AI Styling App - Beginner's Complete Guide

This guide explains, step-by-step, how to build a mobile-first AI-powered personal styling app using Google Cloud Vision (OCR) and Google AI Studio / Gemini for recommendations. It includes code snippets (React Native, Node.js, Python), API references, and practical developer steps for an MVP targeted at Delhi, India.

What you will find in this document:

- Detailed beginner-friendly step-by-step build path
- Code snippets for core flows (upload, OCR, pose, tagging, recommendation)
- Suggested APIs and official references
- Testing, deployment, and next steps

1. Quick setup checklist

Quick Start Checklist (what to set up before coding):

1. Google Cloud account with billing enabled.

2. Create a GCP project (Google Cloud Console) for your app.

3. Enable APIs: Cloud Vision API, Vertex AI (Generative AI/Gemini), Cloud Storage, Cloud Run (or Cloud Functions).

4. Install Google Cloud SDK (gcloud) and login: `gcloud auth login` and `gcloud auth application-default login`.

5. Create a service account with roles: Vision API User, Vertex AI User, Storage Object Admin (for backend). Download the JSON key securely.

6. Create a Cloud Storage bucket for storing uploaded images (private).

7. Choose mobile framework: React Native (recommended) or Flutter.

2. Recommended Architecture (textual)

Architecture (high-level):

- Mobile app (React Native) --- handles camera & image upload to GCS (signed URL).
- Backend (Cloud Run / Cloud Functions) --- receives upload notification, calls Cloud Vision OCR, calls custom vision or Vertex models, stores metadata in Firestore or MongoDB, calls Gemini (Vertex AI) for outfit suggestions.
- ML components: MediaPipe (pose & landmarks), YOLOv8 or Vertex AutoML (garment detection), embedding service (optional) for compatibility scoring.
- Storage: Google Cloud Storage for images; Firestore/Postgres for metadata.
- Optional: CDN, Redis queue for async processing, monitoring via Google Cloud Monitoring.

3. Key APIs & Official Docs (start here)

Key APIs and official documentation (start here):

- Google Cloud Vision (OCR & Label Detection) --- official docs and quickstarts. Use this for text on clothing labels and base image labels. (Docs: cloud.google.com/vision/docs)
- Vertex AI / Gemini (Google AI Studio) --- use Vertex's generative models (Gemini) to convert structured metadata + user profile into natural-language outfit suggestions and structured JSON responses. (Docs: cloud.google.com/vertex-ai/generative-ai/docs)
- MediaPipe Pose Landmarker --- extract body landmarks (shoulders, hips, waist) from selfies to infer body shape. (Docs: ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker)
- YOLOv8 (Ultralytics) --- a practical open-source detector for clothes detection, segmentation and fine-tuning on fashion datasets. (Docs: docs.ultralytics.com)
- Firebase Authentication --- recommended for user authentication in the mobile app (email, phone, social logins). (Docs: firebase.google.com/docs/auth)

4. Step-by-step Beginner Guide (MVP)

4. Beginner step-by-step build guide (MVP-focused). Each step assumes you have a GCP project and a basic dev environment (Node.js & Python).

Phase 0 --- Project Setup (1-3 days)

1. Create GCP project and enable APIs (Vision, Vertex AI, Storage). Example gcloud commands:
   - gcloud services enable vision.googleapis.com
   - gcloud services enable aiplatform.googleapis.com
   - gcloud services enable storage.googleapis.com
2. Create a storage bucket: Use Console or `gsutil mb -l us-central1 gs://your-bucket-name`
3. Create service account and grant roles, then download key JSON.
4. Pick a mobile framework (React Native recommended for speed).

Phase 1 --- Image upload & OCR POC (3-10 days)

Goal: Have mobile pick photo -> upload to GCS -> backend runs Vision OCR.

- Mobile: pick image and upload to signed URL (see code snippet).
- Backend: generate signed URL, store metadata, trigger processing on upload (via pub/sub or Cloud Storage event).
- Backend (Python): call Cloud Vision `text_detection` for label/brand extraction.

Phase 2 --- Clothes detection & tagging (7-21 days)

Goal: Tag images with category (top, bottom, dress), color, pattern, and formality.

Options:

- Quick MVP: Use Cloud Vision label detection + heuristics for dominant color (OpenCV).
- Better: Train a Vertex AutoML model (or fine-tune YOLOv8) on fashion dataset (DeepFashion2 / ModaNet) to get higher-accuracy categories and bounding masks.

Store these tags in your DB.

Phase 3 --- Body analysis & personalization (7-14 days)

- Use MediaPipe Pose Landmarker on a selfie to extract key body landmarks.
- From landmarks, derive simple body-shape heuristics: hip/shoulder ratios, torso length, etc.
- Map skin-tone sample from facial area to a friendly "warm/cool/neutral" palette (careful with bias and testing).

Phase 4 --- Recommendation engine (7-21 days)

- Start with a prompt-based approach: feed Gemini a structured prompt (profile + items) and ask for 3 outfit sets in JSON.
- Later: add hybrid ML recommender with embeddings + rules for body-shape constraints.
- Save and use user feedback (worn / saved / skipped) to refine prompts & retrain models.

Phase 5 --- UX, feedback loops & beta (ongoing)

- Build the mobile UI for 5 core screens (onboarding, add-item, home, outfit view, profile).
- Build feedback capture and analytics (suggestion acceptance rate is a vital metric).

- Run a closed beta in Delhi, gather labelled corrections and retrain models.

5. Code Snippet - React Native: pick image and upload to Signed URL

Use this as a beginner example to upload images directly to GCS via a backend-signed URL.

```
// React Native (Expo) - pick image, get signed URL from backend, upload (PUT)
import * as ImagePicker from 'expo-image-picker';
import React from 'react';
import { Button } from 'react-native';

async function pickAndUpload() {
  const res = await ImagePicker.requestMediaLibraryPermissionsAsync();
  if (!res.granted) return alert('Permission required');
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    quality: 0.8,
  });
  if (result.cancelled) return;
  const uri = result.uri;
  const resp = await fetch('https://your-backend/get-signed-url', {
    method: 'POST',
    headers: {'Content-Type':'application/json'},
    body: JSON.stringify({ filename: 'item1.jpg', contentType: 'image/jpeg' })
  });
  const { url } = await resp.json();
  const blob = await (await fetch(uri)).blob();
   await fetch(url, { method: 'PUT', body: blob, headers: {'Content-Type':'image/jpeg'}
});
  alert('Uploaded!');
}
```

6. Code Snippet - Node.js backend: generate Signed URL for uploads

Run this on your backend; never expose your service account key in the mobile app.

```
// Node.js - generate signed URL (server-side) using @google-cloud/storage
const { Storage } = require('@google-cloud/storage');
const storage = new Storage({ keyFilename: '/path/to/service-account.json' });
const bucketName = 'your-bucket-name';

async function generateV4UploadSignedUrl(filename) {
  const options = {
    version: 'v4',
    action: 'write',
    expires: Date.now() + 15 * 60 * 1000, // 15 minutes
    contentType: 'image/jpeg',
  };
  const [url] = await storage.bucket(bucketName).file(filename).getSignedUrl(options);
  return url;
}
```

7. Code Snippet - Python: Cloud Vision OCR

Run this in your backend worker to parse tags/labels printed on clothing items (brand, size, material).

```
# Python - Cloud Vision OCR (server-side)
from google.cloud import vision_v1
client = vision_v1.ImageAnnotatorClient()

def extract_text_from_gcs(gcs_uri):
    image = vision_v1.Image()
    image.source.image_uri = gcs_uri
    response = client.text_detection(image=image)
    texts = response.text_annotations
    return texts[0].description if texts else ''

# Example usage:
# text = extract_text_from_gcs('gs://your-bucket-name/item1.jpg')
# print('OCR text:', text)
```

8. Code Snippet - Python: MediaPipe Pose Landmarker (landmarks for bod

Use pose landmarks to compute shoulder/hip ratios and infer simple body-shape categories.

```
# Python - MediaPipe Pose (simple example)
import cv2
import mediapipe as mp

mp_pose = mp.solutions.pose
mp_drawing = mp.solutions.drawing_utils

image = cv2.imread('selfie.jpg')
with mp_pose.Pose(static_image_mode=True) as pose:
    results = pose.process(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    if results.pose_landmarks:
        for idx, lm in enumerate(results.pose_landmarks.landmark):
            print(idx, lm.x, lm.y, lm.z)
        annotated = image.copy()
                        mp_drawing.draw_landmarks(annotated,   results.pose_landmarks,
mp_pose.POSE_CONNECTIONS)
        cv2.imwrite('annotated_selfie.jpg', annotated)
```

9. Code Snippet - YOLOv8: quickstart for clothing detection & segmentatio

YOLOv8 is useful for bounding boxes and segmentation masks; fine-tune on fashion datasets for best results.

```
# YOLOv8 Quickstart (train & predict) - requires ultralytics package
# pip install ultralytics
from ultralytics import YOLO

# Use a pretrained model for prototype
model = YOLO('yolov8n.pt')

# Inference
results = model.predict('some_item_photo.jpg', save=True)
print(results)

# Training (requires data.yaml describing your dataset)
# model.train(data='data.yaml', epochs=50, imgsz=640)
```

10. Code Snippet - Gemini (Vertex AI): pseudocode & guidance

Use Vertex AI generative models (Gemini) via the Vertex API; call from your backend and parse structured JSON responses.

```
# Pseudocode (server-side flow)
# 1. Build structured prompt with user profile and items.
# 2. Call Vertex AI generative endpoint (REST or SDK).
# 3. Parse JSON output (3 outfit sets) and return to mobile client.

# Example (conceptual):
prompt = {
  "system": "You are a helpful stylist.",
  "user_profile": {...},
  "items": [...]
}
# call Vertex API to generate structured JSON (see Vertex docs for SDK/REST examples)
response_json = call_vertex_gemini(prompt)
# parse response_json and store outfits in DB
```

11. Best Practices & Tips

- Start small and iterate: validate OCR + 1 clothing category, then expand.
- Save user corrections for retraining models.
- Ensure opt-in consent for using images to improve models.
- Always call heavy Google APIs from your backend (not directly from mobile).
- Monitor costs: Vision and Vertex AI can be costly at scale; cache responses where possible.
- Localize: when launching in Delhi, account for Indian garments like kurtas and sarees in your training data.

12. References & Useful Links

Official docs & quickstarts:

- Google Cloud Vision docs (OCR & labels): https://cloud.google.com/vision/docs (Cloud Vision API). citeturn0search0
- Vertex AI Generative Models (Gemini) guide: https://cloud.google.com/vertex-ai/generative-ai/docs citeturn0search21
- MediaPipe Pose Landmarker docs: https://ai.google.dev/edge/mediapipe/solutions/vision/pose_landmarker citeturn0search2
- Ultralytics YOLOv8 docs: https://docs.ultralytics.com citeturn0search3
- Firebase Authentication: https://firebase.google.com/docs/auth citeturn0search4
- YOLOv8 GitHub: https://github.com/ultralytics/ultralytics citeturn0search13
- Vertex AI Gemini quickstart: https://developers.google.com/learn/pathways/solution-ai-gemini-101 citeturn0search16
- DeepFashion dataset (research dataset): http://mmlab.ie.cuhk.edu.hk/projects/DeepFashion.html