Views.py

from django.shortcuts import render :- Imports the **render** function from **django.shortcuts**. The **render** function is used to render HTML templates with context data and return an **HttpResponse**.

from app.forms import \* :- Imports all forms defined in the **forms.py** file inside the **app** directory. The use of **\*** indicates that all forms are being imported.

from django.http import HttpResponse,HttpResponseRedirect :- Imports the **HttpResponse** and **HttpResponseRedirect** classes from **django.http**. These classes are used to generate HTTP responses.

from django.core.mail import send\_mail :- Imports the **send\_mail** function, which is used for sending email.

from django.contrib.auth import authenticate,login,logout

:- Imports functions related to user authentication.

* **authenticate**: Used to authenticate a user based on the provided username and password.
* **login**: Used to log in a user after they have been authenticated.
* **logout**: Used to log out a user.

from django.urls import reverse :- Imports the **reverse** function from **django.urls**. **reverse** is used to generate URLs for Django views, given the view name.

from django.contrib.auth.decorators import login\_required :- Imports the **login\_required** decorator from **django.contrib.auth.decorators**. This decorator is used to ensure that a view can only be accessed by authenticated users. If a user is not authenticated, they will be redirected to the login page.

**def home(request):**

    return render(request,'home.html')

**def registration(request):** :- **Registration Function:**

:- This function is the view for user registration.

umf=UserMF() :- **Creating Model Instances**

    pmf=ProfileMF()

    d={'umf':umf,'pmf':pmf}

:- It creates instances of the **UserMF** and **ProfileMF** models and stores them in a dictionary **d**.

    if request.method=='POST' and request.FILES: :- **Handling Form Submission**

:- This checks if the form is submitted using the POST method and if there are any files attached.

umfd=UserMF(request.POST) :- **Validating Form Data**

        pmfd=ProfileMF(request.POST,request.FILES)

        if umfd.is\_valid() and pmfd.is\_valid():

:- It creates instances of the form classes **UserMF** and **ProfileMF** with the submitted POST data and files, and then checks if both forms are valid.

Nud=umfd.save(commit=False) :- **Saving User Data:**

            submittedpw=umfd.cleaned\_data['password']

            Nud.set\_password(submittedpw)

            Nud.save()

:- If both forms are valid, it saves the user data. It sets the user's password using the **set\_password** method to ensure it is hashed before saving.

            Npd=pmfd.save(commit=False) :- **Saving Profile Data:**

            Npd.username=Nud

            Npd.save()

:- It saves the profile data, associating it with the user by setting the **username** field.

send\_mail( :- **Sending Email:**

                'Registration',

                'Ur Hotstar Registration succefully completed.....!',

                'abc@gmail.com',

                [Nud.email],

                fail\_silently=False),

:- After saving the data, it sends a registration confirmation email to the user using the **send\_mail** function.

            return HttpResponse('data submitted') :- **Returning Response**

:- it returns an HTTP response indicating that the data has been successfully submitted.

    return render(request,'registration.html',d) :- **Rendering Registration Form:**

**:-** If the request method is not POST or the forms are not valid, it renders the registration form template (**registration.html**) along with the form instances.

**def user\_login(request):** :- **Handling Form Submission:**

    if request.method=='POST':

:- **:-** The function checks if the form is submitted using the POST method.

        username=request.POST['username'] :- **Getting Username and Password from Form:**

        password=request.POST['password']

:- It retrieves the username and password from the submitted form data.

        AUO=authenticate(username=username,password=password) :- **Authenticating User:**

**:-** It uses the **authenticate** function from Django to check the provided username and password against the user database. The result is stored in the variable **AUO** (Authenticated User Object).

        if AUO: :- **Checking Authentication Result**

**:-** If the authentication is successful (i.e., **AUO** is not **None**), it proceeds to the next step.

            if AUO.is\_active: :- **Checking User Activation:**

**:-** It checks if the authenticated user (**AUO**) is active.

                login(request,AUO) :- **Logging in User:**

**:-** If the user is active, it logs in the user using the **login** function from Django.

                request.session['username']=username :- **Setting Session Data:**

**:-** It sets a session variable with the username for future use.

                return HttpResponseRedirect(reverse('home')) :- **Redirecting to Home Page:**

**:-** If the user is active and successfully logged in, it redirects to the home page.

            else: :- **Handling Inactive User:**

                return HttpResponse('Not active User')

:- If the user is not active, it returns an HTTP response indicating that the user is not active.

    return render(request,'user\_login.html') :- **Rendering Login Form:**

**:-** If the request method is not POST or the authentication is not successful, it renders the user login form template (**user\_login.html**). The form will be displayed for the user to enter their credentials.

37) @login\_required

:-The @login\_required decorator is used to restrict access to the display\_profile view to only authenticated users. If a user is not authenticated (i.e., not logged in), Django will redirect them to the login page before allowing access to this view.

38) **def display\_profile(request):**

:- This function is a Django view that takes a request object as a parameter.

39) username=request.session.get('username') :- **Retrieve Username from Session:**

:- It retrieves the username from the user's session. The session is a mechanism in Django to store and retrieve arbitrary data on a per-site-visitor basis.

40) UO=User.objects.get(username=username) :- **Retrieve User and Profile Objects**

41) PO=Profile.objects.get(username=UO)

:- It uses the username to fetch the corresponding User object (UO) from Django's built-in User model (django.contrib.auth.models.User). Then, it uses this User object to fetch the associated Profile object (PO). The assumption here is that there's a one-to-one relationship between the User and Profile models.

42) d={'UO':UO,'PO':PO} :- **Create a Dictionary for Context**

:- It creates a dictionary (d) containing the User object (UO) and the Profile object (PO). This dictionary will be passed as context data to the template.

return render(request,'display\_profile.html',d) :- **Render Template**

:- It renders the 'display\_profile.html' template using the render function. The template will have access to the User and Profile objects through the context data (d) and can use them to display information about the user.

@login\_required :- The **@login\_required** decorator ensures that only authenticated users can access this view. If a user is not logged in and tries to access this view, they will be redirected to the login page.

**def user\_logout(request):**

    logout(request) :- This function is provided by Django and is used to log the user out. It takes the **request** object as an argument and handles the necessary tasks to log the user out, such as clearing the session data.

    return HttpResponseRedirect(reverse('home')) :- After logging the user out, the view redirects them to the 'home' page. The **reverse('home')** function is used to generate the URL for the 'home' page. This is beneficial because it allows you to change the URL in the future without modifying the code in multiple places.

**def movies(request):**

    return render(request,'movies.html')

:- This is a simple view function for rendering a page related to movies. When a user accesses the URL associated with this view, it will render the 'movies.html' template using the **render** function. The **render** function takes the **request** object and the name of the template as arguments.

**def sports(request):**

    return render(request,'sports.html')

:- this is another view function for rendering a page related to sports. When a user accesses the URL associated with this view, it will render the 'sports.html' template using the **render** function.

@login\_required :- The **@login\_required** decorator is used to ensure that only authenticated users can access this view. If a user is not authenticated (i.e., not logged in) and attempts to access this view, they will be redirected to the login page.

**def change\_password(request):** :- This function handles the change password functionality.

    if request.method=="POST": :- **Checking the Request Method:**

**:-** This checks whether the request method is POST. In Django, forms usually use the POST method to submit data.

        pw=request.POST['newpassword'] :- **Processing POST Data**

**:-** Retrieves the new password from the POST data submitted by the form.

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

:- Retrieves the username from the user's session. The assumption here is that the username is stored in the session when the user logs in.

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

:- Retrieves the User object from the database based on the obtained username.

        UO.set\_password(pw)

:- Uses the **set\_password** method provided by Django's User model to set the user's password. This ensures that the password is properly hashed.

        UO.save()

:- Saves the updated User object with the new hashed password to the database.

        return HttpResponse('PASSWORD CHANGED SUCCESSFULLY') :- **Returning a Response:**

**:-** If the password change is successful, the view returns a simple HTTP response indicating success. In a real application, we might want to redirect the user to another page or display a success message in a template.

    return render(request,'change\_password.html') :- **Rendering the Change Password Form:**

:- If the request method is not POST (i.e., it's a GET request), the view renders the 'change\_password.html' template. This template likely contains a form for users to input their new password.

**def reset\_password(request):** :- This function is responsible for handling the reset password functionality.

    if request.method=='POST': :- **Checking the Request Method**

**:-** This checks whether the request method is POST. In Django, forms typically use the POST method to submit data.

        password=request.POST['password'] :- **Processing POST Data**

**:-** Retrieves the new password from the POST data submitted by the form.

        username=request.POST['username']

:- Retrieves the username from the POST data submitted by the form.

        LUO=User.objects.filter(username=username)

:- Queries the database to get a list of User objects with the specified username. The use of **filter** is appropriate here, as there might be multiple users with the same username.

        if LUO:

:- Checks if there is at least one User object with the specified username.

            UO=LUO[0]

:- If there is, retrieves the first User object from the list. Note that we assume here that usernames are unique, and there is at most one user with the given username.

            UO.set\_password(password)

:- Uses the **set\_password** method provided by Django's User model to set the user's password. This ensures that the password is properly hashed.

            UO.save()

:- Saves the updated User object with the new hashed password to the database.

            return HttpResponse('Modify The password successfully')

:- If the username is valid and the password is updated successfully, the view returns a simple HTTP response indicating success.

        else:

            return HttpResponse('Username is not validate')

:- If there are no User objects with the specified username, returns an HTTP response indicating that the username is not valid.

    return render(request,'reset\_password.html')

:- If the request method is not POST (i.e., it's a GET request), the view renders the 'reset\_password.html' template. This template likely contains an HTML form for users to input their new password and username.

Urls.py

from django.contrib import admin :- Imports the **admin** module from **django.contrib**. This is for setting up the Django admin interface.

from django.urls import path :- Imports the **path** function from **django.urls**. The **path** function is used to define URL patterns in Django.

from django.conf import settings :- Imports the **settings** module from **django.conf**. The **settings** module contains configuration settings for a Django project.

from django.conf.urls.static import static :- Imports the **static** function from **django.conf.urls.static**. This is used for serving static files during development.

from app.views import \* :- Imports all views from the **views** module within the **app** directory. The use of **\*** indicates that all views are being imported.

urlpatterns = [

    path('admin/', admin.site.urls), :- Maps the 'admin/' URL to the Django admin interface. This is the default setup for accessing the Django admin panel.

    path('home/',home,name='home'), :- Maps the 'home/' URL to the **home** view function. The **name='home'** is a convenient way to reference this URL in your templates or code using the **{% url 'home' %}** template tag or **reverse('home')**.

    path('registration/',registration,name='registration'), :- Maps the 'registration/' URL to the **registration** view function. Similar to the previous example, **name='registration'** is a reference for this URL.

    path('user\_login/',user\_login,name='user\_login'), :- Maps the 'user\_login/' URL to the **user\_login** view function with the name 'user\_login'.

    path('display\_profile/',display\_profile,name='display\_profile'), :- Maps the 'display\_profile/' URL to the **display\_profile** view function with the name 'display\_profile'.

    path('user\_logout/',user\_logout,name='user\_logout'), :- Maps the 'user\_logout/' URL to the **user\_logout** view function with the name 'user\_logout'.

    path('movies/',movies,name='movies'), :- Maps the 'movies/' URL to the **movies** view function with the name 'movies'.

    path('change\_password/',change\_password,name='change\_password'), :- Maps the 'change\_password/' URL to the **change\_password** view function with the name 'change\_password'.

    path('reset\_password/',reset\_password,name='reset\_password'), :- Maps the 'reset\_password/' URL to the **reset\_password** view function with the name 'reset\_password'.

    path('sports/',sports,name='sports') :- Maps the 'sports/' URL to the **sports** view function with the name 'sports'.

]+static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT) :- This line is adding the URL patterns for serving media files during development. It is necessary to serve media files in development, but you typically wouldn't use this in a production environment.

Wsgi.py

import os :- Imports the **os** module, which provides a way to interact with the operating system. In this case, it's used to set environment variables.

from django.core.wsgi import get\_wsgi\_application :- Imports the **get\_wsgi\_application** function from **django.core.wsgi**. This function returns a WSGI application that can be used to serve a Django project.

os.environ.setdefault('DJANGO\_SETTINGS\_MODULE', 'hotstar.settings')

* :- Sets the default value for the **DJANGO\_SETTINGS\_MODULE** environment variable. This variable tells Django which settings module to use. In this case, it's set to **'hotstar.settings'**.
* This line is crucial for informing Django about the project's settings. The **'hotstar.settings'** indicates that the settings module is located in the **hotstar** package or directory.

application = get\_wsgi\_application()

* :- Calls the **get\_wsgi\_application** function to retrieve a WSGI application for the Django project.
* The **application** variable is then assigned the WSGI application object, making it available for the WSGI server to use.

Settings.py

STATIC\_URL = 'static/' :- Static Files Configuration

:- This setting defines the base URL to serve static files from. In this case, static files like CSS, JavaScript, and images will be served under the '/static/' URL.

MEDIA\_URL='media/' :- Media Files Configuration

:- This setting defines the base URL for serving media files. Media files typically include user-uploaded files such as images or documents.

MEDIA\_ROOT=os.path.join(BASE\_DIR,'media')

:- This setting specifies the absolute filesystem path to the directory that will hold user-uploaded media files. In this case, it is set to a directory named 'media' within the project's base directory (**BASE\_DIR**).

EMAIL\_HOST='smtp.gmail.com' :- Email Configuration

:- This is the hostname of the email server. In this case, it is set to Gmail's SMTP server.

EMAIL\_HOST\_USER='abc@gmail.com'

:- The username for the email server, often the sender's email address.

EMAIL\_HOST\_PASSWORD='cyal zezu flko mxba'

:- The password for the email server.

EMAIL\_USE\_TLS=True

:- Specifies whether to use TLS (Transport Layer Security) when connecting to the email server. In this case, it's set to **True**

EMAIL\_USE\_SSL=False

:- Specifies whether to use SSL (Secure Sockets Layer) when connecting to the email server. In this case, it's set to **False**.

EMAIL\_PORT=587

:- The port to use when connecting to the email server. In this case, it's set to the default port for TLS on Gmail, which is **587**.

Media:

It is to user-uploaded files such as images, videos, audio files, and documents. Media files are distinct from static files, which include assets like stylesheets, JavaScript files, and images that are part of the application and don't change frequently.

**User Uploads:**

* Allow users to upload profile pictures, images, or any other files. For example, a social media platform might allow users to upload and share images.

**File Storage:**

* Store and manage files uploaded by users. Django provides a **FileField** and **ImageField** in its models for handling file uploads.

from django.db import models

class UserProfile(models.Model):

user = models.OneToOneField(User, on\_delete=models.CASCADE)

profile\_picture = models.ImageField(upload\_to='profile\_pics/')

**profile\_picture** is an **ImageField** that will store user-uploaded profile pictures.

**File Serving:**

* Serve these user-uploaded files to users when requested. Django provides a development server for this purpose during development. In production, you might use a web server like Nginx or Apache, or a cloud storage service.

from django.conf import settings

from django.conf.urls.static import static

urlpatterns = [

# ... other URL patterns

] + static(settings.MEDIA\_URL, document\_root=settings.MEDIA\_ROOT)

This configuration is necessary to serve media files during development. In production, you would typically use a web server or a cloud storage service.

**Downloadable Content:**

* Allow users to upload and share documents or files that others can download.

class Document(models.Model):

title = models.CharField(max\_length=255)

file = models.FileField(upload\_to='documents/')

**file** is a **FileField** that can store various types of files.

**Custom File Storage Backends:**

* Django allows us to use different storage backends for managing media files. We might use Google Cloud Storage or another storage service for handling large-scale media files.