Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**
Subject: Capstone Project	**Aim: Implementation - Continuous progress review**
Date: 24-9-2025	**Enrolment No: 92310133004**

1. Abstract

This document explains how the Attendance System was implemented using AWS — a web application that enables faculty to upload a single classroom image containing multiple students, automatically detect and recognize faces, and produce an Excel report that lists the students who are present and absent. The system is built with a React front-end, a Flask backend (using Python), and integrates AWS services such as S3 for storage, Rekognition for face detection and matching, DynamoDB for storing metadata, and optional SQS and Lambda for handling asynchronous processing. The implementation focuses on writing clean code, maintaining modularity, handling errors effectively, including tests, and providing clear, reproducible steps for deployment.

2. System Overview & Objectives

Primary objective: Provide a low-effort workflow for marking attendance from class images: faculty uploads an image and the system returns an Excel sheet with Present/Absent lists.

Functional requirements implemented:

- Upload a classroom image via web UI.
- Detect and recognize multiple faces in uploaded image.
- Match recognized faces against registered student faces stored in AWS Rekognition Collection.
- Produce an Excel (.xlsx) file with two sheets: Present and Absent (or one sheet with status column).
- Backend APIs implemented in Flask, frontend in React, everything tied together securely.

3. High-Level Architecture

Below is a simple diagram showing component interactions (Mermaid syntax for easy rendering):

4. Technology Stack

- **Backend:** Python 3.10+, Flask, boto3 (AWS SDK), openpyxl (Excel), Pillow (image ops), face-processing libs (optional: MTCNN/FaceNet locally for fallback).
- **Frontend:** React (Vite or Create React App), Fetch/axios, UI library optional (Tailwind or Material UI).
- **AWS Services:** S3 (image storage), Rekognition (collection for registered faces), DynamoDB (student metadata), IAM for secure credentials. Optional: Lambda for serverless matching, SQS for work queue.

Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**
Subject: Capstone Project	**Aim: Implementation - Continuous progress review**
Date: 24-9-2025	**Enrolment No: 92310133004**

5. Code Structure (recommended)

```
attendance-system/
├── backend/
│   ├── app.py # Flask app (entrypoint)
│   ├── requirements.txt
│   ├── config.py # AWS config, env loader
│   ├── services/
│   │   ├── s3_service.py
│   │   ├── rekognition_service.py
│   │   ├── dynamo_service.py
│   │   └── excel_service.py
│   ├── models/
│   │   └── student.py
│   ├── tests/
│   └── utils/
├── frontend/
│   ├── src/
│   │   ├── components/UploadForm.jsx
│   │   └── api.js
├── docs/
└── README.md
```

6. Key Implementation Details

6.1 Registration flow

- Faculty/admin register students by uploading individual face images (or via webcam). Each registration call:
    - Uploads the student's face image to S3.
    - Calls Rekognition IndexFaces to add face(s) to a Rekognition Collection and stores the returned FaceId with the student metadata in DynamoDB ({student_id, name, face_id, roll_no}).

6.2 Attendance marking (image upload)

- Frontend submits multipart/form-data POST /upload with image, class_id, subject, date.
- Backend app.py flow:
    1. Save image temporarily and upload to S3.
    2. Call Rekognition DetectFaces (optional) or directly SearchFacesByImage against the

Collection to find matching FaceIds.

3. For each matched face, lookup student metadata in DynamoDB.
4. Build two lists: present (matched) and absent (registered students not matched).
5. Use openpyxl to create an Excel with Present and Absent sheets (or a single sheet with status). Save file to /tmp and upload to S3 (optional) or stream back.

6.3 Matching thresholds and edge cases

- Use configurable Rekognition similarity threshold (e.g., 85%). Provide logic for duplicate matches, low-confidence matches (flag as unknown), and logging.
- If Rekognition returns no matches for a face, you can optionally run a local face detector/embedding to attempt a match as fallback.

7. API Endpoints (example)

- POST /api/register — Register a student (name, roll_no, image). Returns student id.
- POST /api/upload — Upload class image to mark attendance. Request: multipart form (+ class_id, date). Response: download link or binary Excel file.
- GET /api/students?class_id=... — Get registered students for the class.

8. Testing

Unit tests

- Test utility functions (image pre-processing, Excel writer) via pytest.
- Mock boto3 clients using moto or unittest.mock to simulate Rekognition/S3/DynamoDB.

Integration tests

- Create integration tests that run POST /api/upload with sample images and a seeded Rekognition collection (or mocked) and assert that resulting Excel has correct present/absent entries.

Manual tests & evidence

- Include screenshots of the React upload page, server logs showing matched FaceIds, and the generated Excel file.

9. Error handling & Security

- Validate uploaded file types and size limits.
- Use try/except blocks around all AWS calls; return HTTP 5xx for service failures with meaningful messages.
- Store AWS credentials in environment variables or use IAM roles (recommended). Do not hardcode keys.
- Use HTTPS between frontend and backend (production) and enable CORS only for allowed origins.

Marwadi University Marwadi Chandarana Group	**Marwadi University** **Faculty of Engineering and Technology** **Department of Information and Communication Technology**
Subject: Capstone Project	**Aim: Implementation - Continuous progress review**
Date: 24-9-2025	**Enrolment No: 92310133004**

10. Deployment & Running Instructions

Prerequisites: Python 3.10+, Node 18+, AWS account with Rekognition permissions, S3 bucket, DynamoDB table, Rekognition collection created. (IAM)

Backend:

1. cd backend && python -m venv && source venv/bin/activate.
2. pip install -r requirements.txt.
3. Set environment variables: AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, AWS_REGION, REKOG_COLLECTION_ID, S3_BUCKET, DYNAMODB_TABLE.
4. Python main.py

Frontend**:**

1. cd frontend && npm install.
2. Configure API base URL in src/api.js.
3. npm run dev (or npm build)