**Face Recognition Attendance System Project Documentation**

**Project Title:**

**Face Recognition-Based Real-Time Attendance System using FaceNet and Firebase**

**Objective:**

To automate the attendance process using real-time face recognition with a webcam. This project captures faces, recognizes individuals using FaceNet embeddings and MTCNN detection, and marks attendance in a Firebase Realtime Database.

**Key Technologies Used:**

* **Programming Language:** Python 3.x
* **Face Detection:** MTCNN (Multi-task Cascaded Convolutional Networks)
* **Face Embedding:** FaceNet (keras-facenet wrapper)
* **Classification Model:** K-Nearest Neighbors (KNN)
* **Database:** Firebase Realtime Database
* **GUI/Display:** OpenCV
* **Data Visualization:** matplotlib, seaborn
* **Audio Notification:** playsound module

**Required Python Packages:**

Ensure the following packages are installed. You can use pip install package\_name to install each:

pip install opencv-python

pip install numpy

pip install mtcnn

pip install keras-facenet

pip install firebase-admin

pip install scikit-learn

pip install playsound

pip install matplotlib

pip install seaborn

**Software Requirements:**

* **Python 3.6+**
* **Any Python IDE or Text Editor** (e.g., PyCharm, VS Code, Jupyter Notebook)
* **Google Chrome or Web Browser** (to check Firebase DB)
* **Internet Connection** (for Firebase communication)

**Hardware Requirements:**

* **Webcam** (built-in or external)
* **System with at least 4GB RAM**

**File/Folder Structure:**

project\_directory/

|— face\_recognizer.pkl # Trained KNN model

|— snapshot/ # Saved snapshots of recognized faces

|— ding.wav # Sound file played on successful recognition

|— facenetDB/ # Directory containing training images

| |— 211FA05146.jpg # Naming convention: rollnumber.jpg

|— face\_attendance.py # Main Python script (your provided code)

|— firebase-adminsdk.json # Firebase credentials JSON file

**How It Works:**

1. **Face Detection:** Detects face in real-time using MTCNN.
2. **Preprocessing:** Lighting normalization using CLAHE.
3. **Embedding:** Converts face to 128-d vector using FaceNet.
4. **Classification:** Predicts identity using a trained KNN model.
5. **Attendance Marking:** Pushes data to Firebase with timestamp.
6. **Snapshot Saving:** Saves the face image with timestamp locally.
7. **Audio Alert:** Plays a sound when a face is recognized.

**How to Train Model:**

* The model is trained automatically if face\_recognizer.pkl is not found.
* Images should be stored in facenetDB/ with filename as the roll number.
* Each image undergoes augmentation to generate 50 variations for training.

**Firebase Setup:**

1. Go to [https://console.firebase.google.com](https://console.firebase.google.com/)
2. Create a new project and enable Realtime Database.
3. Download the admin SDK JSON file and place it in your project directory.
4. Replace the credential path and database URL in the script.

**How to Run:**

python face\_attendance.py

* Press **'x'** to stop webcam and end session.

**Output:**

* Live webcam with name label on recognized faces.
* Attendance entry created in Firebase.
* Local snapshot of the recognized face.

**Notes for Future Students:**

* You can add new students by placing their labeled images in facenetDB/ and retraining.
* Use clear frontal images for better accuracy.
* Firebase email/password auth is not required for Realtime DB.

**Troubleshooting:**

* **No camera detected:** Check webcam permissions.
* **Model not accurate:** Ensure dataset images are high quality and consistent.
* **Firebase error:** Recheck JSON path and DB URL.

**Credits:**

* Open Source libraries: MTCNN, FaceNet, Firebase Admin SDK, Scikit-learn
* Developed by: [Your Name / Team Name]

**CODE:**

import cv2

import numpy as np

from mtcnn import MTCNN

from keras\_facenet import FaceNet

import pickle

import os

import firebase\_admin

from firebase\_admin import credentials, db

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, precision\_score, classification\_report, confusion\_matrix

from datetime import datetime

from playsound import playsound

import matplotlib.pyplot as plt

import seaborn as sns

*# Initialize Face Detector & Embedder*

try:

    detector = MTCNN()

    embedder = FaceNet()

except Exception as e:

    print("Error initializing MTCNN or FaceNet:", e)

    exit()

*# Paths*

DATASET\_PATH = r"D:\mtcnn\facenetDB"

MODEL\_OUTPUT\_PATH = "face\_recognizer.pkl"

SNAPSHOT\_DIR = r"D:\mtcnn\snapshot"

SOUND\_PATH = r"D:\mtcnn\ding.wav"

os.makedirs(SNAPSHOT\_DIR, exist\_ok=True)

*# Firebase Setup*

cred\_path = r"D:\mtcnn\face-recognition-attenda-653fb-firebase-adminsdk-fbsvc-8207c166c6.json"

if not os.path.exists(cred\_path):

    print("Error: Firebase credential file not found.")

    exit()

cred = credentials.Certificate(cred\_path)

firebase\_admin.initialize\_app(cred, {

    'databaseURL': "https://face-recognition-attenda-653fb-default-rtdb.firebaseio.com/"

})

attendance\_ref = db.reference("Attendance")

*# Registered Names*

registered\_names = ["211FA05146", "211FA05189", "211FA05283", "211FA05308", "211FA05309", "221LA05006",

                    "211FA05204", "211FA05252", "211FA05241", "211FA05183", "GSR"]

*# Normalize lighting using CLAHE*

def normalize\_lighting(face):

    lab = cv2.cvtColor(face, cv2.COLOR\_RGB2LAB)

    l, a, b = cv2.split(lab)

    clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8, 8))

    cl = clahe.apply(l)

    lab = cv2.merge((cl, a, b))

    normalized = cv2.cvtColor(lab, cv2.COLOR\_LAB2RGB)

    return normalized

*# Augment image with 50 variations*

def augment\_image(img):

    aug\_imgs = []

    rows, cols, \_ = img.shape

    for i in range(50):

        aug = img.copy()

*# Random horizontal flip with 50% chance*

        if np.random.rand() > 0.5:

            aug = cv2.flip(aug, 1)

*# Random brightness and contrast*

        alpha = 0.8 + np.random.rand() \* 0.6  *# Contrast between 0.8 to 1.4*

        beta = np.random.randint(-40, 40)    *# Brightness between -40 to 40*

        aug = cv2.convertScaleAbs(aug, alpha=alpha, beta=beta)

*# Random rotation (-15 to +15 degrees)*

        angle = np.random.uniform(-15, 15)

        M = cv2.getRotationMatrix2D((cols / 2, rows / 2), angle, 1)

        aug = cv2.warpAffine(aug, M, (cols, rows))

*# Random Gaussian noise*

        noise = np.random.normal(0, 10, aug.shape).astype(np.uint8)

        aug = cv2.add(aug, noise)

*# Random scaling (zoom in/out 0.9 to 1.1)*

        scale = np.random.uniform(0.9, 1.1)

        aug = cv2.resize(aug, None, fx=scale, fy=scale)

*# Crop or pad back to original size*

        if scale < 1.0:

*# pad*

            pad\_h = (rows - aug.shape[0]) // 2

            pad\_w = (cols - aug.shape[1]) // 2

            aug = cv2.copyMakeBorder(aug, pad\_h, rows - aug.shape[0] - pad\_h,

                                     pad\_w, cols - aug.shape[1] - pad\_w,

                                     cv2.BORDER\_REPLICATE)

        else:

*# crop center*

            start\_h = (aug.shape[0] - rows) // 2

            start\_w = (aug.shape[1] - cols) // 2

            aug = aug[start\_h:start\_h+rows, start\_w:start\_w+cols]

        aug\_imgs.append(aug)

    return aug\_imgs

*# Get face embedding*

def get\_embedding(face\_img):

    try:

        face = normalize\_lighting(face\_img)

        face = cv2.resize(face, (160, 160))

        face = np.expand\_dims(face, axis=0)

        embedding = embedder.embeddings(face)

        return embedding[0]

    except Exception as e:

        return None

*# Normalize embedding*

def normalize(embedding):

    return embedding / np.linalg.norm(embedding)

*# Train Model*

def train\_model():

    embeddings = []

    labels = []

    for file in os.listdir(DATASET\_PATH):

        path = os.path.join(DATASET\_PATH, file)

        if os.path.isfile(path):

            img = cv2.imread(path)

            img\_rgb = cv2.cvtColor(img, cv2.COLOR\_BGR2RGB)

            detections = detector.detect\_faces(img\_rgb)

            if detections:

                x, y, w, h = detections[0]['box']

                x, y = max(0, x), max(0, y)

                face = img\_rgb[y:y+h, x:x+w]

                augmented\_faces = augment\_image(face)

                for aug\_face in augmented\_faces:

                    embedding = get\_embedding(aug\_face)

                    if embedding is not None:

                        label = os.path.splitext(file)[0]

                        embeddings.append(normalize(embedding))

                        labels.append(label)

    if embeddings:

        knn = KNeighborsClassifier(n\_neighbors=1, metric='euclidean')

        knn.fit(embeddings, labels)

        with open(MODEL\_OUTPUT\_PATH, 'wb') as f:

            pickle.dump(knn, f)

        predictions = knn.predict(embeddings)

        acc = accuracy\_score(labels, predictions)

        prec = precision\_score(labels, predictions, average='macro', zero\_division=0)

        print(f"Model trained and saved.")

        print(f"Training Accuracy: {acc\*100:.2f}%")

        print(f"Precision Score: {prec\*100:.2f}%")

        cm = confusion\_matrix(labels, predictions, labels=knn.classes\_)

        plt.figure(figsize=(10, 7))

        sns.heatmap(cm, annot=True, fmt='d', xticklabels=knn.classes\_, yticklabels=knn.classes\_, cmap='Blues')

        plt.title("Confusion Matrix")

        plt.xlabel("Predicted")

        plt.ylabel("True")

        plt.tight\_layout()

        plt.show()

        print("\nClassification Report:")

        print(classification\_report(labels, predictions, zero\_division=0))

    else:

        print("No embeddings generated. Check your dataset.")

*# Load Model*

if not os.path.exists(MODEL\_OUTPUT\_PATH):

    print("Model not found. Training a new one...")

    train\_model()

with open(MODEL\_OUTPUT\_PATH, "rb") as f:

    knn = pickle.load(f)

*# Mark Attendance*

def mark\_attendance(name, seen\_names):

    now = datetime.now()

    date\_str = now.strftime("%Y-%m-%d")

    time\_str = now.strftime("%H:%M:%S")

    if name not in seen\_names:

        attendance\_ref.child(date\_str).child(name).set({

            "name": name,

            "time": time\_str,

            "status": "Present"

        })

        seen\_names.add(name)

        print(f"☑️Attendance marked for {name} at {time\_str}")

        playsound(SOUND\_PATH)

*# Mark Absentees*

def mark\_absentees(seen\_names):

    now = datetime.now()

    date\_str = now.strftime("%Y-%m-%d")

    time\_str = now.strftime("%H:%M:%S")

    for name in registered\_names:

        if name not in seen\_names:

            attendance\_ref.child(date\_str).child(name).set({

                "name": name,

                "time": time\_str,

                "status": "Absent"

            })

            print(f"❎Marked {name} as Absent at {time\_str}")

*# Real-Time Frame Processing*

def process\_frame():

    cap = cv2.VideoCapture(0)

    if not cap.isOpened():

        print("Error: Could not open camera.")

        return

    seen\_names = set()

    saved\_names = set()

    print("Press 'x' to exit.")

    while True:

        ret, frame = cap.read()

        if not ret:

            print("Camera read error.")

            break

        rgb = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

        detections = detector.detect\_faces(rgb)

        for det in detections:

            x, y, w, h = det['box']

            x, y = max(0, x), max(0, y)

            face = rgb[y:y+h, x:x+w]

            embedding = get\_embedding(face)

            name = "Unknown"

            if embedding is not None:

                embedding = normalize(embedding)

                name = knn.predict([embedding])[0]

                distance = knn.kneighbors([embedding])[0][0][0]

                if distance > 0.8:

                    name = "Unknown"

                elif name in registered\_names:

                    mark\_attendance(name, seen\_names)

*# Draw box and label*

            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)

            cv2.putText(frame, name, (x, y - 10), cv2.FONT\_HERSHEY\_SIMPLEX,

                        0.8, (0, 255, 0), 2)

*# Save snapshot*

            if name != "Unknown" and name not in saved\_names:

                timestamp = datetime.now().strftime("%Y%m%d\_%H%M%S")

                filename = f"{name}\_{timestamp}.jpg"

                snapshot\_path = os.path.join(SNAPSHOT\_DIR, filename)

                face\_bgr = cv2.cvtColor(face, cv2.COLOR\_RGB2BGR)

                cv2.imwrite(snapshot\_path, face\_bgr)

                saved\_names.add(name)

        cv2.imshow("Face Recognition Attendance", frame)

        if cv2.waitKey(1) & 0xFF == ord('x'):

            break

    mark\_absentees(seen\_names)

    cap.release()

    cv2.destroyAllWindows()

*# Run*

process\_frame()

step-by-step execution guide for your Face Recognition Attendance System Python project, covering setup, training, and real-time execution:

**1. Setup and Initializations**

**a. Import Required Libraries**

* Libraries include OpenCV, NumPy, MTCNN, Keras FaceNet, Firebase Admin, Scikit-learn, etc.
* These are used for:
  + Face detection (MTCNN)
  + Face embedding (FaceNet)
  + Classification (KNN)
  + Attendance logging (Firebase)
  + Snapshot capture, GUI display, and sound alert

**b. Initialize Models**

detector = MTCNN()

embedder = FaceNet()

* MTCNN: Detects face bounding boxes
* FaceNet: Generates 128D face embeddings

**2. Configure Paths and Firebase**

* Define paths to:
  + Dataset folder (DATASET\_PATH)
  + Trained model output file (face\_recognizer.pkl)
  + Attendance sound file (ding.wav)
  + Snapshot saving directory (snapshot)
* Initialize Firebase using your .json credentials to access the realtime database for marking attendance.

**3. Face Preprocessing and Augmentation**

**a. Lighting Normalization**

* Applies CLAHE to enhance local contrast.

**b. Image Augmentation (50 variations per face)**

* Random:
  + Flip
  + Brightness/contrast
  + Rotation
  + Gaussian noise
  + Scaling (zoom in/out)

This increases dataset robustness during training.

**4. Get and Normalize Embeddings**

* Extract 128D vector for each face image using FaceNet
* Normalize embeddings to unit length for distance comparison

**5. Train the Model (Only if Not Already Trained)**

If face\_recognizer.pkl does not exist:

train\_model()

Steps:

1. Read each image in the dataset folder.
2. Detect the face → apply augmentation → extract embeddings.
3. Fit KNN classifier using Euclidean distance.
4. Save the trained model to disk.
5. Show performance using accuracy, precision, confusion matrix, and classification report.

**6. Load Trained KNN Model**

with open(MODEL\_OUTPUT\_PATH, "rb") as f:

knn = pickle.load(f)

**7. Real-Time Attendance Execution**

**a. Start Webcam**

cap = cv2.VideoCapture(0)

**b. For Each Frame:**

1. Detect faces using MTCNN
2. For each detected face:
   * Crop & preprocess
   * Get embedding
   * Predict name using KNN
   * If matched and distance ≤ 0.8:
     + Mark attendance in Firebase
     + Play sound
     + Save snapshot

**c. Exit Condition**

* Press 'x' to exit the camera loop.

**d. Mark Absentees**

* Any name from registered\_names **not seen** is marked **Absent** in Firebase.

**8. Snapshot Saving**

* Captures and saves snapshot of detected faces in the snapshot directory with a timestamped filename.

**9. Audio Notification**

* When a known face is recognized and attendance is marked, it plays a beep sound using playsound().

**Expected Output**

* A live webcam window showing bounding boxes and predicted names.
* Firebase Realtime Database updated with:
  + Name
  + Time
  + Status (Present/Absent)
* Snapshots saved locally
* Sound plays on successful attendance mark
* Print logs in the terminal

**Typical Flow of Use**

1. Place face images (named as roll numbers) in facenetDB/
2. Run the script → it trains if model is absent
3. Launches webcam → detects & recognizes faces
4. Marks attendance in Firebase and saves snapshots
5. Ends when user presses 'x'

**code for attendance dashboard**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0"/>

<title>Attendance Dashboard</title>

<script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-app.js"></script>

<script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-database.js"></script>

<script src="https://www.gstatic.com/firebasejs/8.10.0/firebase-auth.js"></script>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/xlsx/0.17.0/xlsx.full.min.js"></script>

<script src="https://cdnjs.cloudflare.com/ajax/libs/jspdf/2.5.1/jspdf.umd.min.js"></script>

<style>

body {

font-family: Arial, sans-serif;

background: #f4f4f4;

padding: 20px;

}

h1 {

text-align: center;

color: #333;

}

.controls {

display: flex;

flex-wrap: wrap;

justify-content: center;

gap: 10px;

margin-bottom: 20px;

}

button, select {

background: #3498db;

color: white;

border: none;

padding: 10px 20px;

border-radius: 25px;

cursor: pointer;

transition: 0.3s;

}

button:hover, select:hover {

background: #2980b9;

}

.auth-buttons {

position: absolute;

top: 20px;

right: 20px;

}

.auth-buttons button {

margin-left: 10px;

}

.date-container {

margin-bottom: 15px;

background: white;

border-radius: 10px;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

overflow: hidden;

}

.date-header {

background: #3498db;

color: white;

padding: 10px;

cursor: pointer;

}

.date-body {

display: none;

padding: 10px;

}

table {

width: 100%;

border-collapse: collapse;

}

th, td {

padding: 8px;

border: 1px solid #ddd;

text-align: center;

}

.chart-container {

margin-top: 30px;

background: white;

padding: 20px;

border-radius: 10px;

box-shadow: 0 2px 5px rgba(0,0,0,0.1);

}

</style>

</head>

<body>

<h1>Attendance Dashboard</h1>

<div class="auth-buttons">

<button onclick="logout()">Logout</button>

<button onclick="loginPrompt()">Login</button>

</div>

<div class="controls">

<button onclick="downloadCSV()">Download CSV</button>

<button onclick="downloadPDF()">Download PDF</button>

<select id="dateFilter" onchange="applyFilters()">

<option value="all">All</option>

<option value="today">Today</option>

<option value="yesterday">Yesterday</option>

<option value="last3">Last 3 Days</option>

<option value="last7">Last 7 Days</option>

</select>

<select id="nameFilter" onchange="filterByName()">

<option value="">Filter by Name</option>

</select>

</div>

<div id="containerArea"></div>

<div class="chart-container">

<h2 style="text-align:center;">Attendance Summary</h2>

<canvas id="attendanceChart" width="400" height="400"></canvas>

</div>

<script>

// Firebase Config

const firebaseConfig = {

apiKey: "AIzaSyDpbr1mu1bQsdyrCtsXff7029jKDJPxkOU",

authDomain: "face-recognition-attenda-653fb.firebaseapp.com",

databaseURL: "https://face-recognition-attenda-653fb-default-rtdb.firebaseio.com",

projectId: "face-recognition-attenda-653fb",

storageBucket: "face-recognition-attenda-653fb.appspot.com",

messagingSenderId: "55702112426",

appId: "1:55702112426:web:78400ae62e56f9847568d3",

measurementId: "G-Z4XM5JXC59"

};

firebase.initializeApp(firebaseConfig);

const auth = firebase.auth();

const dbRef = firebase.database().ref("Attendance");

const containerArea = document.getElementById("containerArea");

const nameFilter = document.getElementById("nameFilter");

const dateFilter = document.getElementById("dateFilter");

let fullData = {};

let currentFilterName = "";

auth.onAuthStateChanged(user => {

if (user) loadAttendanceData();

else {

// User is not logged in, you might want to display a message or the login prompt directly

console.log("User not logged in");

loginPrompt(); // Show login prompt on page load if not logged in

}

});

function loginPrompt() {

const email = prompt("Email:");

const password = prompt("Password:");

if (email && password) {

auth.signInWithEmailAndPassword(email, password)

.then(() => loadAttendanceData())

.catch(err => alert("Login failed: " + err.message));

} else if (email === null || password === null) {

alert("Login cancelled.");

} else {

alert("Please enter both email and password.");

loginPrompt(); // Re-prompt if fields are empty

}

}

function logout() {

auth.signOut().then(() => {

alert("Logged out");

location.reload();

});

}

function getFormattedDate(offset = 0) {

const d = new Date();

d.setDate(d.getDate() + offset);

return d.toISOString().split('T')[0];

}

function getPastDates(n) {

return [...Array(n)].map((\_, i) => {

const d = new Date();

d.setDate(d.getDate() - i);

return d.toISOString().split('T')[0];

});

}

function loadAttendanceData() {

dbRef.on("value", snapshot => {

fullData = snapshot.val() || {};

populateNameDropdown(fullData);

applyFilters();

});

}

function populateNameDropdown(data) {

const names = new Set();

for (const date in data) {

for (const id in data[date]) {

names.add(data[date][id].name);

}

}

nameFilter.innerHTML = `<option value="">Filter by Name</option>`;

[...names].sort().forEach(name => {

const opt = document.createElement("option");

opt.value = name;

opt.textContent = name;

nameFilter.appendChild(opt);

});

}

function applyFilters() {

const filter = dateFilter.value;

let targetDates = [];

if (filter === "today") targetDates = [getFormattedDate()];

else if (filter === "yesterday") targetDates = [getFormattedDate(-1)];

else if (filter === "last3") targetDates = getPastDates(3);

else if (filter === "last7") targetDates = getPastDates(7);

const filtered = {};

for (const date in fullData) {

if (filter === "all" || targetDates.includes(date)) {

for (const id in fullData[date]) {

const record = fullData[date][id];

if (currentFilterName && record.name !== currentFilterName) continue;

if (!filtered[date]) filtered[date] = {};

filtered[date][id] = record;

}

}

}

renderData(filtered);

}

function filterByName() {

currentFilterName = nameFilter.value;

applyFilters();

}

function renderData(data) {

containerArea.innerHTML = "";

let present = 0, absent = 0;

const dates = Object.keys(data).sort((a, b) => new Date(b) - new Date(a));

dates.forEach(date => {

const box = document.createElement("div");

box.className = "date-container";

const head = document.createElement("div");

head.className = "date-header";

head.textContent = `Date: ${date}`;

head.onclick = () => body.style.display = body.style.display === "none" ? "block" : "none";

const body = document.createElement("div");

body.className = "date-body";

const table = document.createElement("table");

table.innerHTML = `<thead><tr><th>Name</th><th>Time</th><th>Status</th></tr></thead><tbody></tbody>`;

const tbody = table.querySelector("tbody");

for (const id in data[date]) {

const { name, time, status } = data[date][id];

const row = `<tr><td>${name}</td><td>${time}</td><td>${status}</td></tr>`;

tbody.innerHTML += row;

if (status.toLowerCase() === "present") present++;

else absent++;

}

body.appendChild(table);

box.appendChild(head);

box.appendChild(body);

containerArea.appendChild(box);

});

updateChart(present, absent);

}

function updateChart(present, absent) {

const ctx = document.getElementById('attendanceChart').getContext('2d');

if (window.attendanceChart) window.attendanceChart.destroy();

window.attendanceChart = new Chart(ctx, {

type: 'pie',

data: {

labels: ['Present', 'Absent'],

datasets: [{

data: [present, absent],

backgroundColor: ['#2ecc71', '#e74c3c']

}]

}

});

}

function downloadCSV() {

let csv = "Date,Name,Time,Status\n";

for (const date in fullData) {

for (const id in fullData[date]) {

const { name, time, status } = fullData[date][id];

csv += `${date},${name},${time},${status}\n`;

}

}

const blob = new Blob([csv], { type: 'text/csv' });

const a = document.createElement("a");

a.href = URL.createObjectURL(blob);

a.download = "attendance.csv";

a.click();

}

async function downloadPDF() {

const { jsPDF } = window.jspdf;

const doc = new jsPDF();

doc.text("Attendance Report", 20, 20);

let y = 30;

for (const date in fullData) {

doc.text(`Date: ${date}`, 10, y);

y += 10;

for (const id in fullData[date]) {

const { name, time, status } = fullData[date][id];

doc.text(`- ${name} | ${time} | ${status}`, 10, y);

y += 8;

if (y > 280) {

doc.addPage();

y = 20;

}

}

}

doc.save("attendance\_report.pdf");

}

</script>

</body>

</html>

* save this file as foldername.html
* keep the file in the public folder