CODING

Android Front-End Implementation of Secure User Login Interface Using XML and Java-

XML Layout Snippet (login_activity.xml):

```
<EditText
   android:hint="Enter Email"
   android:inputType="textEmailAddress"
   android:background="@android:drawable/edit_text"
   android:textColor="@android:color/white" />

<EditText
   android:hint="Enter Password"
   android:inputType="textPassword"
   android:background="@android:drawable/edit_text"
   android:textColor="@android:color/white" />

<Button
   android:text="Login"
   android:backgroundTint="#333"
   android:textColor="@android:color/white" />
```

Java Code Snippet (LoginActivity.java):

```
Button loginBtn = findViewById(R.id.loginBtn);
loginBtn.setOnClickListener(v -> {
   // Navigate to Home Screen
   startActivity(new Intent(LoginActivity.this, HomeActivity.class));
});
```

Android-Based User Registration Interface with Input Validation and Navigation Control-

XML Layout Snippet (signin_activity.xml):

```
<EditText
   android:hint="Enter Name"
   android:background="@android:drawable/edit_text"
   android:textColor="@android:color/white" />

<EditText
   android:hint="Enter Mobile"
   android:inputType="phone"
   android:background="@android:drawable/edit_text"
   android:textColor="@android:color/white" />

<Button
   android:text="Sign Up"
   android:backgroundTint="#333"
   android:textColor="@android:color/white" />
```

Java Code Snippet (SigninActivity.java):

```
signUpBtn.setOnClickListener(v -> {
   // Navigate to Home Screen (placeholder) after successful sign-up
   startActivity(new Intent(SignUpActivity.this, HomeActivity.class));
```

});

Splash Screen Interface with Timed Navigation

XML Layout Snippet (splash_activity.xml):

xml

CopyEdit

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#000"
    android:gravity="center">

    <ImageView
        android:id="@+id/logo"
        android:src="@drawable/app_logo"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerInParent="true" />
</RelativeLayout>
```

Java Code Snippet (SplashActivity.java):

java

CopyEdit

```
new Handler().postDelayed(() -> {
    Intent intent = new Intent(SplashActivity.this, LoginActivity.class);
    startActivity(intent);
    finish();
}, 3000); // 3-second splash screen
```

OTP Verification Screen

XML Layout Snippet (otp_activity.xml):

xml

CopyEdit

```
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:gravity="center"
    android:orientation="vertical"
    android:background="#121212"
    android:padding="24dp">

    <TextView
        android:text="Enter OTP"
        android:textColor="@android:color/white"
        android:textSize="20sp"
        android:layout_marginBottom="16dp" />

    <EditText
```

```
        android:id="@+id/otpField"
        android:hint="6-digit code"
        android:inputType="number"
        android:maxLength="6"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/edit_text"
        android:textColor="@android:color/white" />

    <Button
        android:id="@+id/verifyBtn"
        android:text="Verify"
        android:layout_marginTop="24dp"
        android:backgroundTint="#333"
        android:textColor="@android:color/white" />
</LinearLayout>
```

Java Code Snippet (OtpActivity.java):
java
CopyEdit
```
verifyBtn.setOnClickListener(v -> {
    // Add OTP verification logic here
    startActivity(new Intent(OtpActivity.this, ExploreActivity.class));
});
```

Explore Page Interface

XML Layout Snippet (explore_activity.xml):
xml
CopyEdit
```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:background="#121212"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <LinearLayout
        android:orientation="vertical"
        android:padding="16dp"
        android:layout_width="match_parent"
        android:layout_height="wrap_content">

        <TextView
            android:text="Explore Styles"
            android:textColor="@android:color/white"
            android:textSize="24sp"
            android:layout_marginBottom="16dp" />

        <ImageView
            android:src="@drawable/style_sample"
            android:layout_width="match_parent"
```

```
        android:layout_height="200dp"
        android:scaleType="centerCrop"
        android:layout_marginBottom="12dp" />

    <!-- Repeat ImageViews or use RecyclerView for dynamic content -->
  </LinearLayout>
</ScrollView>
```

Enable Location Permission Popup

Java Code Snippet (ExploreActivity.java):

```
java
CopyEdit
if (ContextCompat.checkSelfPermission(this, Manifest.permission.ACCESS_FINE_LOCATION)
      != PackageManager.PERMISSION_GRANTED) {
  ActivityCompat.requestPermissions(this,
      new String[]{Manifest.permission.ACCESS_FINE_LOCATION}, 1);
} else {
  // Location access already granted
}
```

Handling permission result:

```
java
CopyEdit
@Override
public void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults) {
  if (requestCode == 1 && grantResults.length > 0
      && grantResults[0] == PackageManager.PERMISSION_GRANTED) {
      Toast.makeText(this, "Location Enabled", Toast.LENGTH_SHORT).show();
  } else {
      Toast.makeText(this, "Location Permission Denied", Toast.LENGTH_SHORT).show();
  }
}
```

Unity C# Script for Real-Time Camera Feed Capture and Local Image Storage via UI Interaction-

```
using UnityEngine;
using UnityEngine.UI;
using System.IO;
public class CameraCapture : MonoBehaviour
{
public RawImage cameraPreview; // UI RawImage to display camera
public Button captureButton; // Button to capture image
private WebCamTexture webcamTexture;
private string filePath;
void Start()
{
// Start the device's camera
webcamTexture = new WebCamTexture();
```

```
cameraPreview.texture = webcamTexture;
webcamTexture.Play();
// Assign button click event
captureButton.onClick.AddListener(CaptureImage);
}
void CaptureImage()
{
// Create a new texture and copy the camera pixels
Texture2D snap = new Texture2D(webcamTexture.width, webcamTexture.height);
snap.SetPixels(webcamTexture.GetPixels());
snap.Apply();
// Save image to device storage
filePath = Path.Combine(Application.persistentDataPath, "capturedCloth.png");
File.WriteAllBytes(filePath, snap.EncodeToPNG());
Debug.Log("Image Saved at: " + filePath);
}
```

Unity C# Script for Generating a 3D Mesh from a Background-Removed Image

```
using UnityEngine;
using System.IO;
\#if UNITY\_EDITOR
using UnityEditor;
\#endif
public class ImageTo3DMesh : MonoBehaviour
{
public void GenerateMeshNow()
{
string path = Path.Combine(Application.persistentDataPath,
"clothing\_no\_bg.png");
if (File.Exists(path))
{
byte\[] fileData = File.ReadAllBytes(path);
Texture2D tex = new Texture2D(2, 2);
tex.LoadImage(fileData);
```
```
if (tex.width <= 2 || tex.height <= 2)
{
Debug.LogError("Loaded texture might be invalid or too small.");
return;
}
Debug.Log("Texture size: " + tex.width + " x " + tex.height);
// Create mesh
Mesh mesh = GenerateSimpleQuad();
GameObject meshObject = new GameObject("Generated3DMesh");
// Add components
MeshRenderer renderer = meshObject.AddComponent<MeshRenderer>();
MeshFilter filter = meshObject.AddComponent<MeshFilter>();
filter.mesh = mesh;
meshObject.AddComponent<MeshCollider>();
// Apply texture
```

```
Material mat = new Material(Shader.Find("Unlit/Texture"));
mat.mainTexture = tex;
renderer.material = mat;
// Position and scale
meshObject.transform.position = Vector3.zero;
meshObject.transform.localScale = new Vector3(4f, 4f, 1f); // bigger for
```
visibility
```
// Position camera to view mesh
if (Camera.main != null)
{
Camera.main.transform.position = new Vector3(0, 0, -10);
Camera.main.transform.LookAt(meshObject.transform);
}
Debug.Log("3D mesh created and visible in scene at: " +
```
meshObject.transform.position);
\#if UNITY_EDITOR
Selection.activeGameObject = meshObject;
SceneView\.lastActiveSceneView\.FrameSelected();
\#endif
}
else
{
Debug.LogError("Background-removed image not found for mesh generation.");
}
}
private Mesh GenerateSimpleQuad()
{
Mesh mesh = new Mesh();
mesh.name = "SimpleQuad";
```
mesh.vertices = new Vector3[]
{
new Vector3(-0.5f, -0.5f, 0),
new Vector3(-0.5f, 0.5f, 0),
new Vector3( 0.5f, 0.5f, 0),
new Vector3( 0.5f, -0.5f, 0)
};
mesh.uv = new Vector2[]
{
new Vector2(0, 0),
new Vector2(0, 1),
new Vector2(1, 1),
new Vector2(1, 0)
};
mesh.triangles = new int[] { 0, 1, 2, 0, 2, 3 };
mesh.RecalculateNormals();
mesh.RecalculateBounds();
```

```
return mesh;
```
}
}

Unity C# Script for Real-Time Camera Feed Capture and Local Image Storage via UI Interaction

```
using UnityEngine;
using UnityEngine.UI;
using System.IO;

public class CameraCapture : MonoBehaviour
{
    public RawImage cameraPreview; // UI RawImage to display camera
    public Button captureButton; // Button to capture image
    private WebCamTexture webcamTexture;
    private string filePath;

    void Start()
    {
        webcamTexture = new WebCamTexture();
        cameraPreview.texture = webcamTexture;
        webcamTexture.Play();

        captureButton.onClick.AddListener(CaptureImage);
    }

    void CaptureImage()
    {
        Texture2D snap = new Texture2D(webcamTexture.width, webcamTexture.height);
        snap.SetPixels(webcamTexture.GetPixels());
        snap.Apply();

        filePath = Path.Combine(Application.persistentDataPath, "capturedCloth.png");
        File.WriteAllBytes(filePath, snap.EncodeToPNG());
        Debug.Log("Image Saved at: " + filePath);
    }
}
```

Unity C# Script to Apply Texture from a 2D Image

```
using UnityEngine;

public class LoadImage : MonoBehaviour
{
    public Texture2D imageTexture; // Assign your 2D image in the Inspector

    void Start()
    {
        Renderer renderer = GetComponent<Renderer>();
```

```
        if (renderer != null && imageTexture != null)
        {
            renderer.material.mainTexture = imageTexture; // Apply texture to the Quad
        }
    }
}
```

Import Libraries:

```
# Core libraries

import os

import requests

from tqdm import tqdm

from dotenv import load_dotenv

# Data manipulation

import pandas as pd

# Deep learning and embeddings

import torch

import open_clip

from PIL import Image

# Vector store

import chromadb

from chromadb.config import Settings

# Generative AI for text-based reasoning

import google.generativeai as genai

# Frontend framework

import streamlit as st
```

Environment Setup:

```
# Load environment variables

load_dotenv()

GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")

genai.configure(api_key=GOOGLE_API_KEY)
```

Dataset Loading:

```
from datasets import load_dataset
from tqdm import tqdm
import os

def load_fashionpedia_dataset(split='train'):
    print(f"Loading Fashionpedia dataset ({split} split)...")
    return load_dataset("detection-datasets/fashionpedia", split=split)

def save_images(dataset, dataset_folder, num_images=1000):
    os.makedirs(dataset_folder, exist_ok=True)
    print(f"Saving up to {num_images} images to '{dataset_folder}'...")

    for i in tqdm(range(min(num_images, len(dataset)))):
        try:
            image = dataset[i]['image']
            image.save(os.path.join(dataset_folder, f'image_{i+1:04d}.png'))
        except Exception as e:
            print(f"Error saving image {i+1}: {e}")

    print(f"Finished saving images to '{dataset_folder}'.")
```

```
if __name__ == "__main__":
    dataset_folder = 'Data'
    num_images_to_save = 1000
    dataset = load_fashionpedia_dataset(split='train')
    save_images(dataset, dataset_folder, num_images=num_images_to_save)
```

Image Dataset to Embeding Model:

```
# IMAGE DATASET TO EMBEDDING MODEL

# Step 1: Load and preprocess images
from PIL import Image
from glob import glob

image_paths = glob("fashion_images/*.jpg")[:1000]  # Limit to 1000 images for faster processing
print(f"Total images to process: {len(image_paths)}")

# Step 2: Initialize the OpenCLIP model and preprocessing
from open_clip import create_model_and_transforms
import torch

device = "cuda" if torch.cuda.is_available() else "cpu"
model, _, preprocess = create_model_and_transforms('ViT-B-32', pretrained='laion2b_s34b_b79k')
model = model.to(device)
model.eval()  # Set model to evaluation mode

# Step 3: Initialize ChromaDB and embedding function
import chromadb
from chromadb.utils.embedding_functions import OpenCLIPEmbeddingFunction

chroma_client = chromadb.Client()
collection = chroma_client.get_or_create_collection(
    name="fashion_embeddings",
```

```
    embedding_function=OpenCLIPEmbeddingFunction(model_name="ViT-B-32::laion2b_s34b_b79k")
)

# Step 4: Generate and store image embeddings
for i, img_path in enumerate(image_paths):
    try:
        img = Image.open(img_path).convert("RGB")
        img_tensor = preprocess(img).unsqueeze(0).to(device)

        with torch.no_grad():
            embedding = model.encode_image(img_tensor).cpu().numpy().flatten()

        collection.add(
            documents=[img_path],  # Store the path or metadata
            embeddings=[embedding.tolist()],
            ids=[f"img_{i}"]
        )

        if i % 100 == 0:
            print(f"{i} images processed...")
    except Exception as e:
        print(f"Error processing {img_path}: {e}")

print("Image embedding complete. Embeddings stored in ChromaDB.")
```

Embedding Model to Vector Database:

```
# EMBEDDING MODEL TO VECTOR DATABASE

# Import required libraries
import chromadb
from chromadb.utils.embedding_functions import OpenCLIPEmbeddingFunction
from open_clip import create_model_and_transforms
```

```
from PIL import Image
import torch
from glob import glob

# Step 1: Set up device and load OpenCLIP model
device = "cuda" if torch.cuda.is_available() else "cpu"
model, _, preprocess = create_model_and_transforms('ViT-B-32', pretrained='laion2b_s34b_b79k')
model = model.to(device)
model.eval()

# Step 2: Initialize ChromaDB client and collection
chroma_client = chromadb.Client()
collection = chroma_client.get_or_create_collection(
    name="fashion_embeddings",
    embedding_function=OpenCLIPEmbeddingFunction(model_name="ViT-B-32::laion2b_s34b_b79k")
)

# Step 3: Load image paths
image_paths = glob("fashion_images/*.jpg")[:1000]  # Limit for performance
print(f"Preparing to store {len(image_paths)} embeddings...")

# Step 4: Generate and add embeddings to ChromaDB
for i, img_path in enumerate(image_paths):
    try:
        img = Image.open(img_path).convert("RGB")
        img_tensor = preprocess(img).unsqueeze(0).to(device)

        with torch.no_grad():
            embedding = model.encode_image(img_tensor).cpu().numpy().flatten()

        collection.add(
            documents=[img_path],         # You can replace this with a description or label
            embeddings=[embedding.tolist()],
            ids=[f"img_{i}"]
```

```
        )

        if i % 100 == 0:
            print(f"Stored {i} embeddings into ChromaDB...")
    except Exception as e:
        print(f"Error processing {img_path}: {e}")

print("All embeddings successfully stored in the vector database.")
```

Query to Vector Database:

```
# QUERY TO VECTOR DATABASE

# Import necessary libraries
from open_clip import tokenize
import numpy as np
import torch
from PIL import Image
import matplotlib.pyplot as plt

# Step 1: Define query functions

# For text-based query
def search_similar_fashion_by_text(query_text, top_k=5):
    results = collection.query(
        query_texts=[query_text],
        n_results=top_k
    )
    print(f"\nTop {top_k} similar items for: \"{query_text}\"")
    for i, path in enumerate(results['documents'][0]):
        print(f"{i+1}. {path}")
```

```
        img = Image.open(path)
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"Match {i+1}")
        plt.show()

# For image-based query
def search_similar_fashion_by_image(image_path, top_k=5):
    try:
        img = Image.open(image_path).convert("RGB")
        img_tensor = preprocess(img).unsqueeze(0).to(device)

        with torch.no_grad():
            embedding = model.encode_image(img_tensor).cpu().numpy().flatten()

        results = collection.query(
            query_embeddings=[embedding.tolist()],
            n_results=top_k
        )

        print(f"\nTop {top_k} visually similar items to: {image_path}")
        for i, path in enumerate(results['documents'][0]):
            print(f"{i+1}. {path}")
            img = Image.open(path)
            plt.imshow(img)
            plt.axis('off')

            plt.title(f"Match {i+1}")
            plt.show()

    except Exception as e:
        print(f"Error processing image query: {e}")

# Example Usage
```

```
# Text Query Example
search_similar_fashion_by_text("red floral summer dress", top_k=5)

# Image Query Example
search_similar_fashion_by_image("fashion_images/sample_query.jpg", top_k=5)
```

Vector Database to Retrieve Image:

```
# VECTOR DATABASE TO RETRIEVE IMAGE

# Import necessary libraries
import matplotlib.pyplot as plt
from PIL import Image

# Step 1: Retrieve and display top similar images

# Function to visualize retrieved images after querying the database
def display_retrieved_images(results, top_k=5):
    """
    Displays the top k retrieved images based on the vector database query result.

    Parameters:

    - results: The result from the vector database query.
    - top_k: The number of top similar images to display.
    """
    print(f"\nDisplaying top {top_k} retrieved images from the database:")

    # Loop through the retrieved paths
    for i, path in enumerate(results['documents'][0]):
```

```
        print(f"{i+1}. {path}")  # Display the file path of the retrieved image

        # Open and display the image
        img = Image.open(path)
        plt.figure(figsize=(4, 4))
        plt.imshow(img)
        plt.axis('off')
        plt.title(f"Match {i+1}")
        plt.show()

# Example Usage (after querying the vector database)

# Assuming `results` is the output of a query like `search_similar_fashion_by_text` or `search_similar_fashion_by_image`
# Example result format: {"documents": [["/path/to/image1.jpg", "/path/to/image2.jpg", ...]]}

# You would pass the `results` from the previous query here to display the images
display_retrieved_images(results, top_k=5)
```

Retrieved Image to Vision Model:

```
# RETRIEVED IMAGE TO VISION MODEL

# Import necessary libraries
from PIL import Image
import torch
import torchvision.transforms as T
```

```
import open_clip  # Ensure open_clip is installed
import numpy as np

# Load OpenCLIP model and preprocessing
model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', pretrained='openai')
tokenizer = open_clip.get_tokenizer('ViT-B-32')

# Set model to evaluation mode
model.eval()

# Function to analyze a retrieved image
def analyze_image_with_vision_model(image_path):
    """
    Analyzes a retrieved fashion image using OpenCLIP and returns image features.

    Parameters:
    - image_path: Path to the retrieved image.

    Returns:
    - img_features: The image embedding vector.
    """
    # Load and preprocess the image
    image = Image.open(image_path).convert("RGB")
    image_input = preprocess(image).unsqueeze(0)  # Add batch dimension

    with torch.no_grad():
        # Extract image embedding
        image_features = model.encode_image(image_input)

    # Normalize the embedding
    image_features = image_features / image_features.norm(dim=-1, keepdim=True)
    return image_features

# Example usage with one of the top retrieved images
```

```
retrieved_img_path = results['documents'][0][0]  # First image from previous ChromaDB result
img_features = analyze_image_with_vision_model(retrieved_img_path)

print("Image features extracted successfully. Vector shape:", img_features.shape)
```

Embedding save:

```
import chromadb
from chromadb.utils.embedding_functions import OpenCLIPEmbeddingFunction
from chromadb.utils.data_loaders import ImageLoader
import os

dataset_folder = 'Data'
chroma_client = chromadb.PersistentClient(path="Vector_database")
image_loader = ImageLoader()
CLIP = OpenCLIPEmbeddingFunction()

image_vdb = chroma_client.get_or_create_collection(name="image", embedding_function = CLIP, data_loader =
image_loader)

ids = []
uris = []

for i, filename in enumerate(sorted(os.listdir(dataset_folder))):
    if filename.endswith('.png'):
        file_path = os.path.join(dataset_folder, filename)

        ids.append(str(i))
        uris.append(file_path)

image_vdb.add(
    ids=ids,
    uris=uris
```

```
)

print("Images stored to the Vector database.")
```

Streamlit App:

```
import asyncio
import sys
if sys.platform.startswith('win'):
    asyncio.set_event_loop_policy(asyncio.WindowsSelectorEventLoopPolicy())

import streamlit as st
import numpy as np
import PIL.Image
import io
import requests
import google.generativeai as genai
from chromadb.utils.embedding_functions import SentenceTransformerEmbeddingFunction
import chromadb
from dotenv import load_dotenv
import os
import warnings
import json
from datetime import datetime
from tensorflow import keras
from fpdf import FPDF

warnings.filterwarnings("ignore")

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

HISTORY_FILE = "styling_history.json"
```

```
WARDROBE_DIR = "wardrobe_images"

# Save styling history
def save_to_history(entry):
    if os.path.exists(HISTORY_FILE):
        with open(HISTORY_FILE, "r") as file:
            history = json.load(file)
    else:
        history = []

    history.append(entry)
    with open(HISTORY_FILE, "w") as file:
        json.dump(history, file, indent=4)

# Load styling history
def load_history():
    if os.path.exists(HISTORY_FILE):
        with open(HISTORY_FILE, "r") as file:
            return json.load(file)
    return []

# Clear styling history
def clear_history():
    if os.path.exists(HISTORY_FILE):
        os.remove(HISTORY_FILE)

# Download recommendations as PDF
def download_recommendations():
    history = load_history()
    if not history:
        st.error("No recommendations to download.")
        return

    pdf = FPDF()
```

```
    pdf.add_page()
    pdf.set_font("Arial", size=12)

    for entry in history:
        pdf.cell(200, 10, f"Timestamp: {entry['timestamp']}", ln=True)
        pdf.cell(200, 10, f"Query: {entry['query']}", ln=True)
        for rec in entry['recommendations']:
            pdf.multi_cell(0, 10, f"{rec['image']} - {rec['advice']}")
        pdf.ln(10)

    pdf.output("recommendations.pdf")
    st.success("Recommendations downloaded as PDF.")

# Style tag helper
def tag_style(tags):
    return ", ".join(tags)

# Image loading helper
def open_image(img_data):
    if isinstance(img_data, str):
        response = requests.get(img_data)
        img = PIL.Image.open(io.BytesIO(response.content))
    elif isinstance(img_data, np.ndarray):
        img = PIL.Image.fromarray(img_data.astype('uint8'))

    elif isinstance(img_data, list):
        try:
            img_data = np.array(img_data, dtype='uint8')
            img = PIL.Image.fromarray(img_data)
        except Exception as e:
            st.error(f"Error converting list to array: {e}")
            raise ValueError("Unsupported image data format")
    else:
        raise ValueError("Unsupported image data format")
```

```
    return img

# Wardrobe scanning
def wardrobe_scanning():
    st.subheader("Wardrobe Scanning")
    uploaded_files = st.file_uploader("Upload your wardrobe images:", type=["jpg", "jpeg", "png"], accept_multiple_files=True)
    if uploaded_files:
        for file in uploaded_files:
            image = np.array(PIL.Image.open(file))
            st.image(image, caption=f"Scanned: {file.name}")

# Mood board helper
def add_to_mood_board(image):
    if not os.path.exists(WARDROBE_DIR):
        os.makedirs(WARDROBE_DIR)
    image.save(os.path.join(WARDROBE_DIR, f"mood_{datetime.now().strftime('%Y%m%d_%H%M%S')}.png"))
    st.success("Added to Mood Board!")

# App UI
st.title("AI the Fashion Styling Assistant")

uploaded_file = st.file_uploader("Upload an image:", type=["jpg", "jpeg", "png"])
query = st.text_input("Enter styling query:")
tags = st.multiselect("Select tags:", ["Casual", "Formal", "Summer", "Winter", "Party", "Business"])

# Sidebar: History
st.sidebar.subheader("Styling History")
history = load_history()
if history:
    for entry in history[::-1]:
        st.sidebar.write(f"**{entry['timestamp']} - {entry['query']}**")
        for rec in entry["recommendations"]:
            st.sidebar.write(f"- {rec['advice'][:100]}...")

# Buttons
if st.button("Show History"):
    st.write(history)

if st.button("Clear History"):
    clear_history()
```

```
    st.success("History cleared successfully.")

if st.button("Download Recommendations"):
    download_recommendations()

if st.button("Share on Social Media"):
    st.success("Shared on social media!")

if st.button("Shop Similar Styles"):
    st.write("[Visit our e-commerce site](https://www.example.com)")

# Generate Recommendations
if st.button("Generate Styling Ideas"):
    try:
        chroma_client = chromadb.PersistentClient(path="Vector_database")
        embedding_function = SentenceTransformerEmbeddingFunction(model_name="all-MiniLM-L6-v2")
        image_vdb = chroma_client.get_or_create_collection(
            name="image",
            embedding_function=embedding_function
        )

        history_entry = {
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "query": query if query else "Image Upload",
            "tags": tags,
            "recommendations": []
        }

        if uploaded_file is not None:
            uploaded_image = np.array(PIL.Image.open(uploaded_file))
            retrieved_imgs = image_vdb.query(query_texts=[query], n_results=3)

            for i, img_data in enumerate(retrieved_imgs['documents'][0]):
                img_url = img_data
                response = requests.get(img_url)
                img = PIL.Image.open(io.BytesIO(response.content))
                st.image(img, caption=f"Image {i+1}")
                model = genai.GenerativeModel(model_name="gemini-1.5-pro")
                response = model.generate_content(["Generate styling advice.", img])
                advice = response.text
```

```
            st.write(f"Styling Advice: {advice}")
            history_entry["recommendations"].append({"image": f"Image {i+1}", "advice": advice})

            add_to_mood_board(img)

        save_to_history(history_entry)

    except Exception as e:
        st.error(f"Error while generating recommendations: {e}")

# Mood Board features
if st.button("Add to Mood Board"):
    if uploaded_file is not None:
        image = PIL.Image.open(uploaded_file)
        add_to_mood_board(image)

if st.button("View Mood Board"):
    if os.path.exists(WARDROBE_DIR):
        images = [os.path.join(WARDROBE_DIR, img) for img in os.listdir(WARDROBE_DIR)]
        st.image(images, caption=[os.path.basename(img) for img in images])
    else:
        st.info("Mood board is empty.")

# Wardrobe scanning
if st.button("Scan Wardrobe"):
    wardrobe_scanning()

# Gamification + Model customization
st.sidebar.write(" Your Style Points: 100")
st.sidebar.write("Level up by using more features!")
st.sidebar.subheader("Model Customization")
selected_model = st.sidebar.selectbox("Choose a model:", ["Model A", "Model B", "Model C"])
```

SCREENSHOTS

Dataset loaded:

Embedding Process:

```
PS G:\aifess> python "Image embedding.py"
Total images to process: 1000
INFO:root:Loaded ViT-B-32 model config.
INFO:root:Loading pretrained ViT-B-32 weights (laion2b_s34b_b79k).
INFO:chromadb.telemetry.product.posthog:Anonymized telemetry enabled. See                    https://docs.trychroma.com/telemetry for mor
e information.
Embedding images:   0%|                                                                                | 0/1000 [00:00<?, ?it/s]I
NFO:embedding_utils:0 images processed...
Embedding images:  10%|                                                                  | 100/1000 [00:09<01:26, 10.38it/s]I
NFO:embedding_utils:100 images processed...
Embedding images:  20%|                                                                  | 200/1000 [00:19<01:17, 10.37it/s]I
NFO:embedding_utils:200 images processed...
Embedding images:  30%|                                                                  | 300/1000 [00:29<01:08, 10.28it/s]I
NFO:embedding_utils:300 images processed...
Embedding images:  40%|                                                                  | 399/1000 [00:38<00:59, 10.05it/s]I
NFO:embedding_utils:400 images processed...
Embedding images:  50%|                                                                  | 500/1000 [00:48<00:47, 10.62it/s]I
NFO:embedding_utils:500 images processed...
Embedding images:  60%|                                                                  | 599/1000 [00:58<00:39, 10.06it/s]I
NFO:embedding_utils:600 images processed...
Embedding images:  70%|                                                                  | 700/1000 [01:08<00:31,  9.39it/s]I
NFO:embedding_utils:700 images processed...
Embedding images:  80%|                                                                  | 799/1000 [01:19<00:21,  9.22it/s]I
NFO:embedding_utils:800 images processed...
Embedding images:  90%|                                                                  | 899/1000 [01:29<00:11,  9.14it/s]I
NFO:embedding_utils:900 images processed...
Embedding images: 100%|                                                                 | 1000/1000 [01:40<00:00,  9.99it/s]
INFO:embedding_utils:Image embedding complete. 1000 embeddings stored in ChromaDB.
```

Streamlit App:

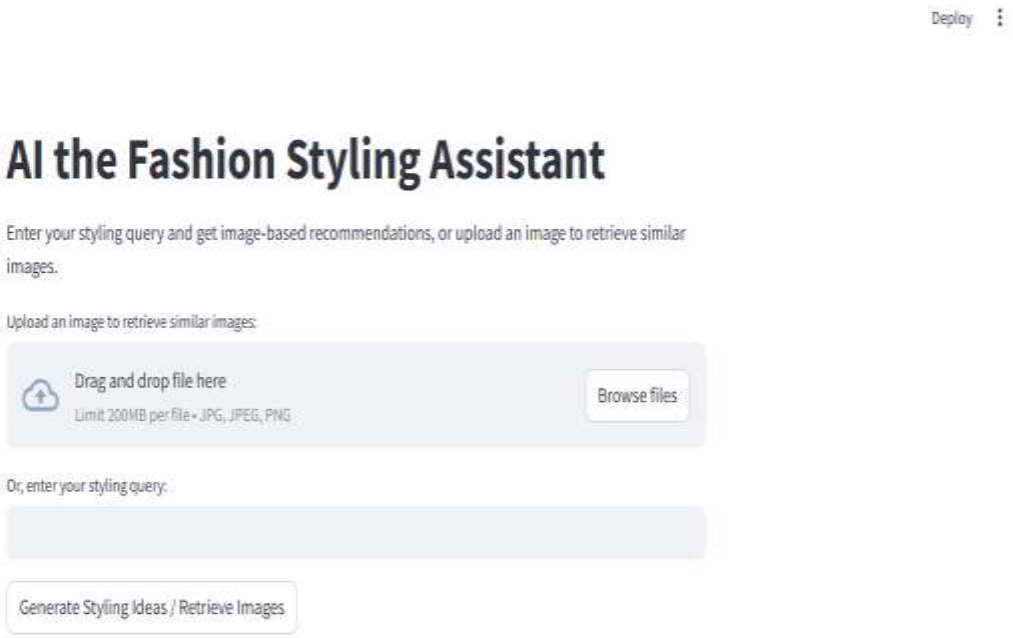
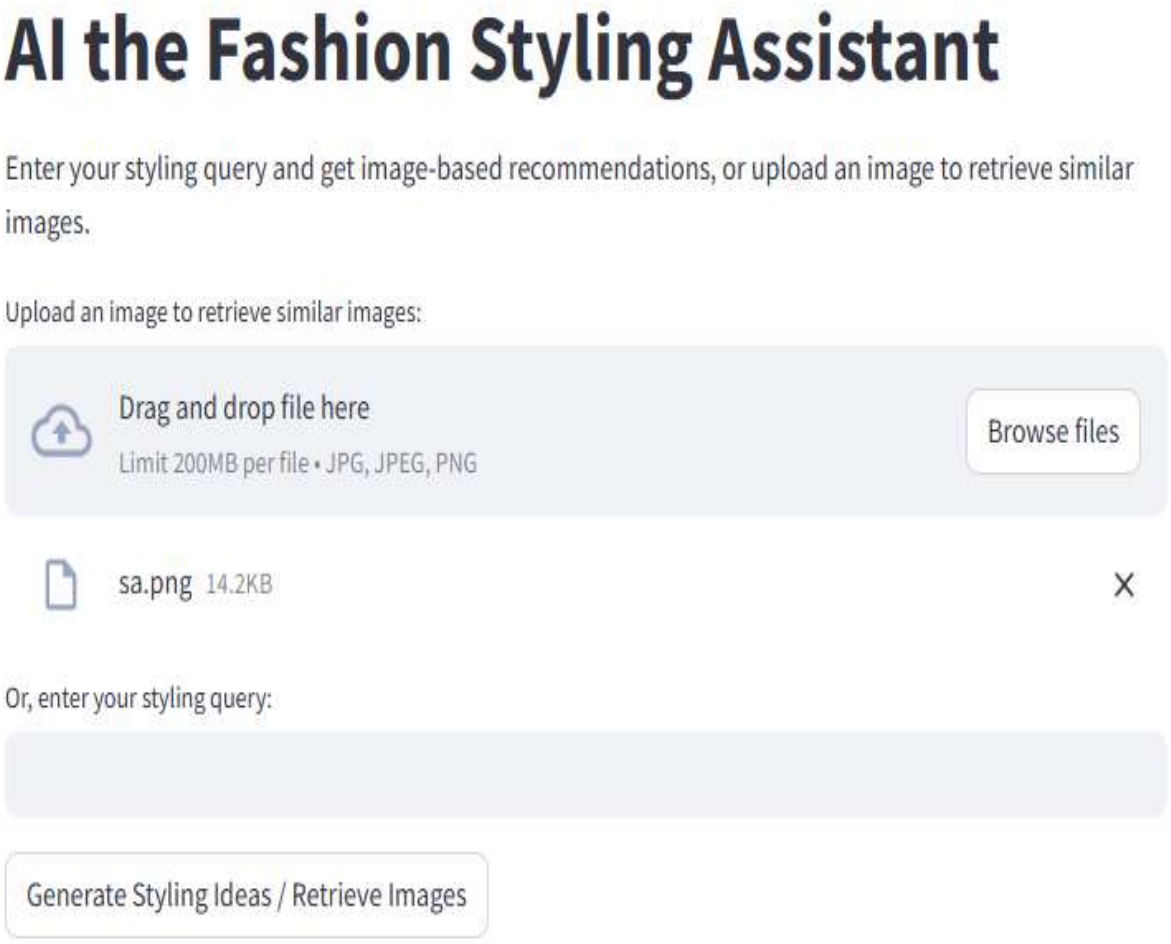


Image Query:

Image Query uploaded:

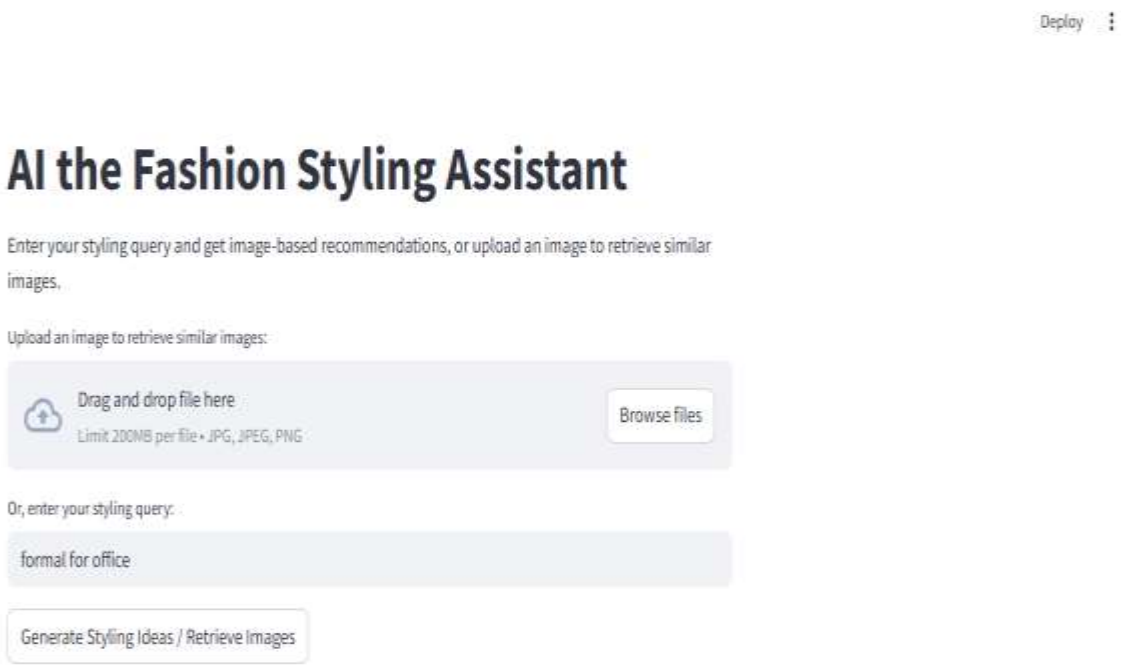


Suggestions:

Retrieved Similar Images:

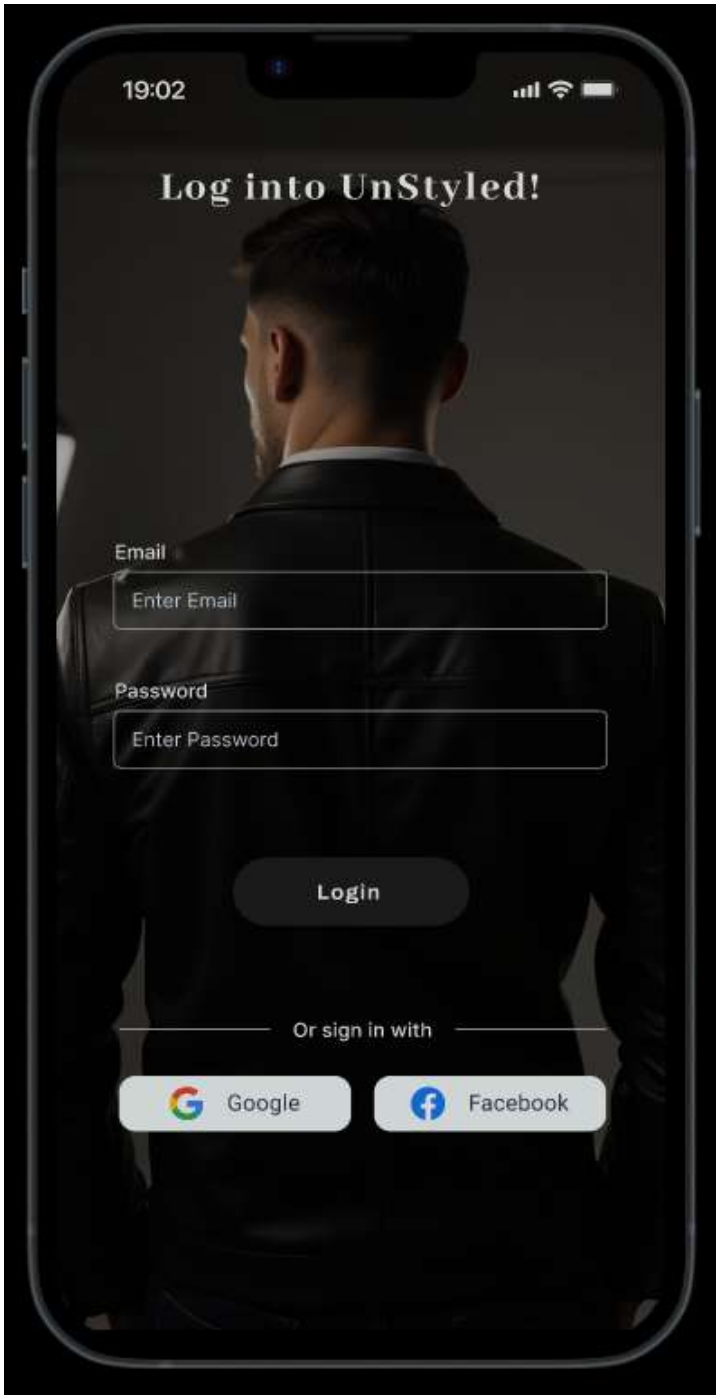
Retrieved image 1

Text Query uploaded:



Suggestions:

Login Page:


19:02
Log into UnStyled!
Email
Enter Email
Password
Enter Password
Login
Or sign in with
Google
Facebook

Signup Page:


19:02
Let's Sign Up!
Name
Enter Name
Email
Enter Email
Mobile
Enter Mobile
Gender
Enter Gender
Password
Enter Password
Confirm Password
Re-enter Password
Sign Up

OTP Screen:

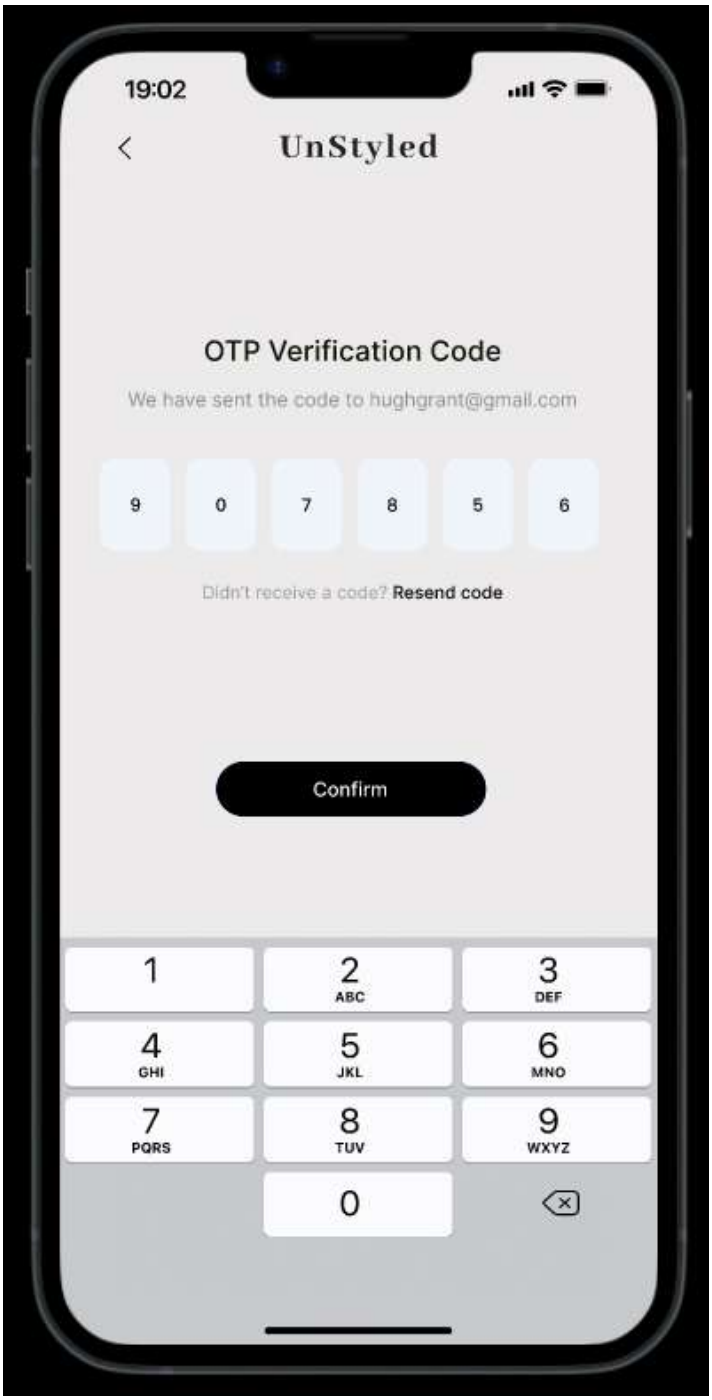


Figure B *Sign up Page*

AR try-on screen:

3D Mesh Generation