Project: Attendance Management System (deliverable 2)

Team Members:

1. **Sunny Kaushik (IMT2021007)**
2. **Samarjeet Wankhade (IMT2021013)**
3. **Saanvi Vishal (IMT2021043)**
4. **Varshith (IMT2021078)**

Components:

1. Face Recognition and Attendance

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

2. Image processing and encoding

- Utilizes OpenCV (**cv2**) and the **face_recognition** library to process images and generate face encodings.
- 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

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

Connectors:

1. Firebase Connector

- **Related Component:** All three parts of the code interact with Firebase services for database management and storage.
- **Description:** The **firebase_admin** library establishes a connection to Firebase using the provided service account key (**serviceAccountKey.json**). It interacts with Firebase's Realtime Database (**db**) and Storage (**storage**) modules.

- **Relationship with Components:**

1. In the second part, images are uploaded to Firebase Storage.
2. In the third part, student data is pushed to the Firebase Realtime Database.
3. The first part doesn't directly interact with Firebase but relies on the data and encodings generated by the second part, which involves Firebase interactions.

2. Encoding Connector

- **Related Component:** The first and second parts of the code utilize OpenCV (**cv2**) extensively for image processing and manipulation.
- **Description:** OpenCV is employed for webcam access, image reading, resizing, conversion to RGB format, and other image-related operations.

- **Relationship with Components:**

1. The second part processes images to generate face encodings.
2. The first part continuously captures frames from the webcam and performs face recognition tasks.

3. Pickle Connector

- **Related Component:** The second part uses the **pickle** module for serialization and deserialization of face encodings and student IDs.

- **Description:** Pickle facilitates the saving and loading of Python objects (encoding data in this case) to/from a file.

- **Relationship with Components:**

1. The second part stores the generated face encodings and associated student IDs using Pickle.
2. The first part loads these encodings and student IDs for face recognition tasks.

Mapping of NFRs and FRs to connectors and Components:

→ Connectors:

- The **Firebase Connector** serves as a bridge between all three parts of the system and the Firebase services, ensuring that data is securely managed and stored in the cloud.
- The **Encoding Connector**, which relies on OpenCV, is essential for image processing and face encoding, linking the facial recognition functionality with the system's ability to recognize and verify user identities.
- The **Pickle Connector** is used for serialization and deserialization of face encodings and is crucial for the system's ability to quickly match detected faces with stored encodings, affecting the system's performance and user experience.

→ Components

- **Facial Recognition Component:**
  - This component is directly linked to FRs regarding real-time detection and identification of individuals through facial recognition technologies. It captures live images, processes them to detect faces, and matches them against stored encodings.
  - The efficiency and accuracy of this component also address NFRs like system performance and usability, as it must process images quickly and accurately to be effective.

- **Attendance Component:**
  - The Attendance component is responsible for the FR of recording and tracking attendance. It interacts with the Database to store attendance records and provides functions to retrieve attendance history.
  - It also impacts NFRs such as reliability and maintainability. The system must reliably record attendance each time, and be maintainable in terms of updating attendance records and correcting any discrepancies.

- **Database Component:**

- Central to both NFRs and FRs, the Database component is where all user and attendance data is stored and managed. It ensures data integrity and consistency, which are crucial NFRs, by providing a structured way to access and manipulate records.
- This component is also essential for scalability, a key NFR, allowing the system to manage a growing amount of data without loss of performance.

- **Student and Administrator Components:**
  - These user-related components are aligned with FRs such as user registration, data review, and account management. They define specific roles within the system and tailor functionality to these roles.
  - NFRs such as security and privacy are also addressed here, as these components must ensure that user data is handled securely and personal information is kept private.

- **User Component:**
  - This is a broader component representing any system user and is involved in common FRs like authentication and data access. It provides generic functionalities that other specific user roles can extend.
  - In terms of NFRs, the User component is crucial for user experience. It must be designed to ensure ease of use and accessibility, providing a seamless interaction with the system.

Architecture Used:

Layered architecture, also known as the "n-tier architecture," involves organizing an application into distinct layers, each responsible for specific functionalities. Each layer communicates with adjacent layers, maintaining separation of concerns and modularity. Let's break down how the provided code aligns with a layered architecture:

1. Presentation Layer: The webcam feed for the face recognition will act as the presentation layer displaying real- time face recognition results and overlays the student information onto the video feed.
2. Application Layer: This layer deals with the part where the main logic/part of the code will interact with the firebase to fetch the student information and update the attendance records based on the face recognition results. This layer also handles the creation and structuring of the student data before pushing it to the firebase.
3. Data Layer: Utilizes firebase for both real time database interaction and storage handling.

The presentation layer interacts with the application layer by triggering face recognition tasks, accessing data from Firebase and displaying results. The application layer interacts with the data layer to push and fetch student data and images.

Important design decisions:

- Modularity and separation of concerns: The code demonstrates a clear separation of responsibilities among its components. One part manages the real-time face recognition, another structures student data for database insertion, and another interacts exclusively with the database for population. This division allows for independent development and maintenance of each module. It simplifies debugging, updates, and scalability since changes in one part are less likely to impact the others.
- Use of API and external libraries: The code integrates external services and libraries to leverage their specialized functionalities. It makes use of Firebase for efficient cloud-based storage and database management. Additionally, it taps into the power of OpenCV and the **face_recognition** library, harnessing their face detection and recognition capabilities for attaining a better accuracy.
- Serialization with pickle: Serialization with **pickle** enables the storage of complex Python objects, such as face encodings and student IDs, into a file. This choice simplifies data storage and retrieval processes, allowing for easy preservation and retrieval of important information without needing to handle complex data structures manually.

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.

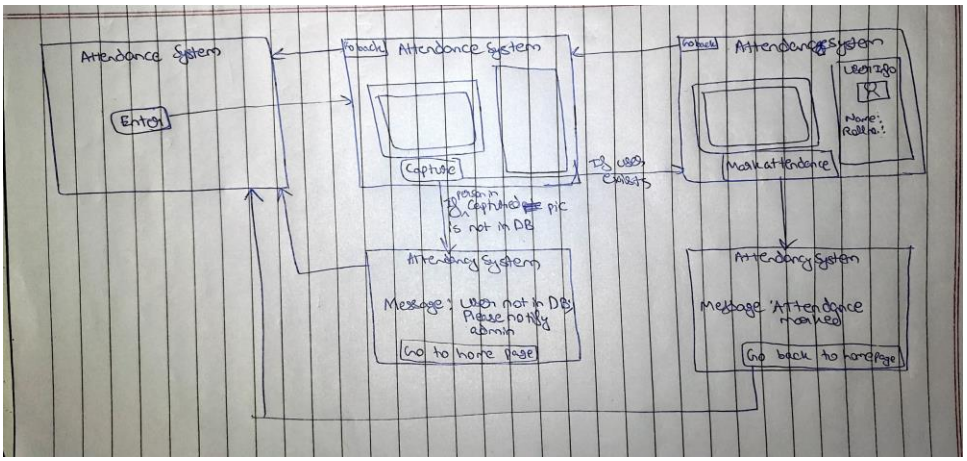
2. Algorithms

a. This projects needs the usage of the following algorithms:
b. Face recognition: The code uses the face recognition recognize faces in images and hence is a image recognition algorithm.
c. Image processing: The code might need to use OpenCV to manipulate images, such as resizing, converting color 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 on order to change the display mode based on certain condition, 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.

Wireframe:



Click [here](#) to view it on Figma