon Crawl", "vector": [0, 0, 0, 1, 1]},
    ]
    candidate_vectors = [course["vector"] for course in courses]
    recommendations = collaborative_filtering(user_vector, candidate_vectors)
    print("Top Course Recommendations:")
    for rank, index in enumerate(recommendations, start=1):
        course = courses[index]
        print(f"{rank}. {course['name']} (Features: {course['vector']})")
    selected_course_index = recommendations[0]
    selected_course = courses[selected_course_index]
    print("\nUser selected course:")
    print(f"Name: {selected_course['name']}")
    print(f"Features: {selected_course['vector']}")
    updated_user_vector = [
        max(user_skill, course_skill)
        for user_skill, course_skill in zip(user_vector, selected_course["vector"])
    ]
    print("\nUpdated User Preferences:")
    print(updated_user_vector)
    new_recommendations = collaborative_filtering(updated_user_vector,
↪   candidate_vectors)
    print("\nNew Top Course Recommendations:")
    for rank, index in enumerate(new_recommendations, start=1):
        course = courses[index]
        print(f"{rank}. {course['name']} (Features: {course['vector']})")
```

D:/Programming/AI-SkillNavigator/batch_allocation/models.py

```python
# models.py
```

```python
from django.db import models
from django.contrib.auth.models import User


class Batch(models.Model):
    name = models.CharField(max_length=100)
    programming_languages = models.CharField(max_length=255)
    min_candidates = models.PositiveIntegerField(default=25)
    max_candidates = models.PositiveIntegerField(default=30)
    current_candidates = models.PositiveIntegerField(default=0)  # Track the number
↪   of candidates
    candidates = models.ManyToManyField(User, related_name='batches')  #
↪   Relationship with User

    def __str__(self):
        return self.name
```

D:/Programming/AI-SkillNavigator/batch_allocation/tests.py

```python
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/batch_allocation/urls.py

```python
# batch/urls.py
from django.urls import path
from .views import batch_enrollment_view, course_page

urlpatterns = [
    path('enroll/', batch_enrollment_view, name='batch_enrollment'),
    path('<int:batch_id>/course/', course_page, name='course_page')

]
```

D:/Programming/AI-SkillNavigator/batch_allocation/utils.py

```python
# batch_allocation/utils.py

from .models import Batch
import google.generativeai as genai
import markdown
from bs4 import BeautifulSoup
import os
from dotenv import load_dotenv

load_dotenv()
api_key = os.getenv('GOOGLE_API_KEY')

# Configure API Key
if not api_key:
    raise ValueError("GOOGLE_API_KEY not found in environment variables. Please set
     ↪ it in your .env file.")

genai.configure(api_key=api_key)

def allocate_batch(user):
    languages = [lang.strip().lower() for lang in
 ↪  user.profile.programming_languages.split(',')]
    batch = None

    # Check which batch to allocate based on programming languages
    if 'java' in languages:
        batch = Batch.objects.filter(name='Java Batch').first()
    elif '.net' in languages:
        batch = Batch.objects.filter(name='.NET Batch').first()
    elif 'python' in languages:
        batch = Batch.objects.filter(name='Data Engineering Batch').first()

    # Check if the batch can accept more candidates
    if batch:
        # Check if the user is already in the batch
        if batch.candidates.filter(id=user.id).exists():
            return "already_enrolled", batch  # Return a specific message for
             ↪ existing enrollment

        if batch.current_candidates < batch.max_candidates:
            batch.current_candidates += 1  # Increment the current candidate count
            batch.candidates.add(user)  # Add user to the batch
            batch.save()  # Save the updated batch
            return "enrolled", batch

    return "no_batch", None
```

```python
def md_to_text(md):
    # Convert Markdown to HTML
    html = markdown.markdown(md)

    # Parse the HTML with BeautifulSoup
    soup = BeautifulSoup(html, features='html.parser')

    # Optionally add custom styles or classes
    for heading in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6']):
        heading['class'] = 'my-heading-class'  # Add a custom CSS class for styling

    # Return the prettified HTML
    return soup.prettify()  # Use prettify() for better formatting (optional)

def generate_content(topic):
    ask = f"Provide a detailed explanation about the topic '{topic}'."
    try:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(ask)

        # Get the generated text
        generated_text = response.text

        # Convert the generated text to Markdown format
        beautiful_content = md_to_text(generated_text)
        return beautiful_content
    except Exception as e:
        print(f"Error generating content: {e}")
        return "Content generation unavailable at the moment."
```

D:/Programming/AI-SkillNavigator/batch_allocation/views.py

```python
# batch/views.py
from django.shortcuts import render, get_object_or_404
from .models import Batch
from django.contrib.auth.decorators import login_required
from django.contrib import messages
from .utils import allocate_batch, generate_content

@login_required
def batch_enrollment_view(request):
    # Allocate batch for the user
    status, batch = allocate_batch(request.user)
    # Check the allocation status and display the appropriate message
    if status == "enrolled":
        messages.success(request, f"You have been allocated to the {batch.name}.")
    elif status == "already_enrolled":
        messages.info(request, f"You are already enrolled in the {batch.name}.")
    else:
        messages.error(request, "No suitable batch available or batch is full.")

    return render(request, 'batch_allocation/batch_enrollment.html', {
        'batch': batch,
    })

@login_required
def course_page(request, batch_id):
    # Fetch the batch and its topics
    batch = get_object_or_404(Batch, id=batch_id)
    topics = batch.topics.all()

    # Default topic material
    selected_topic = request.GET.get('topic', None)  # Get the selected topic,
 ↪  default to None
    material = None

    if selected_topic:
        # Generate learning material based on the selected topic
        material = generate_content(selected_topic)  # Function to generate content

    context = {
        'batch': batch,
        'topics': topics,
        'selected_topic': selected_topic,
        'material': material,
    }

    # Render only the content part for AJAX requests
    if request.headers.get('x-requested-with') == 'XMLHttpRequest':
        return render(request, 'batch_allocation/partial_material.html', context)

    # Render full page for non-AJAX requests
    return render(request, 'batch_allocation/course_page.html', context)
```

D:/Programming/AI-SkillNavigator/core/asgi.py

```
"""
ASGI config for core project.

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

For more information on this file, see
https://docs.djangoproject.com/en/5.1/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

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

application = get_asgi_application()
```

D:/Programming/AI-SkillNavigator/core/settings.py

```
"""
Django settings for core project.

Generated by 'django-admin startproject' using Django 5.1.2.

For more information on this file, see
https://docs.djangoproject.com/en/5.1/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/5.1/ref/settings/
"""

from pathlib import Path
from datetime import timedelta


# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent


# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/5.1/howto/deployment/checklist/
```

```python
# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-+dzbs2e1iqm$0*+nduiw&jo!2&qtl5z0cu0eajtwtr+e4@p+t@'

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

ALLOWED_HOSTS = []


# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'rest_framework',
    'rest_framework_simplejwt',
    'accounts',
    'profiles',
    'batch_allocation',
    'visual',
    'test_',
    'feedback',
]
REST_FRAMEWORK = {
    'DEFAULT_AUTHENTICATION_CLASSES': (
        'rest_framework_simplejwt.authentication.JWTAuthentication',
    ),
}

# Optional: You can customize JWT settings if needed
SIMPLE_JWT = {
    'ACCESS_TOKEN_LIFETIME': timedelta(minutes=5),  # Adjust as needed
    'REFRESH_TOKEN_LIFETIME': timedelta(days=1),
    'ROTATE_REFRESH_TOKENS': True,
    'BLACKLIST_AFTER_ROTATION': True,
}
MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'core.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'core.wsgi.application'


# Database
# https://docs.djangoproject.com/en/5.1/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}


# Password validation
# https://docs.djangoproject.com/en/5.1/ref/settings/#auth-password-validators
```

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


# Internationalization
# https://docs.djangoproject.com/en/5.1/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True


# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/5.1/howto/static-files/

STATIC_URL = 'static/'

# Default primary key field type
# https://docs.djangoproject.com/en/5.1/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_REDIRECT_URL = 'home'
LOGOUT_REDIRECT_URL = 'login'

import os
```

```
from pathlib import Path

BASE_DIR = Path(__file__).resolve().parent.parent

MEDIA_URL = '/media/'
MEDIA_ROOT = os.path.join(BASE_DIR, 'certificates')  #
```

D:/Programming/AI-SkillNavigator/core/urls.py

```
# urls.py
from django.urls import path, include
from django.shortcuts import redirect
from django.contrib import admin
from django.conf import settings
from django.conf.urls.static import static
from rest_framework_simplejwt.views import TokenObtainPairView, TokenRefreshView

def redirect_to_home(request):
    if request.user.is_authenticated:
        return redirect('home')
    else:
        return redirect('login')

urlpatterns = [
    path('admin/', admin.site.urls),
   path('', redirect_to_home),  # Redirect to login or home based on authentication
    path('accounts/', include('accounts.urls')),
    path('profile/', include('profiles.urls')),
    path('batch/', include('batch_allocation.urls')),
    path('test/', include('test_.urls')),
    path('visualization/', include('visual.urls')),
    path('feedback/', include('feedback.urls')),
    path('api/token/', TokenObtainPairView.as_view(), name='token_obtain_pair'),
    path('api/token/refresh/', TokenRefreshView.as_view(), name='token_refresh'),

] + static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

D:/Programming/AI-SkillNavigator/feedback/admin.py

```
# feedback/admin.py
from django.contrib import admin
from .models import Feedback
```

```
@admin.register(Feedback)
class FeedbackAdmin(admin.ModelAdmin):
    list_display = ('user', 'course_quality', 'test_quality',
 ↪  'website_experience', 'created_at')
    search_fields = ('user__username',)
    list_filter = ('course_quality', 'test_quality', 'website_experience',
 ↪  'created_at')
    ordering = ('-created_at',)  # Order by the most recent feedback

    def has_change_permission(self, request, obj=None):
        return False  #
```

D:/Programming/AI-SkillNavigator/feedback/apps.py

```
from django.apps import AppConfig


class FeedbackConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'feedback'
```

D:/Programming/AI-SkillNavigator/feedback/forms.py

```
from django import forms
from .models import Feedback

class FeedbackForm(forms.ModelForm):
    class Meta:
        model = Feedback
        fields = ['course_quality', 'test_quality', 'website_experience',
 ↪  'additional_feedback']
        widgets = {
            'course_quality': forms.Select(attrs={'class': 'form-select'}),
            'test_quality': forms.Select(attrs={'class': 'form-select'}),
            'website_experience': forms.Select(attrs={'class': 'form-select'}),
            'additional_feedback': forms.Textarea(attrs={
                'class': 'form-control',  # Bootstrap class for styling
                'rows': 4,
                'placeholder': 'Enter any additional feedback here...',
                'aria-label': 'Additional Feedback',  # Accessibility improvement
            }),
        }
```

D:/Programming/AI-SkillNavigator/feedback/models.py

```
from django.db import models
from django.contrib.auth.models import User

class Feedback(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE,
↪   related_name='feedbacks')

    # Questions about courses, tests, and the website
    course_quality = models.IntegerField(choices=[
        (1, '1 - Worst'),
        (2, '2 - Poor'),
        (3, '3 - Average'),
        (4, '4 - Good'),
        (5, '5 - Excellent')
    ])
    test_quality = models.IntegerField(choices=[
        (1, '1 - Worst'),
        (2, '2 - Poor'),
        (3, '3 - Average'),
        (4, '4 - Good'),
        (5, '5 - Excellent')
    ])
    website_experience = models.IntegerField(choices=[
        (1, '1 - Worst'),
        (2, '2 - Poor'),
        (3, '3 - Average'),
        (4, '4 - Good'),
        (5, '5 - Excellent')
    ])
    additional_feedback = models.TextField(blank=True, null=True)  # Optional
↪   feedback

    created_at = models.DateTimeField(auto_now_add=True)

    def __str__(self):
        return f"Feedback from {self.user.username} on {self.created_at}"
```

D:/Programming/AI-SkillNavigator/feedback/tests.py

```python
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/feedback/urls.py

```python
from django.urls import path
from .views import feedback_view, feedback_summary_view

urlpatterns = [
    path('', feedback_view, name='feedback'),
    path('summary/', feedback_summary_view, name='feedback_summary'),

]
```

D:/Programming/AI-SkillNavigator/feedback/views.py

```python
from django.shortcuts import render, redirect
from django.contrib import messages
from .forms import FeedbackForm
from django.contrib.auth.decorators import login_required
from django.db.models import Q


@login_required
def feedback_view(request):
    if request.method == 'POST':
        form = FeedbackForm(request.POST)
        if form.is_valid():
            feedback = form.save(commit=False)
            feedback.user = request.user  # Associate feedback with the logged-in
↪   user
            feedback.save()
            messages.success(request, 'Thank you for your feedback!')
            return redirect('feedback')  # Redirect to the same feedback page or a
              ↪  thank you page
    else:
        form = FeedbackForm()

    return render(request, 'feedback/feedback.html', {'form': form})
```

```
from django.shortcuts import render
from .models import Feedback
from textblob import TextBlob
from django.contrib.auth.decorators import login_required

@login_required
def feedback_summary_view(request):
    feedbacks = Feedback.objects.all()

    # Initialize counters for each rating category
    total_course_quality = 0
    total_test_quality = 0
    total_website_experience = 0
    feedback_count = feedbacks.count()

    # Collect text feedback for sentiment analysis
    all_feedback_texts = []

    for feedback in feedbacks:
        total_course_quality += feedback.course_quality
        total_test_quality += feedback.test_quality
        total_website_experience += feedback.website_experience
        if feedback.additional_feedback:
            all_feedback_texts.append(feedback.additional_feedback)

    # Calculate average ratings
    avg_course_quality = total_course_quality / feedback_count if feedback_count
↪   else 0
    avg_test_quality = total_test_quality / feedback_count if feedback_count else 0
    avg_website_experience = total_website_experience / feedback_count if
↪   feedback_count else 0

    # Perform sentiment analysis on additional feedback
    combined_text = " ".join(all_feedback_texts)
    sentiment_analysis = TextBlob(combined_text).sentiment
    overall_sentiment = 'Positive' if sentiment_analysis.polarity > 0 else
↪   'Negative' if sentiment_analysis.polarity < 0 else 'Neutral'

    feedback_list = Feedback.objects.exclude(Q(additional_feedback="") |
↪   Q(additional_feedback__isnull=True))

    # Render data to template
    context = {
        'avg_course_quality': avg_course_quality,
        'avg_test_quality': avg_test_quality,
        'avg_website_experience': avg_website_experience,
        'overall_sentiment': overall_sentiment,
        'sentiment_score': sentiment_analysis.polarity,
        'feedback_list': feedback_list,
    }

    return render(request, 'feedback/feedback_summary.html', context)
```

D:/Programming/AI-SkillNavigator/profiles/admin.py

```
# profile/admin.py
from django.contrib import admin
from .models import Profile, Course, Internship, Certification

@admin.register(Profile)
class ProfileAdmin(admin.ModelAdmin):
    list_display = ('user', 'name', 'email', 'degree', 'specialization',
 ↪  'phone_number')
    search_fields = ('user__username', 'name', 'email')

@admin.register(Course)
class CourseAdmin(admin.ModelAdmin):
    list_display = ('user', 'name', 'platform', 'certificate')
    search_fields = ('user__username', 'name', 'platform')
    list_filter = ('user',)

@admin.register(Internship)
class InternshipAdmin(admin.ModelAdmin):
    list_display = ('user', 'title', 'company', 'start_date', 'end_date',
 ↪  'certificate')
    search_fields = ('user__username', 'title', 'company')
    list_filter = ('user',)

@admin.register(Certification)
class CertificationAdmin(admin.ModelAdmin):
    list_display = ('user', 'name', 'certificate')
    search_fields = ('user__username', 'name')
    list_filter = ('user',)
```

D:/Programming/AI-SkillNavigator/profiles/apps.py

```
from django.apps import AppConfig


class ProfilesConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'profiles'
```

D:/Programming/AI-SkillNavigator/profiles/forms.py

```
from django import forms
from .models import Profile, Course, Internship, Certification
from django import forms

class ProfileForm(forms.ModelForm):
    PROGRAMMING_LANGUAGE_CHOICES = [
        ('Python', 'Python'),
        ('.Net', '.NET'),
        ('Java', 'Java'),
    ]

    programming_languages = forms.ChoiceField(
        choices=PROGRAMMING_LANGUAGE_CHOICES,
        widget=forms.Select(attrs={'class': 'form-select'})  # Add Bootstrap class
↪   for styling
    )

    class Meta:
        model = Profile
        fields = [
            'name', 'email', 'degree', 'specialization', 'phone_number',
            'linkedin_profile', 'github_profile', 'programming_languages'
        ]
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder':
             ↪  'Full Name'}),
            'email': forms.EmailInput(attrs={'class': 'form-control',
             ↪  'placeholder': 'Email Address'}),
            'degree': forms.TextInput(attrs={'class': 'form-control',
             ↪  'placeholder': 'Degree'}),
            'specialization': forms.TextInput(attrs={'class': 'form-control',
             ↪  'placeholder': 'Specialization'}),
            'phone_number': forms.TextInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Phone Number'}),
            'linkedin_profile': forms.URLInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'LinkedIn Profile'}),
            'github_profile': forms.URLInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'GitHub Profile'}),
            # 'programming_languages' field is now handled above with a ChoiceField
        }


class CourseForm(forms.ModelForm):
    class Meta:
        model = Course
        fields = ['name', 'platform', 'certificate']  # Include the certificate
↪   field
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder':
            ↪  'Course Name'}),
            'platform': forms.TextInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Platform'}),
            'certificate': forms.ClearableFileInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Upload Certificate (PDF)'}),  # PDF upload field
        }

class InternshipForm(forms.ModelForm):
    class Meta:
        model = Internship
        fields = ['title', 'company', 'start_date', 'end_date', 'certificate']  #
↪   Include the certificate field
        widgets = {
            'title': forms.TextInput(attrs={'class': 'form-control', 'placeholder':
            ↪  'Internship Title'}),
            'company': forms.TextInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Company'}),
            'start_date': forms.DateInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Start Date', 'type': 'date'}),
            'end_date': forms.DateInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'End Date', 'type': 'date'}),
            'certificate': forms.ClearableFileInput(attrs={'class': 'form-control',
            ↪  'placeholder': 'Upload Certificate (PDF)'}),  # PDF upload field
        }

class CertificationForm(forms.ModelForm):
    class Meta:
        model = Certification
        fields = ['name', 'certificate']  # Include the certificate field
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control', 'placeholder':
             ↪   'Certification Name'}),
           'certificate': forms.ClearableFileInput(attrs={'class': 'form-control',
             ↪   'placeholder': 'Upload Certificate (PDF)'}),  # PDF upload field
        }
```

D:/Programming/AI-SkillNavigator/profiles/models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.utils import timezone


class Profile(models.Model):
    PROGRAMMING_LANGUAGES = [
        ('Python', 'Python'),
        ('.Net', '.Net'),
        ('Java', 'Java'),
    ]

    user = models.OneToOneField(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=100)
    email = models.EmailField()
    degree = models.CharField(max_length=100)
    specialization = models.CharField(max_length=100)
    phone_number = models.CharField(max_length=15)
    linkedin_profile = models.URLField(blank=True, null=True)
    github_profile = models.URLField(blank=True, null=True)
    programming_languages = models.CharField(
        max_length=7,
        choices=PROGRAMMING_LANGUAGES,
        help_text="Choose a programming language",
    )

    def __str__(self):
        return self.user.username

class Course(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=200)
    platform = models.CharField(max_length=200)
    certificate = models.FileField(upload_to='certificates/courses/', blank=True,
↪   null=True)  # Optional PDF upload

    class Meta:
        unique_together = (('user', 'name', 'platform'),)

class Internship(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    title = models.CharField(max_length=200)
    company = models.CharField(max_length=200)
    start_date = models.DateField(default=timezone.now)  # Set default to current
↪   date
    end_date = models.DateField(default=timezone.now)
    certificate = models.FileField(upload_to='certificates/internships/',
↪   blank=True, null=True)  # Optional PDF upload

    class Meta:
        unique_together = (('user', 'title', 'company', 'start_date'),)

class Certification(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    name = models.CharField(max_length=200, blank=True)
    certificate = models.FileField(upload_to='certificates/certifications/',
↪   blank=True, null=True)  # Optional PDF upload

    class Meta:
        unique_together = (('user', 'name'),)
```

D:/Programming/AI-SkillNavigator/profiles/tests.py

```
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/profiles/urls.py

```
# profiles/urls.py
from django.urls import path
from .views import update_profile_view, profile_view, delete_course,
↪   delete_internship, delete_certification, ProtectedProfileView
```

```
urlpatterns = [
    path('', profile_view, name='profile'),
    path('protected/', ProtectedProfileView.as_view(), name='protected_profile'),
    path('update', update_profile_view, name='update_profile'),
    path('delete_course/<int:course_id>/', delete_course, name='delete_course'),
    path('delete_internship/<int:internship_id>/', delete_internship,
↪   name='delete_internship'),
    path('delete_certification/<int:certification_id>/', delete_certification,
↪   name='delete_certification'),
]
```

D:/Programming/AI-SkillNavigator/profiles/views.py

```
from django.shortcuts import render, redirect
from django.contrib.auth.decorators import login_required
from .models import Profile, Course, Internship, Certification
from .forms import ProfileForm, CourseForm, InternshipForm, CertificationForm
from datetime import datetime
from django.utils import timezone
import os

from rest_framework.permissions import IsAuthenticated
from rest_framework.views import APIView
from rest_framework.response import Response

class ProtectedProfileView(APIView):
    permission_classes = [IsAuthenticated]

    def get(self, request):
        content = {'message': 'This is a protected profile view!'}
        return Response(content)

@login_required
def profile_view(request):
    try:
        profile = Profile.objects.get(user=request.user)

        # Fetch related courses, internships, and certifications
        courses = Course.objects.filter(user=request.user)
        internships = Internship.objects.filter(user=request.user)
        certifications = Certification.objects.filter(user=request.user)
        return render(request, 'profiles/profile.html', {
            'profile': profile,
            'courses': courses,
            'internships': internships,
            'certifications': certifications,
        })
    except Profile.DoesNotExist:
        return redirect('update_profile')


@login_required
def update_profile_view(request):
    try:
        profile = Profile.objects.get(user=request.user)
    except Profile.DoesNotExist:
        profile = Profile(user=request.user)  # Create a new Profile instance but
↪   don't save yet

    if request.method == 'POST':
        profile_form = ProfileForm(request.POST, instance=profile)

        # Save profile details if valid
        if profile_form.is_valid():
            profile_form.save()  # Save the profile instance

            # Process multiple courses
            i = 0
            while True:
                name = request.POST.get(f'courses[{i}][name]')
                platform = request.POST.get(f'courses[{i}][platform]')
                certificate = request.FILES.get(f'courses[{i}][certificate]')

                if name is None and platform is None and certificate is None:
                    break  # Stop if no more courses are found

                if name and platform:  # Check if name and platform are provided
                    # Check if the course already exists
                    course, created = Course.objects.get_or_create(
                        user=request.user,
                        name=name,
                        platform=platform,
                    )
                   if created:  # If a new course was created, set the certificate
                        course.certificate = certificate
                        course.save()

                i += 1

            # Process multiple internships
            j = 0
            while True:
                title = request.POST.get(f'internships[{j}][title]')
                company = request.POST.get(f'internships[{j}][company]')
                start_date = request.POST.get(f'internships[{j}][start_date]')
                end_date = request.POST.get(f'internships[{j}][end_date]')
                certificate = request.FILES.get(f'internships[{j}][certificate]')

                if title is None and company is None and start_date is None and
                 ↪ end_date is None and certificate is None:
                    break  # Stop if no more internships are found

                if title and company:  # Check if title and company are provided
                    try:
                        start_date = datetime.strptime(start_date,
 ↪   '%Y-%m-%d').date() if start_date else timezone.now().date()
                end_date = datetime.strptime(end_date, '%Y-%m-%d').date() if
 ↪   end_date else timezone.now().date()
                    except ValueError:
                       start_date = end_date = timezone.now().date()  # Use today's
 ↪   date if parsing fails

                    # Check if the internship already exists
                    internship, created = Internship.objects.get_or_create(
                        user=request.user,
                        title=title,
                        company=company,
                        start_date=start_date,
                        end_date=end_date,
                    )
                    if created:  # If a new internship was created, set the
                     ↪ certificate
                        internship.certificate = certificate
                        internship.save()

                j += 1
```

```
            # Process multiple certifications
            k = 0
            while True:
                name = request.POST.get(f'certifications[{k}][name]')
                certificate = request.FILES.get(f'certifications[{k}][certificate]')

                if name is None and certificate is None:
                    break  # Stop if no more certifications are found

                if name:  # Check if name is provided
                    # Check if the certification already exists
                    certification, created = Certification.objects.get_or_create(
                        user=request.user,
                        name=name,
                    )
                    if created:  # If a new certification was created, set the
                     ↪  certificate
                        certification.certificate = certificate
                        certification.save()

                k += 1

            return redirect('profile')  # Redirect to profile after saving

        else:
            print("Profile form errors:", profile_form.errors)  # Log errors if the
             ↪  form is not valid

    else:
        profile_form = ProfileForm(instance=profile)  # Pass the profile instance

    # Retrieve all related objects to display in the form
    courses = Course.objects.filter(user=request.user)
    internships = Internship.objects.filter(user=request.user)
    certifications = Certification.objects.filter(user=request.user)

    return render(request, 'profiles/update_profile.html', {
        'profile_form': profile_form,
        'courses': courses,
        'internships': internships,
        'certifications': certifications,
    })
```

```python
from django.http import HttpResponseRedirect
from django.urls import reverse

@login_required
def delete_course(request, course_id):
    course = Course.objects.get(id=course_id)
    if course.user == request.user:
        if course.certificate:
            file_path = course.certificate.path
            if os.path.isfile(file_path):
                os.remove(file_path)
        course.delete()
    return HttpResponseRedirect(reverse('profile'))

@login_required
def delete_internship(request, internship_id):
    internship = Internship.objects.get(id=internship_id)
    if internship.user == request.user:
        if internship.certificate:
            file_path = internship.certificate.path
            if os.path.isfile(file_path):
                os.remove(file_path)
        internship.delete()
    return HttpResponseRedirect(reverse('profile'))

@login_required
def delete_certification(request, certification_id):
    certification = Certification.objects.get(id=certification_id)
    if certification.user == request.user:
        if certification.certificate:
            file_path = certification.certificate.path
            if os.path.isfile(file_path):
                os.remove(file_path)
        certification.delete()
    return HttpResponseRedirect(reverse('profile'))
```

D:/Programming/AI-SkillNavigator/test_/admin.py

```python
# test/admin.py
from django.contrib import admin
from .models import TestTopic, UserScore
```

```python
@admin.register(TestTopic)
class TestTopicAdmin(admin.ModelAdmin):
    list_display = ('id', 'batch', 'topic_name')
    search_fields = ('topic_name',)
    list_filter = ('batch',)

@admin.register(UserScore)
class UserScoreAdmin(admin.ModelAdmin):
    list_display = ('id', 'user', 'topic', 'score', 'attempts', 'last_attempted')
    search_fields = ('user__username', 'topic__topic_name')
    list_filter = ('user', 'topic')
```

D:/Programming/AI-SkillNavigator/test_/apps.py

```python
from django.apps import AppConfig


class TestConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'test_'
```

D:/Programming/AI-SkillNavigator/test_/models.py

```python
# test/models.py
from django.db import models
from django.contrib.auth.models import User
from batch_allocation.models import Batch  # Ensure Batch model is in a separate app

class TestTopic(models.Model):
    batch = models.ForeignKey(Batch, on_delete=models.CASCADE,
 ↪  related_name="topics")
    topic_name = models.CharField(max_length=255)

    def __str__(self):
        return f"{self.topic_name} ({self.batch.name})"

class UserScore(models.Model):
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    topic = models.ForeignKey(TestTopic, on_delete=models.CASCADE)
    score = models.IntegerField(default=0)
    attempts = models.IntegerField(default=0)
    last_attempted = models.DateTimeField(auto_now=True)  # Automatically updated
↪   with each attempt
    batch = models.ForeignKey(Batch, on_delete=models.CASCADE)  # Add this line


    class Meta:
        unique_together = ('user', 'topic')  # Ensure one score per user-topic
↪   combination

    def __str__(self):
        return f"{self.user.username} - {self.topic.topic_name} - Score:
        ↪  {self.score}, Attempts: {self.attempts}"
```

D:/Programming/AI-SkillNavigator/test_/tests.py

```
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/test_/urls.py

```
# test/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('generate_report_pdf/', views.generate_report_pdf,
↪   name='generate_report_pdf'),
    path('<int:batch_id>/', views.generate_test, name='generate_test'),
    path('<int:batch_id>/<str:selected_topic>', views.generate_topic_test,
↪   name='generate_topic_test'),
    path('toppers/', views.topper_view, name='topper_page'),
    path('submit_test/', views.submit_test, name='submit_test'),
    path('success/', views.success_page, name='success_page'),
]
```

D:/Programming/AI-SkillNavigator/test_/utils.py

```
# test/utils.py
import google.generativeai as genai
import random
import json
```

```python
import re
import os
from urllib.parse import urlparse
from markdown import markdown
from bs4 import BeautifulSoup
import markdown
from dotenv import load_dotenv

load_dotenv()

# Configure API Key
api_key = os.getenv('AIzaSyDd0zliIUMWns5L-cSbCtDAU3SHZazE9vE')
genai.configure(api_key=api_key)

def generate_mcq_question(topic):
    """
    Generates a multiple-choice question using an AI model based on a given topic.

    Parameters:
    - topic (str): The topic for which to generate a question.

    Returns:
    - dict: A dictionary containing the question, options, and correct answer.
    """
    # Structured prompt for consistent formatting
    ask = (
        f"Generate a multiple-choice question on the topic '{topic}'. "
        "Format answer as:\n\n"
        "Question: [Your question]\n"
        "A) Option 1\n"
        "B) Option 2\n"
        "C) Option 3\n"
        "D) Option 4\n\n"
        "Correct Answer: [Letter of correct option]"
    )

    # Call the AI model to generate the question
    try:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(ask)
        response_text = response.text
    except Exception as e:
        print(f"Error generating question: {e}")
        return None

    # Parse the AI response for structured data extraction
    def parse_response(response_text):
        lines = response_text.strip().split('\n')

        # Extract the question
        question_line = next((line for line in lines if
↪   line.startswith("Question:")), None)
       question = question_line.split(":", 1)[1].strip() if question_line else None

        # Extract options
        options = {}
        for line in lines:
            if line.startswith(("A)", "B)", "C)", "D)")):
                key, value = line.split(")", 1)
                options[key.strip()] = value.strip()

        # Extract correct answer
        correct_answer_line = next((line for line in lines if
↪   line.startswith("Correct Answer:")), None)
        correct_answer = correct_answer_line.split(":", 1)[1].strip() if
↪   correct_answer_line else None

        return {
            "question": question,
            "options": options,
            "correct_answer": correct_answer
        }

    # Parse and return structured data
    parsed_data = parse_response(response_text)
    return parsed_data


def is_valid_url(url):
    """Check if the URL is valid."""
    try:
        result = urlparse(url)
        return all([result.scheme, result.netloc])
    except ValueError:
        return False
```

```
def Ai_course_recom(incorrect_topics):
    python = [
        'https://www.udemy.com/course/data-engineering-101-the-beginners-guide/',
        'https://www.udemy.com/course/the-complete-sql-bootcamp/',
        'https://www.udemy.com/course/data-warehouse-the-ultimate-guide/',
        'https://www.udemy.com/course/etl-developer-mysql-data-migration-ms-sql-
         ↪  server-ssis/',
        'https://www.udemy.com/course/hands-on-hadoop-masterclass-tame-the-big-
         ↪  data/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/learn-data-lake-
         ↪  fundamentals/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/apache-spark-programming-in-python-for-
         ↪  beginners/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/the-complete-hands-on-course-to-master-
         ↪  apache-airflow/',
        'https://www.udemy.com/course/aws-data-engineer/'
    ]
    net = [
        'https://www.udemy.com/course/asp-net-core-true-ultimate-guide-real-
         ↪  project/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/complete-aspnet-core-21-
         ↪  course/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/build-rest-apis-with-aspnet-core-web-api-
         ↪  entity-framework/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/csharp-oops-mvc-asp-dotnet-core-webapi-sql-
         ↪  questions-mock-interviews/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/aspnet-mvc-course-aspnet-
         ↪  core/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/complete-aspnet-core-31-and-entity-
         ↪  framework-development/?couponCode=KEEPLEARNING'
    ]
    java = [
        'https://www.udemy.com/course/java-the-complete-java-developer-
         ↪  course/?couponCode=KEEPLEARNING',
       'https://www.udemy.com/course/java-se-programming/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/java-programming-tutorial-for-
         ↪  beginners/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/the-complete-java-development-
         ↪  bootcamp/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/full-stack-java-developer-
         ↪  java/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/java-programming-a-comprehensive-bootcamp-
         ↪  from-zero-to-hero/?couponCode=KEEPLEARNING',
        'https://www.udemy.com/course/spring-5-with-spring-boot-
         ↪  2/?couponCode=KEEPLEARNING'
    ]

   # Embed the predefined arrays directly into the prompt for better AI guidance
    prompt = (
        f"I have identified the following topics I am weak in: {',
         ↪  '.join(incorrect_topics)}. "
        "Here are the available course links: "
        f"Python courses: {', '.join(python)}; "
        f".NET courses: {', '.join(net)}; "
        f"Java courses: {', '.join(java)}. "
        "Based on these, please select and recommend the most relevant courses
         ↪  according to the weak topics mentioned. "
        "Return the result in JSON format with these fields: course_name,
         ↪  course_link, platform, and course_image_link. "
        "Ensure course_image_link is a valid URL."
    )

    # The rest of the function
    while incorrect_topics:
        model = genai.GenerativeModel("gemini-1.5-flash")
        response = model.generate_content(prompt)
        response_text = response.text

        # Use regex to extract JSON from the response text
        json_match = re.search(r'\[.*\]', response_text, re.DOTALL)
        if json_match:
            json_str = json_match.group(0)  # Get the matched JSON string
            try:
                courses = json.loads(json_str)

                # Validate the image links
                for course in courses:
                    image_link = course.get("course_image_link")
                    if not is_valid_url(image_link):
                        # Set a standard image link if the provided link is invalid
                        course["course_image_link"] =
↪   "https://example.com/standard-image.jpg"  # Replace with your default image URL

                return courses
            except json.JSONDecodeError:
                print("Failed to decode JSON response. Response text:",
                 ↪  response_text)
                return []
            except Exception as e:
                print(f"An error occurred: {e}")
                return []

    return []



# List of available course image links
image_links = [
    "https://img-c.udemycdn.com/course/480x270/567828_67d0.jpg",
    "https://img-c.udemycdn.com/course/480x270/2776760_f176_10.jpg",
    "https://img-b.udemycdn.com/course/480x270/903744_8eb2.jpg",
    "https://img-c.udemycdn.com/course/480x270/543600_64d1_4.jpg",
    "https://img-c.udemycdn.com/course/480x270/629302_8a2d_2.jpg",
    "https://img-c.udemycdn.com/course/480x270/692188_9da7_34.jpg",
    "https://img-c.udemycdn.com/course/480x270/2473048_8255_5.jpg",
    "https://img-c.udemycdn.com/course/480x270/903378_f32d_7.jpg",
    "https://img-c.udemycdn.com/course/480x270/822444_a6db.jpg",
    "https://img-c.udemycdn.com/course/480x270/836376_8b97_4.jpg",
    "https://img-c.udemycdn.com/course/480x270/1350984_2355_6.jpg",
    "https://img-c.udemycdn.com/course/480x270/1340588_e1b6_4.jpg",
    "https://img-c.udemycdn.com/course/480x270/1386294_cf10_3.jpg",

     ↪  "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
     ↪  full-stack-cloud-developer/XDP~SPECIALIZATION!~ibm-full-stack-cloud-
     ↪  developer.jpeg",
    "https://img-c.udemycdn.com/course/480x270/1565838_e54e_18.jpg",
    "https://img-c.udemycdn.com/course/480x270/1646980_23f7_3.jpg",
    "https://img-b.udemycdn.com/course/480x270/2306676_57ba_2.jpg",
    "https://via.placeholder.com/300x200",
    "https://img-c.udemycdn.com/course/480x270/1672410_9ff1_5.jpg",
    "https://img-c.udemycdn.com/course/480x270/3716888_5054.jpg",
    "https://img-b.udemycdn.com/course/480x270/959700_8bd2_12.jpg",
    "https://img-c.udemycdn.com/course/480x270/382002_5d4a_3.jpg",
    "https://s.udemycdn.com/meta/default-meta-image-v2.png",
    "https://img-c.udemycdn.com/course/480x270/806922_6310_3.jpg",
    "https://s.udemycdn.com/meta/default-meta-image-v2.png",
"https://img-b.udemycdn.com/course/480x270/383576_fd27_4.jpg",
"https://img-c.udemycdn.com/course/480x270/356030_0209_6.jpg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
↪ programming/XDP~SPECIALIZATION!~java-programming.jpeg",
"https://img-c.udemycdn.com/course/480x270/533682_c10c_4.jpg",
"https://img-c.udemycdn.com/course/480x270/1535678_0ce9_7.jpg",
"https://img-c.udemycdn.com/course/480x270/358540_d06b_16.jpg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
↪ programming/XDP~SPECIALIZATION!~java-programming.jpeg",
"https://img-c.udemycdn.com/course/480x270/533682_c10c_4.jpg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~object-
↪ oriented-java/XDP~COURSE!~object-oriented-java.jpeg",
"https://img-c.udemycdn.com/course/480x270/1656228_5278_5.jpg",
"https://img-c.udemycdn.com/course/480x270/1217894_e8cc_4.jpg",
"https://img-c.udemycdn.com/course/480x270/1419186_5b21_2.jpg",
"https://img-c.udemycdn.com/course/480x270/1352468_3d97_8.jpg",
"https://img-c.udemycdn.com/course/480x270/2208130_c37b_6.jpg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
↪ data-science/XDP~SPECIALIZATION!~ibm-data-science.jpeg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
↪ data-science/XDP~SPECIALIZATION!~jhu-data-science.jpeg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~machine-
↪ learning/XDP~COURSE!~machine-learning.jpeg",
"https://img-c.udemycdn.com/course/480x270/821726_8071.jpg",
"https://img-b.udemycdn.com/course/480x270/903744_8eb2.jpg",
"https://img-c.udemycdn.com/course/480x270/513244_b831_4.jpg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
↪ learning/XDP~SPECIALIZATION!~deep-learning.jpeg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~r-
↪ programming/XDP~COURSE!~r-programming.jpeg",

↪ "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~data-
↪ analysis-with-python/XDP~COURSE!~data-analysis-with-python.jpeg",
"https://img-c.udemycdn.com/course/480x270/1298780_731f_4.jpg",
     ↪  "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~SPECIALIZATION!~
     ↪  science-python/XDP~SPECIALIZATION!~data-science-python.jpeg",

     ↪  "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~sql-
     ↪  for-data-science/XDP~COURSE!~sql-for-data-science.jpeg",

     ↪  "https://s3.amazonaws.com/coursera_assets/meta_images/generated/XDP/XDP~COURSE!~machine-
     ↪  learning-with-python/XDP~COURSE!~machine-learning-with-python.jpeg",
    "https://www.cdmi.in/courses@2x/python-training-institute.webp",
    "https://www.clariwell.in/resources/best-java-certification-course-top-
     ↪  training-institute-in-pune.webp",
]


def get_unique_image_links(num_links):
    # Shuffle the image links randomly
    random.shuffle(image_links)

    # Select the first num_links from the shuffled list
    unique_links = image_links[:num_links]
    return unique_links


def generate_feedback(incorrect_topics):
    # Create a structured prompt based on the number of topics
    if len(incorrect_topics) > 2:
        prompt_parts = []
        for topic in incorrect_topics:
            prompt_parts.append(f"Provide constructive feedback for the topic:
 ↪  {topic}. Suggest study techniques, useful resources, and practical exercises
 ↪  to deepen understanding.")
        prompt = " ".join(prompt_parts)
    else:
        topic_list = ', '.join(incorrect_topics)  # Join topics in a readable format
        prompt = (
            f"The student is experiencing challenges in these topics: {topic_list}.
             ↪  "
            "Provide constructive feedback to help them improve. "
            "Include a couple of paragraphs suggesting study techniques, useful
             ↪  resources, and practical exercises "
            "to deepen their understanding."
        )
```

```python
    # Generate content using the Gemini model
    model = genai.GenerativeModel("gemini-1.5-flash")
    response = model.generate_content(prompt)

    # Convert response to structured Markdown for easier HTML processing later
    response_md = f"## Feedback on Improvement\n\n{response.text}"
    return response.text


def md_to_html(md):
    # Convert Markdown to HTML
    html_content = markdown(md)

    # Parse HTML for additional styling or formatting with BeautifulSoup
    soup = BeautifulSoup(html_content, 'html.parser')

    # Optional: Add custom styling
    for heading in soup.find_all(['h1', 'h2', 'h3']):
        heading['class'] = 'pdf-heading'
    for paragraph in soup.find_all('p'):
        paragraph['class'] = 'pdf-paragraph'

    return soup.prettify()  # Convert to a formatted HTML string


def md_to_text(md):
    # Convert Markdown to HTML
    html = markdown.markdown(md)

    # Parse the HTML with BeautifulSoup
    soup = BeautifulSoup(html, features='html.parser')

    # Optionally add custom styles or classes
    for heading in soup.find_all(['h1', 'h2', 'h3', 'h4', 'h5', 'h6']):
        heading['class'] = 'my-heading-class'  # Add a custom CSS class for styling

    # Return the prettified HTML
    return soup.prettify()  # Use prettify() for better formatting (optional)
```

D:/Programming/AI-SkillNavigator/test_/views.py

```python
# test/views.py
```

```
from django.shortcuts import render, get_object_or_404, redirect
from django.http import HttpResponse
from django.contrib.auth.decorators import login_required

from . models import TestTopic, UserScore
from .utils import generate_mcq_question, Ai_course_recom, get_unique_image_links,
 ↪  generate_feedback, md_to_html, md_to_text
from batch_allocation.models import Batch

from reportlab.pdfgen import canvas
from reportlab.lib.pagesizes import A4
from reportlab.platypus import Paragraph
from reportlab.lib.styles import getSampleStyleSheet
from datetime import datetime
import random
import logging

@login_required
def generate_test(request, batch_id):
    # Retrieve the batch and its topics
    batch = Batch.objects.get(id=batch_id)
    topics = list(batch.topics.all())

    # Select 10 random topics
    selected_topics = random.sample(topics, min(10, len(topics)))

    # Generate questions for each topic
    questions = []
    for topic in selected_topics:
        generated_question = generate_mcq_question(topic.topic_name)
        if generated_question:
            correct_answer = generated_question.get("correct_answer")
            if correct_answer is not None:
                questions.append({
                    "topic": topic.topic_name,
                    "question": generated_question["question"],
                    "options": generated_question["options"],  # A dictionary of
                     ↪  options (A, B, C, D)
                    "correct_answer": correct_answer.strip(')')  # Store without
                     ↪  closing parenthesis
                })
            else:
                # Log or handle the case where the correct answer is None
                print(f"Warning: No correct answer generated for topic:
                 ↪  {topic.topic_name}")
        else:
            print(f"Warning: No question generated for topic: {topic.topic_name}")

    request.session['questions'] = questions  # Store in session for score
↪   calculation
    request.session['test_type'] = 'generate_test'  # Set flag for submit_test

    # Render questions on the test page
    return render(request, 'test_/generate_test.html', {
        'batch': batch,
        'questions': questions,
    })


def generate_topic_test(request, batch_id, selected_topic):
    """
    Generates a test with 10 questions on a single specified topic.
    """
    batch = get_object_or_404(Batch, id=batch_id)
    questions = []
    max_attempts, count, attempts = 20, 0, 0

    logging.debug(f"Starting question generation for topic: {selected_topic}")

    while count < 10 and attempts < max_attempts:
        generated_question = generate_mcq_question(selected_topic)
        attempts += 1

        if generated_question and generated_question.get("correct_answer"):
            questions.append({
                "topic": selected_topic,
                "question": generated_question["question"],
                "options": generated_question["options"],
                "correct_answer": generated_question["correct_answer"].strip(')')
            })
            count += 1
            logging.debug(f"Question {count} added for topic {selected_topic}")
        else:
            logging.warning(f"Attempt {attempts} failed to generate a valid
↪   question for topic: {selected_topic}")
```

```
    if count < 10:
        logging.warning(f"Only {count} questions generated after {attempts}
↪   attempts for topic {selected_topic}")

    # Save questions in the session
    request.session['questions'] = questions
    request.session['test_type'] = 'generate_topic_test'  # Set flag for
↪   submit_test

    return render(request, 'test_/generate_topic_test.html', {
        'batch': batch,
        'selected_topic' : selected_topic,
        'questions': questions,
    })


@login_required
def submit_test(request):
    if request.method == 'POST':
        questions = request.session.get('questions', [])
        test_type = request.session.get('test_type')  # Get the test type flag
        score = 0
        incorrect_questions = []  # Store incorrectly answered questions and
↪   correct answers
        incorrect_topics = []  # Track incorrect answers if needed
        total_questions = len(questions)  # Track total questions

        batch_id = request.POST.get('batch_id')  # Ensure this is sent in the POST
↪   request
        print(batch_id)
        # batch_id = int(batch_id)
        for index, question in enumerate(questions, start=1):
            submitted_answer = request.POST.get(f'answer_{index}')
            correct_answer = question.get('correct_answer')
            correct_answer_text = question["options"].get(correct_answer)  #
↪   Retrieve answer text
            correct_answer_text = correct_answer.lower() + ') ' +
↪   str(correct_answer_text)


            if submitted_answer and correct_answer:
                if submitted_answer[0] == correct_answer[0]:  # Correct answer
                    score += 1
                else:
                    # Track topic if the answer is incorrect
                    # Add incorrect question details
                    incorrect_questions.append({
                        "topic": question["topic"],
                        "question": question["question"],
                        "correct_answer": correct_answer_text,
                    })
                    # if test_type == 'generate_test':  # Only track for
                     ↪  'generate_test'
                    incorrect_topics.append(question["topic"])

        # Process score and attempts for each topic only if called from
         ↪  'generate_topic_test'
        if test_type == 'generate_topic_test':
            unique_topics = set(question["topic"] for question in questions)
            for topic_name in unique_topics:
               logging.debug(f"Attempting to retrieve TestTopic for: {topic_name}")
                try:
                    topic = TestTopic.objects.get(topic_name=topic_name)

                    # Retrieve or create the UserScore entry for the topic
                    user_score, created =
↪   UserScore.objects.get_or_create(user=request.user, topic=topic,
↪   batch_id=batch_id)

                    # Update the attempt count only once per test
                    if created:
                        user_score.attempts = 1
                    else:
                       user_score.attempts += 1  # Increment the attempt count for
↪   this topic

                   # Update score only if the new score is higher than the previous
                    ↪  score
                    user_score.score = max(user_score.score, score)
                    user_score.save()

                except TestTopic.DoesNotExist:
                    logging.error(f"TestTopic with name {topic_name} does not
↪   exist.")
                    # Handle the case where the topic does not exist
                    continue  # Skip to the next topic or handle it as needed
        # Store the score and incorrect topics in the session, then clear questions
        request.session['score'] = score
        request.session['total_questions'] = total_questions  # Store total
↪   questions in session
        request.session['incorrect_questions'] = incorrect_questions


        request.session['incorrect_topics'] = list(set(incorrect_topics))  # Save
↪   incorrect topics if needed

        # Redirect to the success page
        return redirect('success_page')


@login_required
def success_page(request):
    user_scores = UserScore.objects.filter(user=request.user)
    test_type = request.session.pop('test_type', None)  # Get the test type from
↪   the session
    score = request.session.pop('score', 0)  # Retrieve and remove score from
↪   session
    total_questions = request.session.pop('total_questions', 0)  # Retrieve and
↪   remove total questions from session
    incorrect_topics = request.session.pop('incorrect_topics', [])  # Retrieve and
↪   remove incorrect topics from session
    recommended_courses = Ai_course_recom(incorrect_topics)
    incorrect_questions = request.session.pop('incorrect_questions', [])
    selected_links = get_unique_image_links(len(recommended_courses))

    # Store values back in the session for the PDF generation
    request.session['pdf_score'] = score
    request.session['pdf_total_questions'] = total_questions
    request.session['pdf_incorrect_topics'] = incorrect_topics
    request.session['pdf_test_type'] = test_type

    # Combine recommended courses and image links
    combined_courses = [
        {
            'course': course,
            'image_link': selected_links[i]
        }
        for i, course in enumerate(recommended_courses)
    ]

    return render(request, 'test_/success_page.html', {
        'score': score,
        'total_questions': total_questions,  # Pass total questions to the template
        'test_type': test_type,  # Pass test type to the template
        'user_scores': user_scores,
        'incorrect_topics': incorrect_topics,  # Pass incorrect topics to the
        ↪  template
        'incorrect_questions': incorrect_questions,
        'recommended_courses': recommended_courses,  # Pass the recommended courses
        ↪  to the template
       'selected_links': selected_links,  # Pass the selected links to the template
        'combined_courses': combined_courses,  # Pass the combined courses to the
        ↪  template


    })


def topper_view(request):
    selected_batch_id = request.GET.get('batch')
    selected_topic_id = request.GET.get('topic')

    # Fetch batches and topics for dropdown filters
    batches = Batch.objects.all()
    topics = TestTopic.objects.all()

    # Filter by selected batch and topic, if specified
    if selected_batch_id:
        batch = Batch.objects.get(id=selected_batch_id)
        topics = topics.filter(batch=batch)  # Only topics for the selected batch
        scores = UserScore.objects.filter(topic__batch=batch)
    else:
        scores = UserScore.objects.all()

    if selected_topic_id:
        topic = TestTopic.objects.get(id=selected_topic_id)
        scores = scores.filter(topic=topic)

    # Get top 10 scores per batch, with topic info
    top_scorers = []
    if selected_batch_id:
        scores = scores.order_by('-score', 'attempts')[:10]
        for score in scores:
            top_scorers.append({
                'user': score.user.username,
                'score': score.score,
                'attempts': score.attempts,
                'topic': score.topic.topic_name,
                'batch': score.topic.batch.name,
                'last_attempted': score.last_attempted,
            })
    else:
        for batch in batches:
            batch_scores = scores.filter(topic__batch=batch).order_by('-score',
 ↪  'attempts')[:10]
            top_scorers += [{
                'user': score.user.username,
                'score': score.score,
                'attempts': score.attempts,
                'topic': score.topic.topic_name,
                'batch': batch.name,
                'last_attempted': score.last_attempted,
            } for score in batch_scores]

    return render(request, 'test_/topper_page.html', {
        'batches': batches,
        'topics': topics,
        'top_scorers': top_scorers,
        'selected_batch_id': selected_batch_id,
        'selected_topic_id': selected_topic_id,
    })
```

```
from reportlab.platypus import SimpleDocTemplate, Paragraph, Spacer, Table,
↪  TableStyle, Image
from reportlab.lib.styles import getSampleStyleSheet, ParagraphStyle
from reportlab.lib.units import inch
from reportlab.lib import colors
from reportlab.lib.pagesizes import A4
from reportlab.lib.utils import ImageReader

import matplotlib
matplotlib.use('Agg')  # Use a non-GUI backend
import matplotlib.pyplot as plt
from django.contrib import messages
import io
import numpy as np
from datetime import datetime
from django.http import HttpResponse
from django.contrib.auth.decorators import login_required

@login_required
def generate_report_pdf(request):
    try:
        response = HttpResponse(content_type='application/pdf')
        response['Content-Disposition'] = f'inline;
↪  filename="{request.user.username}_report.pdf"'
```

```
        # Initialize PDF structure with SimpleDocTemplate
        pdf = SimpleDocTemplate(response, pagesize=A4)
        elements = []
        styles = getSampleStyleSheet()

        # Custom styles
        title_style = ParagraphStyle(name='TitleStyle', fontSize=18, leading=22,
    ↪   spaceAfter=10, alignment=1)
        section_header_style = ParagraphStyle(name='SectionHeader', fontSize=14,
    ↪   leading=16, spaceAfter=8, textColor=colors.blue)
        body_text_style = styles['BodyText']

        username = request.user.username if request.user.is_authenticated else
    ↪   "Guest"

        # Add content to the PDF
        add_title_and_date(elements, request.user.username, title_style,
    ↪   body_text_style)
        add_performance_summary(elements, request, section_header_style,
    ↪   body_text_style)
        add_topic_analysis(elements, request, section_header_style,
    ↪   body_text_style)
        add_incorrect_questions_table(elements, request, section_header_style,
    ↪   body_text_style)
        add_recommended_next_steps(elements, request, section_header_style,
    ↪   body_text_style)

        # Insert Charts into the PDF
        add_charts(elements, request)

        # Build the PDF
        pdf.build(elements)
        return response

    except Exception as e:
        # Handle errors and add a message to be shown on the website
        # messages.error(request, f"An error occurred while generating the PDF:
         ↪  {str(e)}")
        # Return a simple HTML response to indicate an error occurred
        return HttpResponse("<h1>You have reloaded the page</h1><h2>An error
         ↪  occurred while generating the PDF. <br> Please write the test again and
         ↪  generate the report</h2>")
```

```
def add_title_and_date(elements, username, title_style, body_text_style):
    elements.append(Paragraph(f"Student Report for {username}", title_style))
    elements.append(Paragraph(f"Date: {datetime.now().strftime('%Y-%m-%d
↪   %H:%M')}", body_text_style))
    elements.append(Spacer(1, 0.2 * inch))


def add_performance_summary(elements, request, section_header_style,
↪   body_text_style):
    score = request.session.get('pdf_score', 0)
    total_questions = request.session.get('pdf_total_questions', 0)
    elements.append(Paragraph("Performance Summary", section_header_style))
    elements.append(Paragraph(f"Score: {score} out of {total_questions}",
↪   body_text_style))
    performance_percentage = (score / total_questions) * 100 if total_questions > 0
↪   else 0
    elements.append(Paragraph(f"Overall Performance:
↪   {performance_percentage:.2f}%", body_text_style))
    elements.append(Spacer(1, 0.2 * inch))


def add_topic_analysis(elements, request, section_header_style, body_text_style):
    incorrect_topics = request.session.get('pdf_incorrect_topics', [])

    # Append section header
    elements.append(Paragraph("Topic-wise Analysis", section_header_style))

    if incorrect_topics:
        # Generate formatted feedback
        feedback_md = generate_feedback(incorrect_topics)
        feedback_text = md_to_text(feedback_md)  # Convert Markdown to HTML

        # Use HTML paragraph formatting for better structure
        elements.append(Paragraph(feedback_text, body_text_style))
        elements.append(Spacer(1, 0.2 * inch))
    else:
        elements.append(Paragraph("All topics were answered correctly.",
↪   body_text_style))
```

```
def add_incorrect_questions_table(elements, request, section_header_style,
↪  body_text_style):
    incorrect_questions = request.session.get('incorrect_questions', [])
    if incorrect_questions:
        elements.append(Paragraph("Detailed Incorrect Questions",
↪  section_header_style))
        table_data = [["Topic", "Question", "Correct Answer"]]
        for question in incorrect_questions:
            table_data.append([question["topic"], question["question"],
↪  question["correct_answer"]])

        # Create and style the table
        table = Table(table_data, colWidths=[2 * inch, 3 * inch, 2 * inch])
        table.setStyle(TableStyle([
            ('BACKGROUND', (0, 0), (-1, 0), colors.lightgrey),
            ('TEXTCOLOR', (0, 0), (-1, 0), colors.whitesmoke),
            ('ALIGN', (0, 0), (-1, -1), 'LEFT'),
            ('FONTNAME', (0, 0), (-1, 0), 'Helvetica-Bold'),
            ('FONTSIZE', (0, 0), (-1, 0), 12),
            ('BOTTOMPADDING', (0, 0), (-1, 0), 12),
            ('BACKGROUND', (0, 1), (-1, -1), colors.beige),
            ('GRID', (0, 0), (-1, -1), 0.5, colors.grey)
        ]))
        elements.append(table)
        elements.append(Spacer(1, 0.2 * inch))


def add_recommended_next_steps(elements, request, section_header_style,
↪  body_text_style):
    incorrect_topics = request.session.get('pdf_incorrect_topics', [])
    elements.append(Paragraph("Recommended Next Steps", section_header_style))
    if incorrect_topics:
        for topic in incorrect_topics:
            elements.append(Paragraph(f"- Focus on additional resources and
↪  practice questions for {topic}.", body_text_style))
        elements.append(Paragraph("Consider revisiting these topics with additional
↪  tutorials or practice tests.", body_text_style))
    else:
        elements.append(Paragraph("Excellent work! Continue to advance in other
↪  topics or explore new areas of study.", body_text_style))


def add_charts(elements, request):
    section_header_style = ParagraphStyle(name='SectionHeader', fontSize=14,
↪   leading=16, spaceAfter=8, textColor=colors.blue)
    score = request.session.get('pdf_score', 0)
    total_questions = request.session.get('pdf_total_questions', 0)
    incorrect_topics = request.session.get('pdf_incorrect_topics', [])

    # Score distribution chart
    elements.append(Paragraph("Score Distribution", section_header_style))
    score_dist_img = create_score_distribution_chart(score, total_questions)
    elements.append(Image(score_dist_img, width=5 * inch, height=3 * inch))
    elements.append(Spacer(1, 0.2 * inch))

    # Topic performance chart
    elements.append(Paragraph("Topic Performance", section_header_style))
    topic_perf_img = create_topic_performance_chart(total_questions,
↪   incorrect_topics)
    elements.append(Image(topic_perf_img, width=5 * inch, height=3 * inch))
    elements.append(Spacer(1, 0.2 * inch))

    # Incorrect topics frequency chart

    test_type = request.session.get('pdf_test_type')

    if test_type == 'generate_test':
        elements.append(Paragraph("Incorrect Topics Frequency",
↪   section_header_style))
        incorrect_topics_img = create_incorrect_topics_chart(request)
        elements.append(Image(incorrect_topics_img, width=5 * inch, height=3 *
↪   inch))
        elements.append(Spacer(1, 0.5 * inch))


def create_score_distribution_chart(score, total_questions):
    labels = ['Correct', 'Incorrect']
    values = [score, total_questions - score]
    colors = ['#4CAF50', '#FF6F61']

    fig, ax = plt.subplots()
    ax.bar(labels, values, color=colors)
    ax.set_title('Score Distribution')
    ax.set_ylabel('Count')

    img_buffer = io.BytesIO()
    plt.savefig(img_buffer, format='PNG')
    plt.close()
    img_buffer.seek(0)
    return img_buffer


def create_topic_performance_chart(total_questions, incorrect_topics):
    correct_topics = total_questions - len(incorrect_topics)
    incorrect_topics_count = len(incorrect_topics)
    sizes = [correct_topics, incorrect_topics_count]
    labels = ['Correct Topics', 'Incorrect Topics']
    colors = ['#8BC34A', '#FF5252']

    fig, ax = plt.subplots()
    ax.pie(sizes, labels=labels, autopct='%1.1f%%', startangle=140, colors=colors)
    ax.set_title('Topic Performance')

    img_buffer = io.BytesIO()
    plt.savefig(img_buffer, format='PNG')
    plt.close()
    img_buffer.seek(0)
    return img_buffer


def create_incorrect_topics_chart(incorrect_questions):
    topic_names = [q['topic'] for q in incorrect_questions]
    unique_topics, counts = np.unique(topic_names, return_counts=True)

    fig, ax = plt.subplots()
    ax.bar(unique_topics, counts, color='#FF7043')
    ax.set_title('Incorrect Topics Frequency')
    ax.set_xlabel('Topics')
    ax.set_ylabel('Mistakes Count')

    img_buffer = io.BytesIO()
    plt.savefig(img_buffer, format='PNG')
    plt.close()
    img_buffer.seek(0)
    return img_buffer
```

D:/Programming/AI-SkillNavigator/visual/admin.py

```
from django.contrib import admin
```

```python
# Register your models here.
```

D:/Programming/AI-SkillNavigator/visual/apps.py

```python
from django.apps import AppConfig


class VisualConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'visual'
```

D:/Programming/AI-SkillNavigator/visual/models.py

```python
from django.db import models

# Create your models here.
```

D:/Programming/AI-SkillNavigator/visual/tests.py

```python
from django.test import TestCase

# Create your tests here.
```

D:/Programming/AI-SkillNavigator/visual/urls.py

```python
# visualization/urls.py
from django.urls import path
from . import views

urlpatterns = [
    path('results/', views.results_view, name='results'),
    path('user_score_visualization/', views.user_score_visualization,
 ↪  name='user_score_visualization'),

]
```

D:/Programming/AI-SkillNavigator/visual/views.py

```python
# visualization/views.py
```

```
from django.shortcuts import render
from test_.models import UserScore
from django.contrib.auth.decorators import login_required
import json


@login_required
def results_view(request):
    # Logic to fetch user scores and prepare data for visualization
    return render(request, 'visual/results.html', {
        # Pass necessary data for visualization
    })

@login_required
def user_score_visualization(request):
    # Fetch user scores for the logged-in user
    user_scores = UserScore.objects.filter(user=request.user)

    # Prepare data for the charts
    topics = [score.topic.topic_name for score in user_scores]
    scores = [score.score for score in user_scores]
    attempts = [score.attempts for score in user_scores]
    last_attempted_dates = [score.last_attempted.strftime("%Y-%m-%d") for score in
↪   user_scores]

    context = {
        'topics': json.dumps(topics),          # Convert data to JSON for JavaScript
         'scores': json.dumps(scores),
         'attempts': json.dumps(attempts),
         'last_attempted_dates': json.dumps(last_attempted_dates),
    }

    return render(request, 'visual/user_score_visualization.html', context)
```