1. Code Snippets

Face Detection and Recognition (Using FaceNet & MTCNN)

```
import cv2
import numpy as np
import torch
from facenet_pytorch import MTCNN, InceptionResnetV1

# Initialize face detection and recognition models
mtcnn = MTCNN(keep_all=True)
resnet = InceptionResnetV1(pretrained='vggface2').eval()

def extract_face_embeddings(image_path):
    img = cv2.imread(image_path)
    img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    faces, _ = mtcnn.detect(img_rgb)

    embeddings = []
    for face in faces:
        x1, y1, x2, y2 = map(int, face)
        face_crop = img_rgb[y1:y2, x1:x2]
        face_crop = torch.tensor(face_crop).permute(2, 0, 1).unsqueeze(0).float()
        embedding = resnet(face_crop).detach().numpy()
        embeddings.append(embedding)

    return embeddings
```

Similarity Matching Using Cosine Similarity

```
from sklearn.metrics.pairwise import cosine_similarity

def find_best_match(query_embedding, stored_embeddings):
    scores = cosine_similarity(query_embedding, stored_embeddings)
    best_match_index = np.argmax(scores)
    return best_match_index, scores[best_match_index]

```

FastAPI Backend for Face Search

```
from fastapi import FastAPI, UploadFile, File
import numpy as np

app = FastAPI()

# Load stored embeddings
stored_embeddings = np.load("stored_face_embeddings.npy")

@app.post("/search/")
async def search_face(file: UploadFile = File(...)):
    query_embedding = extract_face_embeddings(file.file)
    best_match, confidence = find_best_match(query_embedding, stored_embeddings)
    return {"match_index": best_match, "confidence": float(confidence)}

```

Streamlit UI for User Interaction

```
import streamlit as st
import requests

st.title("Face Recognition-Based Event Photo Organization")

uploaded_file = st.file_uploader("Upload an image to find similar faces", type=["jpg", "png"])

if uploaded_file is not None:
    files = {"file": uploaded_file.getvalue()}
    response = requests.post("http://localhost:8000/search/", files=files)

    if response.status_code == 200:
        result = response.json()
        st.write(f"Best Match Index: {result['match_index']}, Confidence: {result['confidence']*100:.2f}%")

```

2. Data Sources

- **Training Data**:
  - VGGFace2 Dataset
  - Labeled Faces in the Wild (LFW)
- **Event Media Storage**:
  - Images stored in **Google Drive / AWS S3 / Local Database**.
  - Pre-extracted embeddings stored in **PostgreSQL / SQLite**.

3. Deployment Steps

Deploy Backend with FastAPI (Local & Cloud)

Local Deployment

```bash
uvicorn main:app --host 0.0.0.0 --port 8000
```

Cloud Deployment (Using Docker & AWS)

1. **Create a Dockerfile**

```
FROM python:3.9
WORKDIR /app
COPY requirements.txt .
RUN pip install -r requirements.txt
COPY . .
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

2. **Build and Run Docker Container**

```
docker build -t face-recognition-api .
docker run -p 8000:8000 face-recognition-api
```

3. **Deploy on AWS (Lambda & API Gateway)**

Deploy Streamlit UI

Run Locally

```
streamlit run app.py
```

Deploy on Streamlit Sharing

```
git clone your_repo_url
cd your_repo
streamlit run app.py
```

4. Configurations

PostgreSQL Database Schema for Storing Embeddings

```
CREATE TABLE face_embeddings (
    id SERIAL PRIMARY KEY,
    person_name VARCHAR(255),
    embedding VECTOR(512)
);
```

Load Data into PostgreSQL

```
import psycopg2
import numpy as np

conn = psycopg2.connect("dbname=face_recognition user
    =admin password=secret")
cur = conn.cursor()

def store_embedding(name, embedding):
    cur.execute("INSERT INTO face_embeddings
        (person_name, embedding) VALUES (%s, %s)",
        (name, embedding.tolist()))
    conn.commit()
```