Folder: autodocai

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: asgi.py

Documented code for asgi.py:

"""

ASGI config for autodocai 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/4.2/howto/deployment/asgi/

"""

```
import os

from django.core.asgi import get_asgi_application

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

application = get_asgi_application()
```

File: settings.py

Documented code for settings.py:

"""

Django settings for autodocai project.

Generated by 'django-admin startproject' using Django 4.2.10.

```
For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""

from pathlib import Path

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



import os
STATIC_URL = '/static/'
STATICFILES_DIRS = [
    os.path.join(BASE_DIR, 'static/'),
]
# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-*lgz9fh=1^u6uf+e-v63%fbylb3q%-$15&%dj*yo52!@0_3ae('

# 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',
    'codereader',
    'gemini',
]

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',

]

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

```
        },
    },
]

WSGI_APPLICATION = 'autodocai.wsgi.application'

ROOT_URLCONF = 'autodocai.urls'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

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



# Password validation
# https://docs.djangoproject.com/en/4.2/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/4.2/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/4.2/howto/static-files/


# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'
```

File: urls.py

Documented code for urls.py:

```
"""
URL configuration for autodocai project.

The `urlpatterns` list routes URLs to views. For more information please see:
```

```
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', (include('codereader.urls'))),
    path('', (include('gemini.urls'))),
]
# Serve static files during development
if settings.DEBUG:
    urlpatterns += static(settings.STATIC_URL, document_root=settings.STATIC_ROOT)

File: wsgi.py
Documented code for wsgi.py:
"""
WSGI config for autodocai project.

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

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

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

application = get_wsgi_application()

Folder: codereader
File: __init__.py
Documented code for __init__.py:

Folder: __pycache__
File: admin.py
Documented code for admin.py:
from django.contrib import admin

Register your models here.

File: apps.py
Documented code for apps.py:
from django.apps import AppConfig

class CodereaderConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'

```
    name = 'codereader'
```

File: forms.py

Documented code for forms.py:

Folder: migrations

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: models.py

Documented code for models.py:

```
from django.db import models

# Create your models here.
```

File: tests.py

Documented code for tests.py:

```
from django.test import TestCase

# Create your tests here.
```

File: urls.py

Documented code for urls.py:

```
from django.contrib import admin
from django.urls import path, include
from .views import generate_pdf, profile_analysis, index, user_signin
```

```
urlpatterns = [
    path('generate_pdf', generate_pdf, name='generate_pdf'),
    path('profile_analysis', profile_analysis, name='profile_analysis'),
    path('', index, name='index'),
    path('user_signin', user_signin, name='user_signin')
]
```

File: views.py

Documented code for views.py:

```
from django.shortcuts import render
from django.http import HttpResponse
import os
import requests
from fpdf import FPDF
import requests
from concurrent.futures import ThreadPoolExecutor
from collections import defaultdict

def fetch_repositories(username):
    """
    Fetches repositories for the given GitHub username.

    Args:
    - username (str): The GitHub username.

    Returns:
    - repositories (list): A list of dictionaries containing repository details.
    """
    url = f"https://api.github.com/users/{username}/repos"
    response = requests.get(url)
    if response.status_code == 200:
```

```
        return response.json()
    else:
        print(f"Error {response.status_code} occurred while fetching repositories.")
        return []

# def select_repository(repositories):
#     """
#     Allows the user to select a repository from the list of repositories.

#     Args:
#     - repositories (list): A list of dictionaries containing repository details.

#     Returns:
#     - selected_repo (dict): The selected repository.
#     """
#     print("Select a repository:")
#     for idx, repo in enumerate(repositories, 1):
#         print(f"{idx}: {repo['name']}")
#     repo_idx = int(input("Enter the repository number: ")) - 1
#     return repositories[repo_idx]

def fetch_contents(url):
    """
    Fetches the contents (files and directories) from the provided URL.

    Args:
    - url (str): The URL to fetch contents from.

    Returns:
    - contents (list): A list of dictionaries containing file/folder details.
    """
    response = requests.get(url)
    if response.status_code == 200:
```

```
        return response.json()
    else:
        print(f"Error {response.status_code} occurred while fetching contents.")
        return []

def visualize_structure(contents, username, repo_name):
    """
    Visualizes the folder structure recursively and documents code for files.

    Args:
    - contents (list): A list of dictionaries containing file/folder details.
    - username (str): The GitHub username.
    - repo_name (str): The repository name.
    """
    result = ""
    for item in contents:
        if item['type'] == 'dir':
            result += f"Folder: {item['name']}\n"
            subdir_contents = fetch_contents(item['url'])
            result += visualize_structure(subdir_contents, username, repo_name)
        else:
            filename = item['name']
            if filename.endswith(('.py', '.dart', '.html')):
                raw_url = item['download_url']
                code = fetch_code(raw_url)
                result += f"File: {filename}\n"
                result += f"Documented code for {filename}:\n{code}\n\n"
    return result

def fetch_code(raw_url):
    """
    Fetches and documents code for the specified file.
```

```
    Args:
    - raw_url (str): The raw URL of the file.

    Returns:
    - code (str): The documented code.
    """
    response = requests.get(raw_url)
    if response.status_code == 200:
        return response.text
    else:
        print(f"Error {response.status_code} occurred while fetching code.")
        return None

def generate_pdf(request):
    if request.method == 'POST':
        username = request.POST.get('username')
        repositories = fetch_repositories(username)

        if repositories:
            selected_repo = request.POST.get('selected_repo')
            repo_name = selected_repo.split('/')[-1]  # Extract the repository name from the URL

            contents = fetch_contents(selected_repo)
            code = visualize_structure(contents, username, repo_name)

            # Construct the PDF filename
            pdf_filename = f"{username}_{repo_name}_code_documentation.pdf"

            # Call the function to convert content to PDF
            convert_txt_to_pdf(code, pdf_filename)

            return HttpResponse(f"PDF '{pdf_filename}' generated successfully.")
        else:
```

```
            return HttpResponse("No repositories found for the given username.")
    else:
        return render(request, 'profile-page.html')



def convert_txt_to_pdf(content, pdf_filename):
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)

    for line in content.split('\n'):
        pdf.cell(200, 10, txt=line, ln=True)

    pdf.output(pdf_filename)



def get_github_user_data(username):
    url = f"https://api.github.com/users/{username}"
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
        print(f"Failed to retrieve user data from GitHub API. Status code: {response.status_code}")
        return None

def get_github_repos(username):
    url = f"https://api.github.com/users/{username}/repos"
    response = requests.get(url)
    if response.status_code == 200:
        return response.json()
    else:
                    print(f"Failed to retrieve repository data from GitHub API. Status code:
```

```
{response.status_code}")
        return None

def get_repo_details(repo):
    languages_url = repo.get("languages_url")
    commits_url = f"{repo.get('url')}/commits"
    languages_response = requests.get(languages_url)
    commits_response = requests.get(commits_url)
    if languages_response.status_code == 200 and commits_response.status_code == 200:
        languages_data = languages_response.json()
        commits_data = commits_response.json()
        return {
            'languages': languages_data,
            'commits_count': min(len(commits_data), 10),
            'repo_name': repo.get("name")  # Include repo_name here
        }
    else:
        return None

def profile_metrics_calculation(username):
    user_data = get_github_user_data(username)
    repos_data = get_github_repos(username)

    if user_data is None or repos_data is None:
        return None

    language_counts = defaultdict(int)
    commits_info = []

    with ThreadPoolExecutor(max_workers=10) as executor:
        futures = [executor.submit(get_repo_details, repo) for repo in repos_data]
        for future in futures:
            result = future.result()
```

```
            if result:
                languages_data = result['languages']
                for language in languages_data:
                    language_counts[language] += 1
                commits_info.append({
                    'repo_name': result['repo_name'],  # Access repo_name from result
                    'commits_count': result['commits_count']
                })

    top_languages = sorted(language_counts.items(), key=lambda x: x[1], reverse=True)

    return {
        'username': username,
        'avatar_url': user_data.get("avatar_url"),
        'name': user_data.get("name"),
        'total_repos': user_data.get("public_repos"),
        'followers': user_data.get("followers"),
        'following': user_data.get("following"),
        'top_languages': top_languages,
        'commits_info': commits_info
    }


def profile_analysis(request):
    data = None
    if request.method == "POST":
        username = request.POST.get('username')
        data = profile_metrics_calculation(username)
        if data is None:
            error = 'Failed to retrieve user data.'
            return render(request, 'profile.html', {'error': error})
    return render(request, 'profile.html', {'data': data})
```

```
# def profile_analyzer(request):
#     if request.method == 'POST':
#         username = request.POST.get('username')
#         github_avatar = "https://avatars.githubusercontent.com/u/120780784?v=4"
#         user_profile_link = f"https://github.com/{username}"
#         user_bio = ""
#         user_location = ""
#         user_top_languages = ""
#         total_repos = ""
#         total_commits_repowise = ""
#         total_followers = ""
#         total_subscribers = ""
#         #graph
#         commits_overtime = ""

#         content = {
#             " : ,

#         }
#         return render(request, 'profile_analyzer.html',content)


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

def user_signin(request):
    return render(request, 'register-page.html')
```

Folder: faiss_index

Folder: gemini

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: admin.py

Documented code for admin.py:

```
from django.contrib import admin

# Register your models here.
```

File: apps.py

Documented code for apps.py:

```
from django.apps import AppConfig


class GeminiConfig(AppConfig):
    default_auto_field = 'django.db.models.BigAutoField'
    name = 'gemini'
```

Folder: migrations

File: __init__.py

Documented code for __init__.py:

Folder: __pycache__

File: models.py

Documented code for models.py:

```
from django.db import models
```

```
# Create your models here.
```

File: tests.py

Documented code for tests.py:

```
from django.test import TestCase

# Create your tests here.
```

File: urls.py

Documented code for urls.py:

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

urlpatterns = [
    path('gemini', gemini, name='gemini'),
]
```

File: views.py

Documented code for views.py:

```
# views.py

from django.shortcuts import render
from django.http import HttpResponse
from django.conf import settings
from PyPDF2 import PdfReader
from langchain.text_splitter import RecursiveCharacterTextSplitter
import os
from langchain_google_genai import GoogleGenerativeAIEmbeddings
import google.generativeai as genai
```

```
from langchain_community.vectorstores import FAISS
from langchain_google_genai import ChatGoogleGenerativeAI
from langchain.chains.question_answering import load_qa_chain
from langchain.prompts import PromptTemplate
from dotenv import load_dotenv
import requests
from bs4 import BeautifulSoup
import urllib.parse

load_dotenv()
genai.configure(api_key=(os.getenv("GOOGLE_API_KEY")))

def get_pdf_text(pdf_docs):
    text = ""
    for pdf in pdf_docs:
        pdf_reader = PdfReader(pdf)
        for page in pdf_reader.pages:
            text += page.extract_text()
    print(text)
    return text

def get_text_chunks(text):
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=10000, chunk_overlap=1000)
    chunks = text_splitter.split_text(text)
    return chunks

def get_vector_store(text_chunks):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
    vector_store = FAISS.from_texts(text_chunks, embedding=embeddings)
    vector_store.save_local(os.path.join(settings.BASE_DIR, "faiss_index"))
    return vector_store

def get_conversational_chain():
```

```
    prompt_template = """
```

Answer the question thoroughly based on the provided code PDF input. As a code documenter, your task is to meticulously explain the code in detail each code line, including code snippets line by line , within a 2000-word limit. Additionally, provide frequently asked or related questions based on the user's query and suggest keywords for finding matching answers. Include relevant links to articles and YouTube videos related to the topic heading to enhance understanding. Ensure all pertinent details are covered. If the answer isn't explicitly stated in the provided context, utilize the information to craft an accurate response, incorporating your knowledge as necessary.

```
    Context:
    {context} (Provide the PDF containing the code for analysis)

    Question:
    {question}

    Answer:
    """

    model = ChatGoogleGenerativeAI(model="gemini-pro", temperature=0.3)

    prompt = PromptTemplate(template=prompt_template, input_variables=["context", "question"])
    chain = load_qa_chain(model, chain_type="stuff", prompt=prompt)

    return chain

def user_input(user_question):
    embeddings = GoogleGenerativeAIEmbeddings(model="models/embedding-001")
    new_db = FAISS.load_local(os.path.join(settings.BASE_DIR, "faiss_index"), embeddings)
    docs = new_db.similarity_search(user_question)
    chain = get_conversational_chain()
                response = chain({"input_documents": docs, "question": user_question}, return_only_outputs=True)
    response_text = response["output_text"]
```

```
    if response_text == "":
        response_text = "It seems that the answer is out of context. Here is a general response: ..."
    return response_text

def search_related_content(query):
    search_query = urllib.parse.quote(query)
    url = f"https://www.google.com/search?q={search_query}"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    search_results = soup.find_all('div', class_='BNeawe UPmit AP7Wnd')
    related_content = []
    for i, result in enumerate(search_results):
        if i >= 3:
            break
        related_content.append(result.text)
    return related_content

def scrape_youtube_videos(query):
    search_query = urllib.parse.quote(query)
    url = f"https://www.youtube.com/results?search_query={search_query}"
    response = requests.get(url)
    soup = BeautifulSoup(response.text, 'html.parser')
    video_results = soup.find_all('a', class_='yt-simple-endpoint style-scope ytd-video-renderer')
    related_videos = []
    for i, video in enumerate(video_results):
        if i >= 3:
            break
        video_title = video.get('title')
        video_link = f"https://www.youtube.com{video.get('href')}"
        related_videos.append((video_title, video_link))
    return related_videos

def display_related_content(related_content):
```

```
    return related_content

def gemini(request):
    if request.method == 'POST':
        # Handle PDF upload
        pdf_docs = request.FILES.getlist('pdf_files')
        raw_text = get_pdf_text(pdf_docs)
        text_chunks = get_text_chunks(raw_text)
        get_vector_store(text_chunks)

        # Handle user question
        user_question = request.POST.get('user_question')
        response_text = user_input(user_question)

        # Search related content
        related_content = search_related_content(user_question)
        youtube_content = scrape_youtube_videos(user_question)

        # Display related content
        related_content = display_related_content(related_content)

        # Return response
            return render(request, 'gemini.html', {'response_text': response_text, 'related_content': related_content})
    else:
        return render(request, 'gemini.html')

# Add appropriate URL mapping in urls.py
```

File: manage.py

Documented code for manage.py:

```
#!/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', 'autodocai.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()
```

Folder: static

Folder: assets

Folder: css

Folder: demo

Folder: fonts

Folder: img

Folder: js

Folder: core

Folder: plugins

Folder: scss

Folder: blk-design-system

Folder: bootstrap

Folder: mixins

Folder: utilities

Folder: custom

Folder: templates