Attendance Management System

This project was done as part of integrated Mtech(2021 batch) course curriculum for the course "Software Engineering - CS_303" under the guidance of the professor Sujit Kumar Chakrabarti

Prepared by:

- Sunny Kaushik (IMT2021007)
- Samarjeet Wankhade (IMT2021013)
- Saanvi Vishal (IMT2021043)
- Varshith Sai Prasad Vattikuti (IMT2021078)

I. Overview

The Attendance Management System is an advanced, facial recognition-based solution for tracking attendance in organisational environments. It is designed to automate and streamline the attendance process, offering an efficient alternative to traditional methods. Key features include automated attendance recording using facial recognition technology, a user-friendly interface for managing attendance data, and robust data handling capabilities. The system targets enhanced operational efficiency and accuracy, catering to the needs of administrators and users alike.

II. Stakeholders

1. Users: The end-users of the system, who will interact with it for attendance tracking.
2. Testers: Ensure the system's reliability and performance through rigorous testing.
3. Organisations: The primary target audience, needing a system to track their employees' attendance.

III. Objectives

1. Provide a convenient and portable solution for personnel responsible for tracking attendance within an organisation.
2. Enable these personnel to maintain and manage attendance records and essential student information efficiently using a computer.
3. Offer various functionalities like editing attendance, viewing statistics, and making notes related to attendance-related elements.
4. Streamline the attendance tracking process for students by implementing a Facial Recognition System, thereby eliminating the hassles associated with traditional attendance methods

IV. Components

1. Face Recognition and Attendance
    a. Uses OpenCV and face recognition to detect faces in real-time using a webcam feed.
    b. Loads pre-stored encodings and student IDs from the encoding file.
    c. Matches detected faces with known encodings to recognize students.
    d. Accesses Firebase to fetch student information and update attendance records based on face recognition results.

2. Image processing and encoding
    a. Utilises OpenCV (cv2) and the face_recognition library to process images and generate face encodings.
    b. Manages Firebase connections to upload images to Firebase Storage and store associated student IDs and their corresponding encodings in a file.

3. Database population and modification
    a. Initializes Firebase and establishes a connection to the database.
    b. Populates the Firebase database with student information such as names, majors, attendance records, etc., using predefined data.

IV. Detailed design

1. Data Structures

a. Lists: Lists are used to store the list of student IDs, image list, student info list, the list of known encodings and many kind of other lists and array type data structures.
b. Dictionaries: Dictionaries are used to store student information and to store the mapping between student IDs and their corresponding known encodings for images. This can also include other Firebase encodings used by google.
c. NumPy arrays: NumPy arrays are used to store images and face locations.
d. Firebase Data Student Information: Student information and attendance details are retrieved from the Firebase Realtime Database.
e. Firebase Data Student Information: Image data is retrieved from Firebase Storage.
f. Other Firebase data Structures for calling APIs and invoking firebase services.

2. Algorithms

a. This projects needs the usage of the following algorithms:
b. Face recognition: The code uses the face recognition to recognize faces in images and hence is an image recognition algorithm.
c. Image processing: The code might need to use OpenCV to manipulate images, such as resizing, converting colour spaces, and drawing rectangles and text for UI design and other Interface functionalities.
d. Database operations: The code uses the Firebase Realtime Database API to store and retrieve student information.
e. Interface switching will itself use many algorithms in order to change the display mode based on certain conditions, altering the appearance of the graphical user interface (GUI).
f. File I/O: The project needs to save the face encodings maybe to a file.

3. APIs

a. face_recognition: The face_recognition library provides functions for face detection, face recognition, and face distance calculation.

b. OpenCV: OpenCV is a library for computer vision and machine learning.
c. Firebase Realtime Database: The Firebase Realtime Database is a NoSQL cloud database that allows you to store and synchronize data in real time.

V. File descriptions:

1. Main.py

➔ *Application Initialization:*
- main.py initiates the application's execution. It starts by setting up necessary configurations, possibly reading from service AccountKey.json for any required authentication or database connection details.

➔ *Workflow Coordination:*
- It directs the application's overall workflow, deciding the order and conditions under which different scripts are executed. This involves calling functions from encoder.py for data processing and AddDataToDatabase.py for database interactions, based on the application logic.

➔ *Data Management:*
- Handles initial data input, which could be from user input, files, or external sources, and manages its flow through the system. It passes this data to encoder.py for necessary processing and subsequently to AddDataToDatabase.py for storage or further manipulation.

➔ *Error Handling and Logging:*
- Implements robust error handling mechanisms to manage exceptions and ensure smooth application operation. This may include logging errors for debugging and maintenance purposes.

➔ *Interscript Communication:*
- Facilitates communication and data transfer between scripts. It ensures that data is appropriately passed between encoder.py and AddDataToDatabase.py, managing inputs and outputs from these modules.

➔ *User Interface and Output:*
- Depending on the application's nature, main.py might also handle user interactions, displaying outputs, and responding to user inputs. It acts as the bridge between the backend processes and the user interface.

2. Encoder.py

➔ *Firebase Initialization:*
- credentials.Certificate("serviceAccountKey.json"): Loads the Firebase credentials from a service account key file.

- firebase_admin.initialize_app(cred, {...}): Initializes the Firebase application with the given credentials and configuration, such as the database URL and storage bucket.

➔ *Image Importing:*

- os.listdir(folderPath): Lists all files in the specified directory ('Images').
- The script reads each image in the directory using OpenCV's cv2.imread and stores them in imgList. Corresponding student IDs are inferred from the file names and stored in studentIds.

➔ *Firebase Storage Upload:*

- storage.bucket() and blob.upload_from_filename(fileName): Uploads each student's image to the Firebase storage in a folder named 'Images'. This part of the script is for storing the images in the cloud, which is not necessarily real-time but provides centralized access to the data.

➔ *Face Encoding:*

- face_recognition.face_encodings(img)[0]: This function from the face_recognition library is used to generate the face encodings. Face encoding is a process of converting facial features into a numerical array that uniquely identifies an individual's face.
- The findEncodings function iterates over each image, converts it from BGR to RGB (since face_recognition uses RGB), and computes the face encodings.

➔ *Encoding Storage*:

- pickle.dump(encodeListKnownWithIds, file): This part of the script uses the pickle module to serialise the face encodings along with student IDs and save them to a file (EncodeFile.p). This file can be used later to identify students by comparing encodings from new images to these stored encodings.

3. Add to Database:

➔ *Data Insertion*: Facilitates the addition of new records into the database, ensuring data integrity and consistency.

➔ *Data Update*: Allows for modifications to existing records, maintaining the database's relevance and accuracy.

➔ *Data Retrieval*: Enables querying the database to fetch required data, which is then used for further processing or display within the application.

➔ *Error Handling*: Incorporates robust error handling mechanisms to manage database connection issues, query failures, and other potential anomalies.

➔ *Scalability and Performance*: Optimised for handling large volumes of data with minimal performance impact, ensuring the application remains responsive and efficient.

4. Service Account Key:

➔ *Authentication Credentials*: Stores essential credentials such as API keys, client IDs, and secret tokens, which are used to authenticate the application with external services.

➔ *Security*: As it contains sensitive information, this file is handled with high security, ensuring that the credentials are not exposed or misused.

➔ *Configuration Data*: May also include various configuration settings necessary for the application to correctly interface with other services, like database URLs, service endpoints, or access permissions.

➔ *Environment-Specific Settings*: Often used to differentiate between different environments (development, testing, production), allowing for seamless transitions and consistent behaviour across various deployment stages.

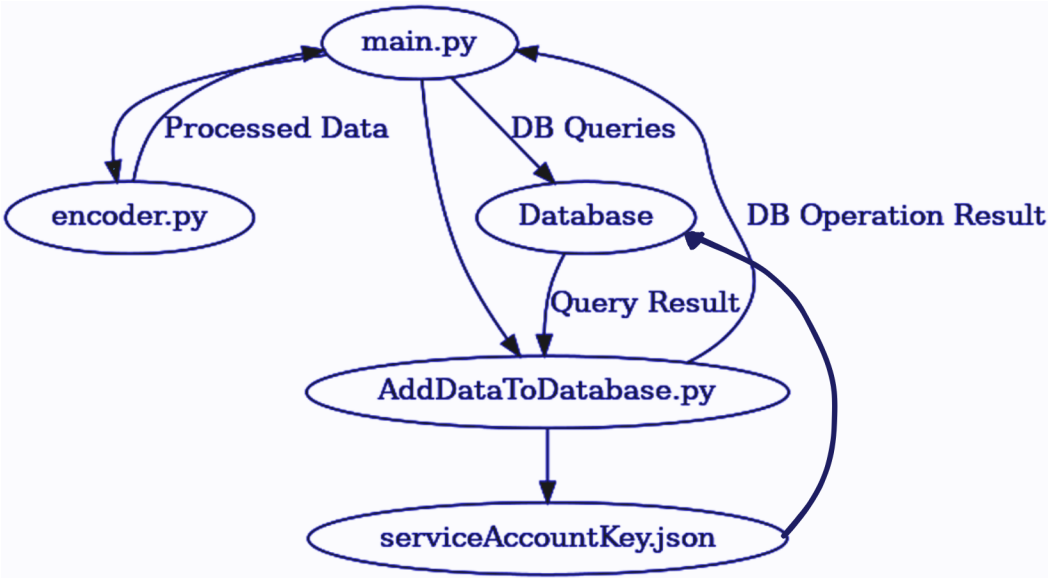
➔ *Access Control*: Typically restricted in terms of access, with its usage and modifications limited to authorised personnel to maintain security and integrity.

VI. Integration and Workflow

1. Starting with main.py:
    a. The main.py script acts as the entry point of the application. It initiates the process and orchestrates the flow of data between other components.
    b. It might begin by fetching or receiving data that needs to be processed. This data could come from a user input, a file, or an external source.

2. Data Processing in encoder.py:
    a. The raw data from main.py is then passed to encoder.py.
    b. The encoder.py script is responsible for encoding the data, transforming it into a format suitable for database storage. This could involve data sanitization, format conversion, compression, or encryption.
    c. Once the data is encoded, it's sent back to main.py.
3. Data Management in AddDataToDatabase.py:
    a. Next, main.py interacts with AddDataToDatabase.py, passing the processed data to it.
    b. AddDataToDatabase.py handles all database operations. It uses the credentials and configuration details stored in serviceAccountKey.json to securely connect to the database.
    c. The script then performs operations like inserting, updating, or retrieving data from the database as required.
4. Using serviceAccountKey.json for Secure Access:
    a. The serviceAccountKey.json file is crucial for authenticating database operations. It likely contains API keys or database credentials.
    b. AddDataToDatabase.py reads these credentials to establish a secure connection to the database. This ensures that all database transactions are authenticated and authorised.
5. Feedback Loop to main.py:
    a. After the database operations are complete, AddDataToDatabase.py may send a confirmation, result, or any retrieved data back to main.py.
    b. main.py could then use this information for further processing, to trigger other actions, or to provide output to the user or an external system.

6. Error Handling and Logging:
    a. Throughout this process, error handling is critical. Each component, especially main.py and AddDataToDatabase.py, likely includes mechanisms to handle exceptions and log errors, possibly using information from serviceAccountKey.json for secure logging services.

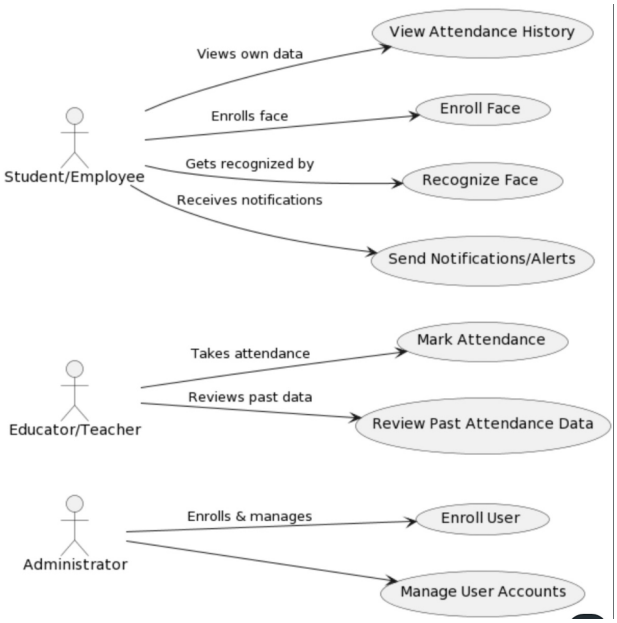


VII. Implementation Details

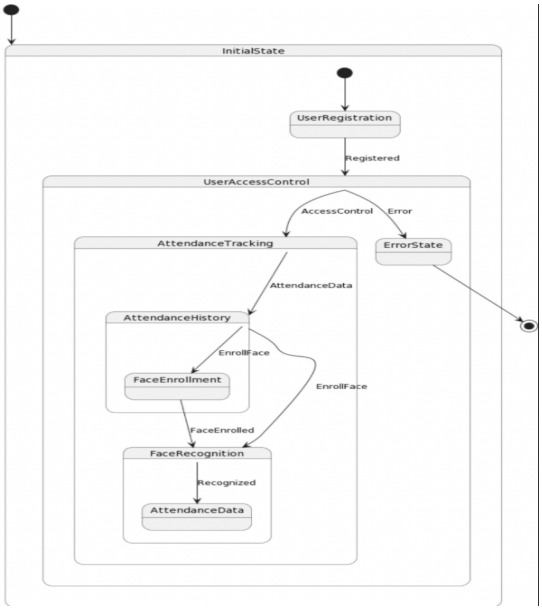
1. Programming Language and Version:
    - The scripts are developed in Python. It's important to specify the exact Python version used (e.g., Python 3.8) to ensure compatibility.
2. Dependencies and Libraries:
    - The project likely uses several external libraries or frameworks. The specifics of these dependencies should be documented, typically in a requirements.txt file, and include libraries for database connection, data encoding, and other functionalities.
3. Configuration and Setup:
    - The serviceAccountKey.json file is used for storing configuration settings and sensitive information, such as API keys or database credentials, crucial for the application's interaction with external services or databases.
    - Detailed instructions for setting up the development environment, including installing Python, setting up a virtual environment, and installing necessary packages, should be provided.
4. Script Functionalities:
    - AddDataToDatabase.py: This script handles database operations such as inserting, updating, and retrieving data. It uses connection information and credentials possibly stored in serviceAccountKey.json.
    - encoder.py: Focuses on encoding and processing data. This could involve converting data formats, sanitising inputs, or preparing data for database insertion.
    - main.py: Acts as the main entry point for the application, orchestrating the workflow by calling functions from the other scripts and managing the overall process flow.

VIII. UML Diagrams

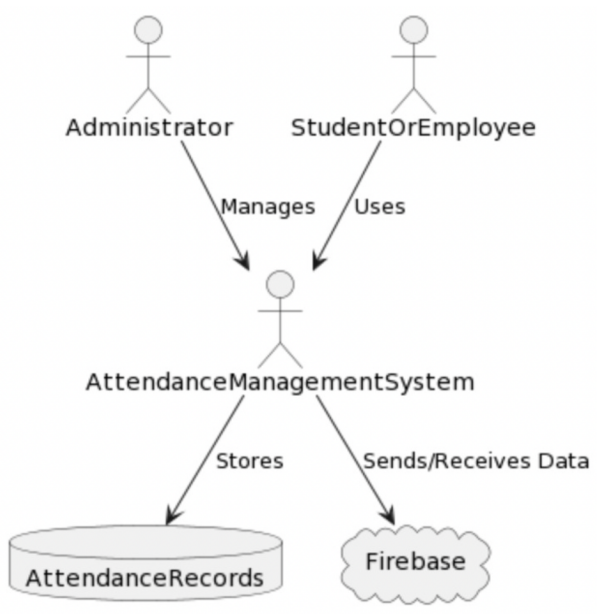
- *Use Case Diagram :*



- *State Flow Diagram:*



- *Data Flow Diagram:*



IX. Challenges Faced

1. The major challenge faced during the working with this project was the model to capture the faces, as we had two option either to train the Machine Learning model, doing which could have been made this project even better and fruitful. We were allowed to use the APIs available, hence the following API's have been used:
   In the provided code:
   - ➔ *OpenCV API (cv2):* Utilized for webcam access, image manipulation, resizing, and video display.
   - ➔ *Firebase Admin SDK:* Employed for interaction with Firebase services like Realtime Database and Storage.
   - ➔ *face_recognition API:* Used for face detection, face recognition, and encoding face features.

2. The second major challenge was the storage and retrieval of the image that was needed to be stored in the database efficiently. Deciding the optimal method to store images while ensuring retrieval for real-time face recognition was crucial, to overcome this we used image resizing as the it was taking considerable amount of time to run as the sizes of images can be upto 10-20 MB.

3. Implementing secure communication protocols with Firebase, securely storing credentials, and adopting Firebase's protocols ensure secure access.

Each challenge was addressed through a combination of strategic decisions, optimizations, and meticulous attention to design and implementation details, ensuring the robustness, efficiency, and reliability of the face recognition and attendance tracking system.

X. Testing

1. Introduction: Face recognition is a critical component in applications dealing with identity verification. This report evaluates the unit testing code for face recognition, covering the detection and identification of faces using the face_recognition library. Another Important component is Firebase. Integration testing is a critical phase in the software development lifecycle, focusing on verifying the interactions between different components of a system. This report evaluates the integration testing code for Firebase integration, which includes tests for uploading and downloading images to/from Firebase Storage and inserting data into the Firebase Realtime Database.

2. The test class TestFirebaseIntegration contains two methods: test_upload_and_download_image, and test_data_insertion. The setUpClass method is responsible for initializing the Firebase app for testing, while the other two methods perform specific integration tests and tear down class method is used to cleanup after testing is done.
    a. Test_upload_and_download_image: This test case verifies the functionality of uploading and downloading images to/from Firebase Storage.
        i. Test Steps: Upload a test image to Firebase Storage. Download the uploaded image. Assert that the uploaded image exists in Firebase Storage. Assert that the uploaded image has a non-zero size.
        ii. Assertions: assertTrue(blob.exists()): Verifies that the uploaded image exists, assertTrue(blob.size > 0): Ensures that the uploaded image has a non-zero size.
    b. test_data_insertion:This test case validates the insertion of data into the Firebase Realtime Database.
        i. Test Steps:Insert test data into the 'Students' node in the database. Retrieve the inserted data from the database. Checks if the inserted data is present in the retrieved data.
        ii. Assertions: assertIn("987654", retrieved_data): Verifies that the inserted data key is present in the retrieved data, assertEqual(data["987654"], retrieved_data["987654"]): Ensures that the inserted data matches the retrieved data.

3. The test class FaceRecognitionTest contains two test methods: test_detect_face and test_identify_face. The setUp method initializes a known face encoding used for comparison in the test_identify_face method.
    a. test_detect_face:This test case focuses on verifying the functionality of face detection.
        i. Test Steps:Load an image containing a face. Detect the face in the image. Check that only one face is detected.
        ii. Assertions: assertEqual(len(face_locations): Ensures that only one face is detected in the image.
    b. Test_identify_face: This test case validates the identification of a known face in an image.
        i. Test Steps: Load an image containing a known face. Detect the face in the image. Encode the face. Compare the face encoding to the known face encoding. Check that the face is correctly identified.
        ii. Assertions: assertTrue(is_match[0]): Verifies that the identified face matches the known face encoding.

XI. Individual Contribution

1. Samarjeet Wankhade:
   Played a significant role in conducting unit tests and integration tests for various modules and components. Assisted in setting up and integrating the database, focusing on storing images and establishing the necessary mechanisms.

2. Sunny Kaushik:
   Collaborated closely with Varshith to develop appropriate modes and design the user interface (UI). Contributed to the database setup and integration within the application, ensuring its functionality. All the retrievals and sending information which was details associated with each images was handled.

3. Saanvi Vishal:
   Successfully implemented encryption for image files by encoding each image, ensuring secure transactions within the application. Played key role in the smooth retrieval and utilization of encoded images during application runtime.

4. Varshith:
   Coordinated with Sunny to design the UI, crafting user-friendly windows and establishing diverse modes for the application. Worked on optimizing window sizes and background design to seamlessly integrate the camera window and student details, a crucial aspect of the final product.