Optimizing Event Photo Management Through Automated Facial Recognition Technology

*

1st Muhammad Tahmidur Rahman
Dept. of Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
Dhaka, Bangladesh
Email: muhammad.rahman12@northsouth.edu

2nd Mohosina Islam Disha
Dept. of Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
Dhaka, Bangladesh
Email: mohosina.disha@northsouth.edu

3rd Anika Tabassum Enan
Dept. of Electrical and Computer Engineering
North South University, Dhaka, Bangladesh
Dhaka, Bangladesh
Email: anika.tabassum23@northsouth.edu

I. Abstract

This project addresses the challenges of organizing event photos by introducing an automated system that categorizes photos based on facial recognition. The solution includes a web platform and a mobile application, providing users with flexible access to their sorted photos. Using tools such as OpenCV [1], Firebase [2], and the `face_recognition` library [3], the system detects and processes photos, identifying individuals and automatically sorting images into personalized folders. Each participant receives a dedicated folder containing all photos where their face appears, eliminating the need to sift through irrelevant images. The system employs Haar Cascades for face detection [4] and embeddings generated by the `face_recognition` library [3] to ensure accurate facial identification. Firebase is integrated for real-time database management and secure photo storage [2], ensuring scalability and convenience. This comprehensive solution simplifies photo retrieval, enhances user satisfaction, and establishes a new standard in event photo organization.

II. Introduction

A. Problem Definition

Event attendees often capture a multitude of photos using personal devices or rely on professional photographers for high-quality coverage. However, the post-event photo retrieval process remains cumbersome, fragmented, and time-consuming. In large gatherings—such as conferences, weddings, corporate seminars, or social meetups—participants frequently share their images through multiple channels, including messaging apps (WhatsApp, Messenger, Telegram) and social media platforms. This uncoordinated dissemination leads to a chaotic landscape where valuable memories are scattered across diverse storage points, inaccessible or hard to find for those who want them.

The current dependency on manual sifting, tagging, and request-based sharing exacerbates the problem. Participants often find themselves waiting for friends or photographers to upload specific images or spending hours browsing through irrelevant photo albums to locate their own pictures. In professional event settings, organizers may host hundreds of images online without any efficient search or sorting mechanism, forcing attendees to scroll endlessly. This lack of organization diminishes user

satisfaction, reduces engagement, and may even undermine the perceived quality of the event. Ultimately, there is a pressing need for a system that can streamline retrieval, ensuring that every attendee can quickly and easily access photographs featuring them, thereby preserving memories more effectively and enhancing the overall event experience.

B. Proposed Solution

To tackle these challenges, we propose an automated photo management system specifically tailored for events. The core idea is to utilize advanced facial recognition technology to detect, identify, and categorize individuals across all event photos. The system comprises both a dedicated web platform [5] and a mobile application [6], enabling participants to interact with the solution through their preferred medium. By integrating secure user authentication and encouraging participants to upload a profile image at the outset, the platform establishes a reliable baseline for facial recognition.

When event hosts or organizers upload a batch of photos, the system runs them through a pipeline that includes face detection (using Haar Cascades integrated with OpenCV [1]) and face embedding generation (via the `face_recognition` library [3]). The embeddings are then matched against a database of known participants. The technology stacks seamlessly with Firebase [2], which manages user profiles, event rooms, and large-scale storage. Once processed, the system automatically sorts images into individual user folders, accessible through their authenticated accounts. A host-triggered sorting feature ensures that the entire process requires minimal manual intervention. Participants no longer need to request images from others; they simply log in and find all relevant photos in a single curated location. This approach not only accelerates retrieval but also reduces frustration and enhances user satisfaction. The proposed solution sets a new standard in event photo organization by merging intelligent face recognition, robust cloud-based architecture, and an intuitive user interface, ensuring a more efficient and pleasant post-event experience for all stakeholders.

C. Previous Work

Previous efforts in photo categorization and management have largely centered on single-user contexts, as exemplified by platforms like Google Photos and Apple Photos, which employ sophisticated machine learning models to cluster faces and streamline personal photo libraries [7], [8]. While these solutions excel in individual environments, they remain less optimized for multi-user, event-specific settings. On the other hand, event photo management tools such as Shutterfly and SmugMug [9], [10] rely heavily on manual organization, tagging, and user input. These strategies, although intuitive, are labor-intensive and do not scale efficiently to larger events.

Research in facial recognition—driven by algorithms like Viola-Jones [4], FaceNet [11], and DeepFace [12]—has achieved near-human accuracy under controlled conditions, providing a solid foundation for automated identification. In addition, cloud-based services like Firebase Realtime Database and Storage [2], [13] make it feasible to maintain large, dynamic datasets and process images rapidly. Yet, the integration of these technologies into a unified solution that addresses the distinct challenges of event photography—high photo volume, multi-user environments, variable image quality, and participant authentication—remains limited.

D. Research Gaps

Although existing tools and research have laid the groundwork for advanced facial recognition and scalable cloud infrastructures, several gaps persist. Many state-of-the-art facial recognition algorithms have not been fully optimized for the dynamic conditions found in events, where lighting, angles, and participant diversity can severely impact accuracy. Moreover, popular consumer platforms do not inherently support role-specific access controls or provide automatic sorting triggers that adapt to event workflows. The absence of openly available, high-performance APIs and frameworks that can seamlessly integrate with event management tasks also hinders accessible, cost-effective adoption. Consequently, there is a need for a solution that bridges these gaps, providing both technological sophistication and event-oriented usability.

E. Research Contributions and Novelties

Our proposed system offers a range of distinctive features and contributions that set it apart from existing approaches:

- **Holistic Multi-Platform Integration:** Unlike many existing systems that focus primarily on mobile or desktop solutions, our framework delivers a cohesive experience across both platforms [5], [6]. Event participants can seamlessly interact with the system, whether they prefer a web interface or a mobile app, ensuring maximum accessibility and convenience.
- **Verified Participant Profiles and Authentication:** By leveraging Firebase's secure authentication mechanisms and requiring profile images [2], our system establishes an authenticated baseline. This ensures higher accuracy in facial recognition by linking distinct visual identities to verified user accounts, significantly reducing misidentification risks.
- **Role-Based Access and Permissions:** Events often involve multiple stakeholders (hosts, guests, administrators). Our platform differentiates user roles and privileges, allowing hosts to create rooms, initiate sorting requests, and manage large photo sets, while guests have restricted, yet user-friendly, access to their own curated collections. This granular control streamlines the user experience and maintains data integrity.
- **Automated Classification and Minimal Manual Intervention:** The system's sorting process can be triggered simply by updating a database field, leveraging Firebase Realtime Database triggers [2]. This automated workflow significantly reduces the manual effort required post-event, enabling real-time or scheduled classification without human bottlenecks.
- **Manual Participant Enrollment and Personalized Communication:** Hosts can add participants from the web interface by supplying necessary details and a profile image, ensuring everyone is registered even if they did not self-enroll. Hosts also have the ability to send targeted emails—potentially including secure links to sorted images—fostering efficient dissemination of results and enhancing participant satisfaction.
- **Robust Backend Processing With Clear Metrics:** Utilizing OpenCV for face detection [1], [20] and the `face_recognition` library [3] for generating embeddings ensures swift and accurate processing. The backend's design offers transparent progress indicators, accuracy metrics, and confusion matrices, enabling continuous monitoring, performance optimization, and quality assurance.
- **Scalability and Reliability Through Cloud Infrastructure:** By employing Firebase for database management and storage [2], [13], our solution ensures stable and responsive data handling. Automatic cleanup mechanisms, fallback procedures, and dynamic triggers guarantee that as the event size and complexity grow, the system can effortlessly scale, maintaining responsiveness and reliability.

III. Literature Review

The landscape of photo management and facial recognition has evolved significantly over the past decade. Originally, personal photo organization relied heavily on manual efforts or simplistic metadata-based indexing. With the advent of machine learning and computer vision techniques, applications like Google Photos and Apple Photos introduced automated clustering and face recognition, enhancing user convenience [7], [8]. These platforms leverage deep neural networks to analyze visual features, group similar faces, and suggest tags, thereby simplifying the retrieval of personal memories. However, their underlying models and interfaces are inherently optimized for single-user libraries, typically a single person's or family's collection, rather than for large-scale events involving numerous attendees.

Event-oriented photo solutions—such as those provided by Shutterfly and SmugMug—mark an initial step toward multi-user scenarios [9], [10]. They facilitate image sharing and allow community members to contribute their photos to a central repository. Yet, these tools predominantly depend on manual tagging, human curation, and rudimentary categorization. For relatively small events, manual organization may suffice, but as the number of participants and images grows exponentially, this

approach quickly becomes impractical. Manually curated galleries become overwhelming, and attendees often struggle to locate their specific photos among a sea of irrelevant images.

In the realm of face detection and recognition, the foundational work by Viola and Jones [4] has set the stage for real-time face detection. This approach introduced Haar-like features and AdaBoost-based learning, which paved the way for cascading classifiers. Building on these foundations, OpenCV's implementation of Haar Cascades [**?**] provides ready-to-use, pretrained models that can detect faces quickly even on resource-limited devices. Furthermore, higher-level Python libraries like `face_recognition` [3] simplify the process of face embedding generation and comparison, allowing developers to integrate robust face recognition functions with minimal effort. The `face_recognition` library, which leverages the power of deep learning (e.g., via FaceNet-like embeddings), enables systems to achieve near-human-level accuracy under favorable conditions.

Subsequent algorithms such as FaceNet [11] and DeepFace [12] further improved embedding generation, achieving robust recognition even in challenging conditions. Despite these advancements, event environments differ significantly from controlled datasets: lighting fluctuations, motion blur, partial occlusions, and diverse vantage points complicate the recognition task. While newer models and techniques (e.g., MTCNN [15] and RetinaFace [16]) show improved resilience, practical event deployment requires careful pipeline structuring, fine-tuning of parameters, and potentially hybrid models that balance speed and accuracy.

Cloud-based infrastructures like Firebase [2], [13] have transformed data handling and scalability. For large events, where thousands of images must be processed rapidly, traditional on-premises solutions often falter. Firebase's Realtime Database and Cloud Storage offer reliable synchronization, secure authentication, and real-time triggers that can seamlessly initiate recognition jobs as soon as images are uploaded. This synergy between vision algorithms and scalable backend services allows event administrators to maintain responsiveness even under heavy loads.

Nevertheless, integrating advanced facial recognition into large-scale, multi-user event scenarios remains relatively unexplored. While Lee and Kim [14] have evaluated the performance of various facial recognition models for real-time applications, there are still trade-offs between accuracy, computational complexity, and deployment feasibility at scale. User experience also plays a pivotal role. Developing intuitive interfaces, enabling semantic queries (e.g., "Show me photos of Alice during the keynote speech"), and ensuring participants can quickly find their own images remain active challenges. Although literature on scalable multimedia retrieval [18] and natural language-based retrieval [19] provides valuable insights, adapting these solutions specifically to event contexts with numerous participants and heterogeneous imagery is an open research frontier.

In conclusion, while the literature on facial recognition and photo management is extensive, most existing solutions and studies do not fully address the distinct requirements of large, multi-user events. The incorporation of Haar Cascades and user-friendly tools like the `face_recognition` library help streamline face detection and recognition tasks. Cloud-based services ensure scalability and continuous availability. By combining these advances into a cohesive, role-based, and fully automated pipeline tailored for event photo management, our work strives to fill these critical gaps. We leverage proven face detection methods (Viola-Jones, Haar Cascades), robust embedding techniques (FaceNet, DeepFace), and scalable backend infrastructures (Firebase), building upon the existing literature to create a holistic solution that meets the complexity, diversity, and dynamic nature of real-world event scenarios.

IV. Methodology

The following is the methodology, which includes how we have executed the entire project.

A. Completed Workflow of Our Project

In this section, the completed system workflow of our project is presented. This workflow diagram contains three figures showing the:

1. App and Database workflow 1
2. Web and Database workflow 2
3. Model Backend and Database workflow 3 4

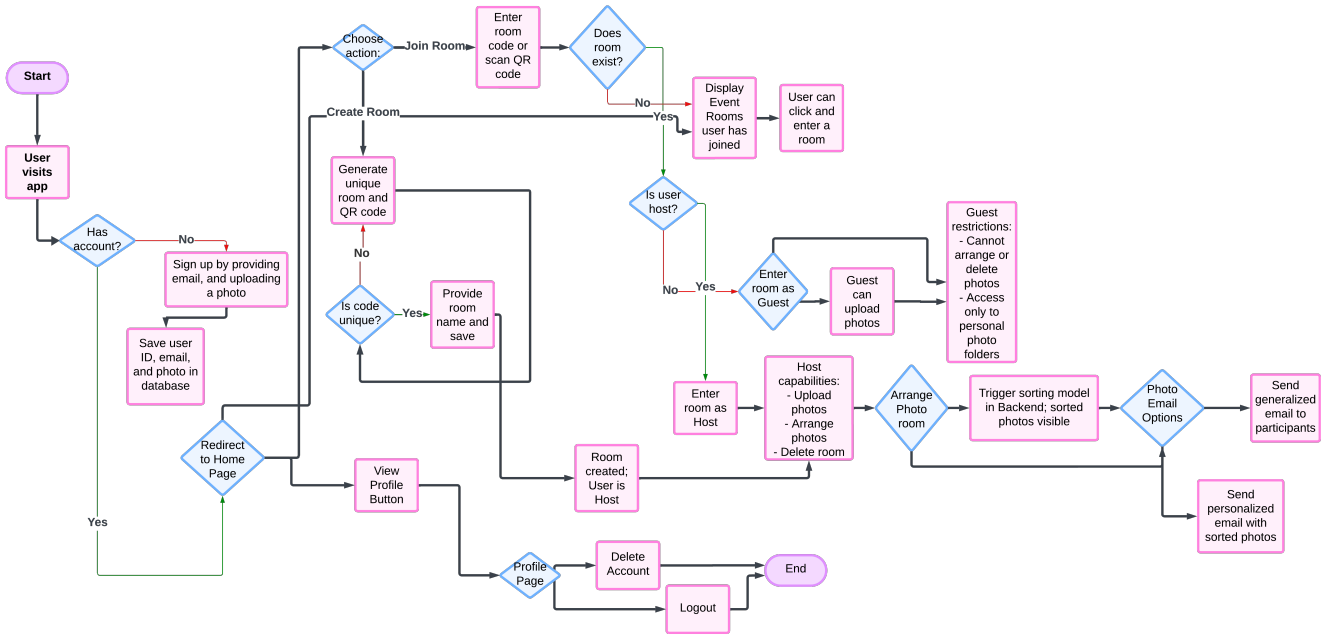

Fig. 1: App Flow Diagram

This App Flow Diagram 1 contains the entire system workflow for our Pictora [6] App, which describes the following features that our app provides:

Key Features for Pictora [6] App

1. **Signup and Login:**
   - Users can sign up using Google Authentication or a valid email address.40
   - After successful signup, users can log in to access the app.41
   - If a user forgets their password, they can reset it by clicking on ”Forgot Password.”42
2. **Photo Upload Requirement:**
   - Users must upload a photo during signup to access the app’s forward stack.
3. **Forward Stack:**
   - **View Event Rooms:**
     - Users can see the list of event rooms they are part of.44
   - **Room Creation:**
     - Users can create a room with a single click.
     - Each room is assigned a unique QR code and text code.
     - During creation, the user (host) must enter an event room name, which becomes permanent upon confirmation and cannot be changed. 4546
   - **Host Features:**
     - Hosts can control the arrangement of photos (images get sorted in the backend).
     - Hosts can delete a room entirely.
     - Hosts can upload photos to the room.
     - Hosts can view:
       - All guest folders.
       - Both uploaded (event room) and sorted (arranged photo room) images of themselves and all guests.48
   - **Room Joining:**
     - Users can join a room as a guest by either typing the room code manually or scanning the QR code.47
   - **Guest Features:**
     - Guests can only upload photos and view their sorted photos.
     - Guests cannot view photos uploaded by the host or other guests.49
4. **User Management:**
   - Users can change their name.
   - Users can delete their account entirely.

- Users can log out of the system.43

5. **Email Notifications:**
   - After images are sorted, the host can:
     - Send a generalized email to all participants in the room, prompting them to check the app for their photos.54
     - Send personalized emails to individual participants.

     53
   - Hosts can also send emails to manually added participants (added via the web platform) with their sorted images attached as a downloadable zip folder.
   - The sorted image zip folder will be attached automatically to the client's default email app for personalized email sending.54
6. **Photo Download:**
   - Both guests and hosts can download their sorted photos and uploaded photos as a zip folder.52

Web Flow Diagram:

The following Web Flow Diagram contains:

the entire web [5] work/system flow, which describes the following key features:

Key Features for Pictora Web:

1) **Signup and Login:**
   - Users can sign up using Google Authentication or a valid email address.27
   - After successful signup, users can log in to access the web interface.28
   - If a user forgets their password, they can reset it by clicking on "Forgot Password."28
2) **Photo Upload Requirement:**
   - Users must upload a photo during signup to access the forward stack of the app.27
3) **Forward Stack:**
   - **Join Event Room:**
     - Any user can join an event room by entering a valid app-generated text code.31
   - **Host Features:**
     - If the user is the host of the room (i.e., they previously created the room in the app):

       a) **Add Manual Guests:**

       ∗ The host can add manual guests by clicking "Add Member" and entering their name, email, and a single image.

       ∗ The host can also delete manual guests if needed.32

       b) **Control Photo Arrangement:**

       ∗ The host can click "Arrange Pho-

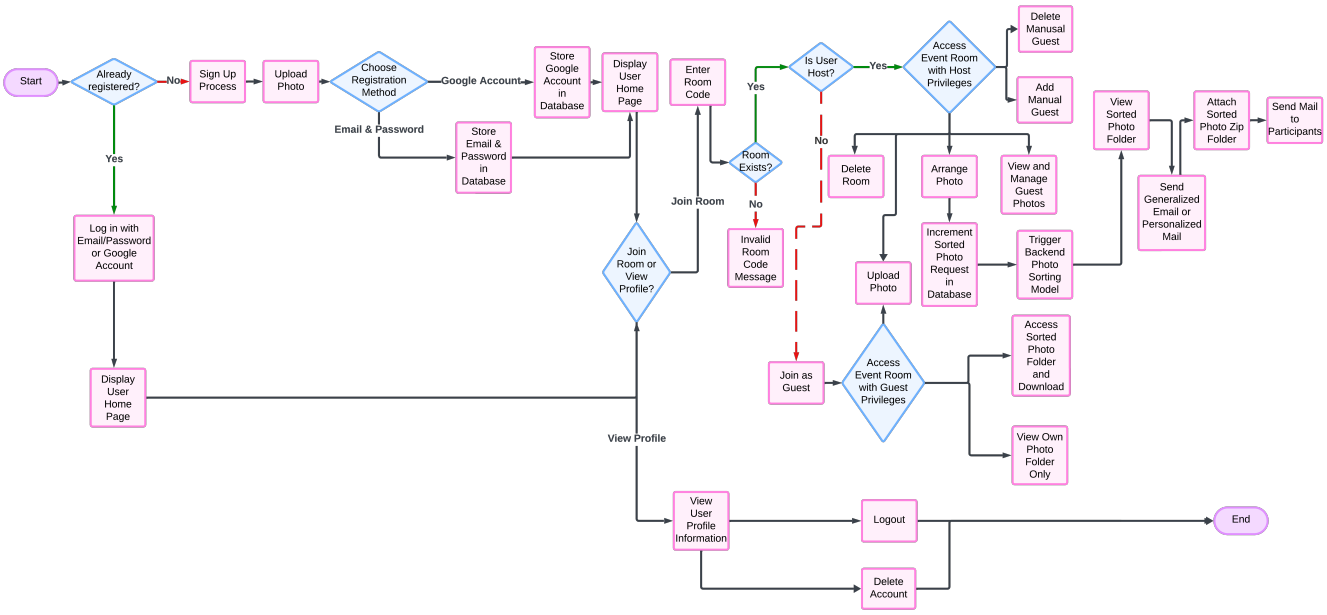

Fig. 2: Web Flow Diagram

tos" to sort images in the backend.35

c) **Upload Photos:**
   * The host can upload photos to the room.32

d) **View Guest Folders:**
   * The host can view all guest folders and see both:
     · Uploaded (event room) photos.
     · Sorted (arranged photo room) photos of the host and all guests.3235

4) **Guest Features:**
- When a user joins a room as a guest:
  – They can upload photos.
  – They can view only their sorted photos.
  – Guests cannot see photos uploaded by the host or other guests.33

5) **User Management:**
- Users can change their name.
- Users can delete their account entirely.
- Users can log out of the system.30

6) **Photo Download:**
- Any user can download their photos as a zip folder from the web interface.36

7) **Email Notifications:**
- After images are sorted:
  – The host can send a generalized email to all participants, prompting them to check the app for their photos.37
  – The host can send personalized emails to manual participants with their downloadable sorted zip file attached.38
- When the host clicks "Send Email," the system will automatically open the client's default email application with a pre-formatted email.
- The host can send the email to the designated addresses with a single click.

Backend Flow Diagram:

The following Backend Flow Diagram 3 4 contains:

the entire backend [21] work/system flow, which describes the following key features:

Key Features for Pictora Backend:

1) **Firebase Initialization and Integration:**
- Securely authenticates and connects to a Firebase Realtime Database and Cloud Storage bucket using a service account JSON file.
- Accesses and manipulates data in specified Firebase database references.
- Downloads and uploads image files (photos) to and from Firebase Storage.

2) **Face Detection and Recognition:**
- Uses OpenCV's Haar Cascade to detect faces in images.
- Utilizes the `face_recognition` library to generate face embeddings (numerical vectors) for each detected face.
- Compares newly processed photo embeddings against stored participant profile embeddings to identify matches.

3) **Participant Embedding Verification:**
- Iterates over participant data retrieved from the database.
- Downloads each participant's profile image, crops the face, generates an embedding, and stores these embeddings back into the database.

4) **Photo Identification Workflow:**
- Retrieves uploaded event photos from host and participant folders in Cloud Storage.
- Processes each image by detecting faces and extracting embeddings.
- Compares these embeddings against previously verified participant embeddings to identify the subject(s) in the photos.
- If a match is found, organizes the photo into the corresponding participant's folder. If no match is found, moves the photo to an "unmatched" folder for review.

5) **Progress and Accuracy Reporting:**
- Displays a dynamic progress bar on the console as images are processed.
- Calculates and prints out accuracy metrics and a confusion matrix after processing all photos.
- Logs matched, unmatched, and total processed photos.

6) **Local File Management:**
- Stores downloaded images temporarily in a

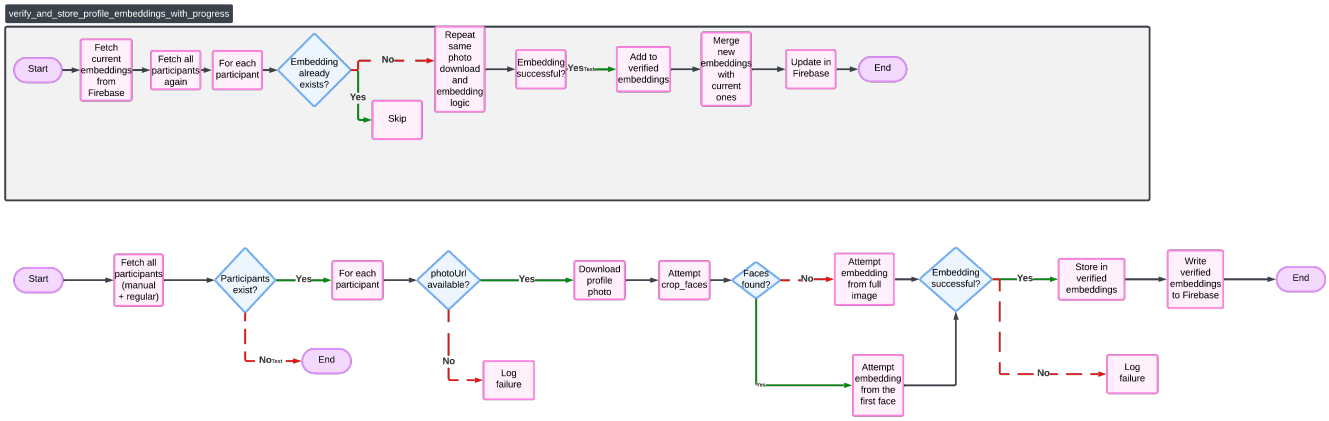

Fig. 3: Backend Flow Diagram for Embeddings

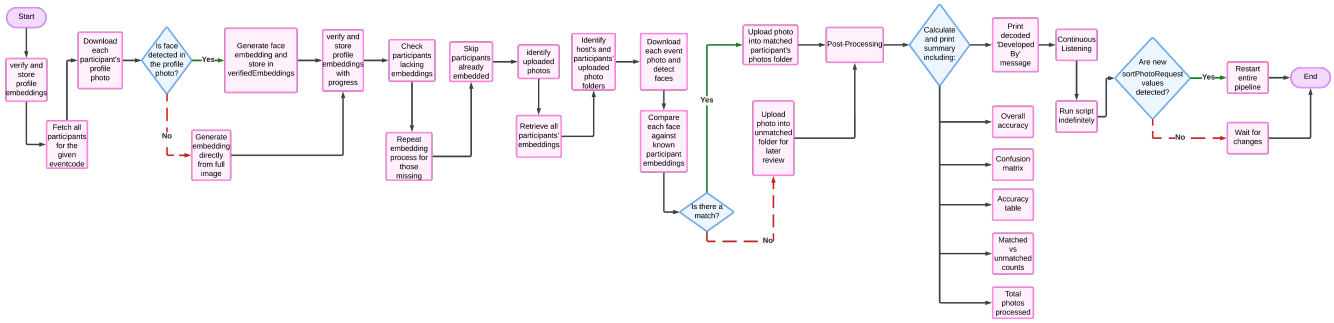

Fig. 4: Backend Flow Diagram for Face Matching

local directory.
- Cleans up local files after processing to avoid clutter.

7) **Dynamic Triggering via Database Events:**
- Listens for changes to `sortPhotoRequest` values in the Firebase Realtime Database.
- Starts the entire face recognition and sorting process automatically whenever a new request is detected that is higher than the previously processed value.

8) **Fallback and Error Handling:**
- Includes retry logic to handle temporarily locked or unavailable files.
- Prints informative error messages and handles exceptions gracefully.
- Uses fallback logic (e.g., attempting to generate embeddings from the full image if no faces are found).

9) **Data Persistence for Progress:**
- Saves and retrieves the last processed `sortPhotoRequest` per room in a temporary JSON file.
- Prevents re-processing the same request multiple times.

10) **Final Summary and Information Output:**
- After completing the facial recognition and sorting process, prints a summary of the results.
- Decodes and prints an encrypted developer message.

B. Dataset and Problem Details

1. Dataset Description

We have utilized two distinct test datasets wherein North South University students shared their outside hangouts and in-campus hangout images.

NSU GROUP IMAGES DATASET

This is our primary test dataset, encompassing images from various participants, including hosts, guests, and manually added guests.

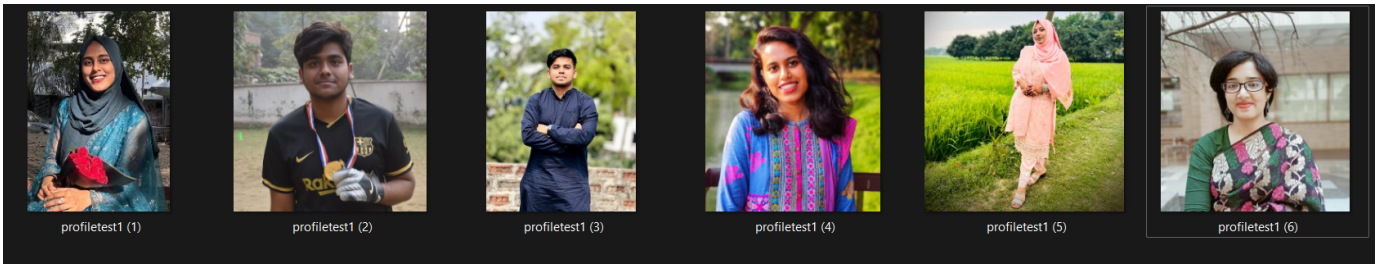

Fig. 5: Profile Images of Test Dataset 1 Participants

a. Source of Data:
The dataset comprises a collection of images sourced from a private Hangout group consisting of North South University students. These images were shared among group members before, during, and after events, capturing various moments, interactions, and individual appearances.

To structure our datasets effectively, we created two types of rooms: *Room 1: NSU GROUP IMAGES* and *Room 2: NSU GROUP IMAGES Test 2*. In *Room 2*, we introduced slightly different images to observe the variety in results and evaluate the differences between matched and unmatched outputs. This was achieved by maintaining consistent profile photos for some common participants and introducing different profile images for one participant to assess differences or similarities. Specifically, in **Test 1**, the host uploaded a total of 26 images 7, while two participants uploaded 12 and 8 images respectively 89, resulting in a total of 46 images stored in Firebase Storage for testing. In **Test 2**, the host uploaded 8 images 10, and two participants uploaded 7 and 9 images respectively 1112, totaling 24 images stored in Firebase Storage for testing.

Test 1 for *Room 1: NSU GROUP IMAGES* includes six participants' profile photos 5, encompassing the host, guests, and manually added guests, which were added by the host 3a via the website.

Test 2 for *Room 2: NSU GROUP IMAGES Test 2* comprises five participants' profile photos 6, including the host, guests, and manually added guests, added by the host 3a through the website.

b. Dataset Composition:

- **Number of Images:** A total of 70 images were collected across both datasets, reflecting a diverse range of scenarios typical in group settings.
- **Image Formats:** All images are in JPEG and PNG formats, ensuring compatibility with the processing libraries employed.
- **Resolution:** Images possess varying resolutions, ranging from low-quality snapshots taken with mobile devices to higher-resolution photos captured by dedicated cameras.
- **Diversity:** The dataset includes photos taken under different lighting conditions, angles, and environments, such as indoor gatherings, outdoor events, and candid moments.

2. Problem Context

a. Challenge of Manual Photo Management:
Managing and organizing a large volume of event photos manually is time-consuming and prone to errors. Participants often struggle to locate specific images featuring themselves or others, leading to frustration and inefficiency in sharing memories.

b. Technical and Operational Challenges:
Despite the potential of automated facial recognition, several challenges hinder its effectiveness:

- **File Format Limitations:** We are unable to process HEIC files due to the absence of dedicated GPUs in our machines, which prevents the installation of the `pyheif` library necessary for handling this format [21].
- **Image Quality Issues:** Our datasets yield unmatched outputs under extreme low-light conditions and when photos are taken at unconventional angles, compromising the accuracy of facial recognition.
- **Backend Infrastructure Constraints:** The backend of our system is hosted on a local machine rather than on dedicated servers. This

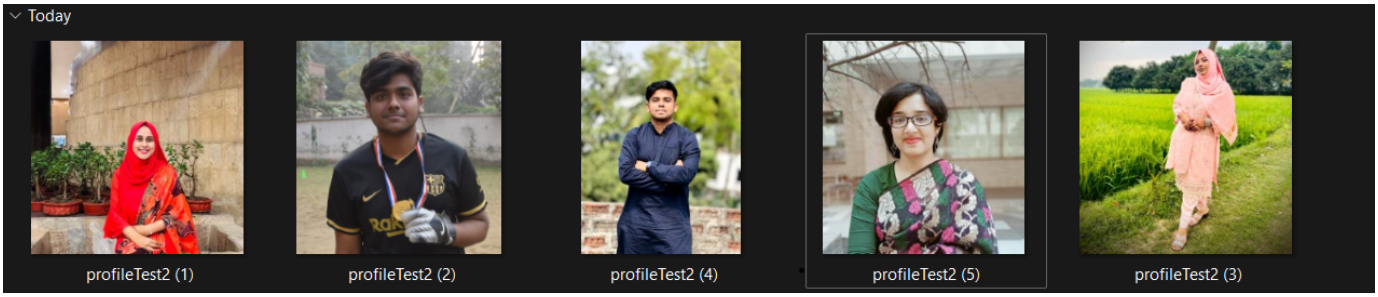

Fig. 6: Profile Images of Test Dataset 2 Participants

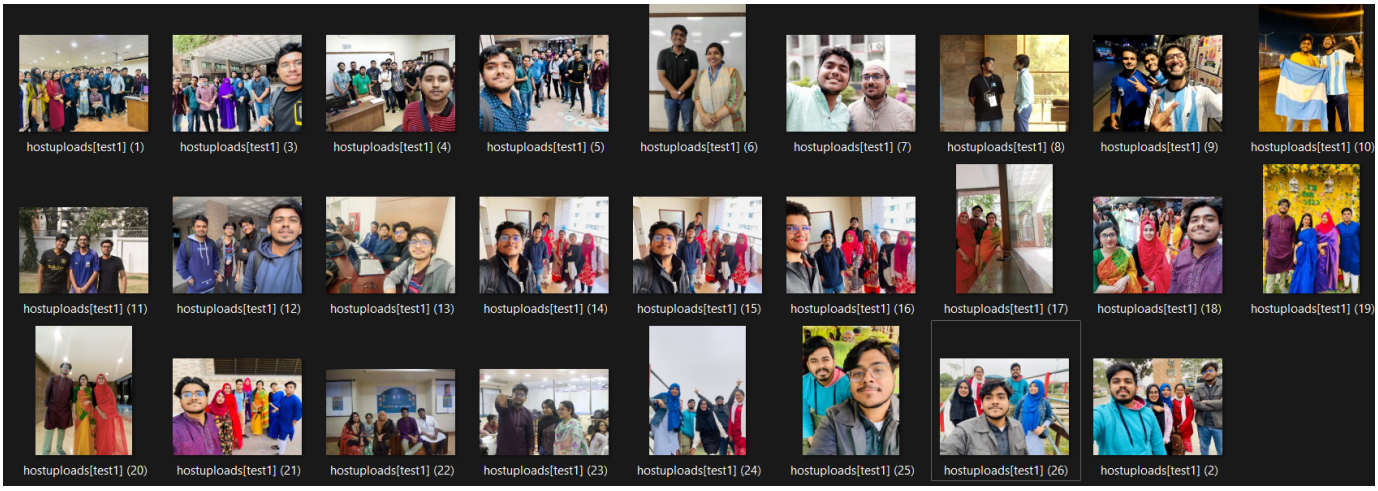

Fig. 7: Host Uploaded Images for Test 1 Dataset

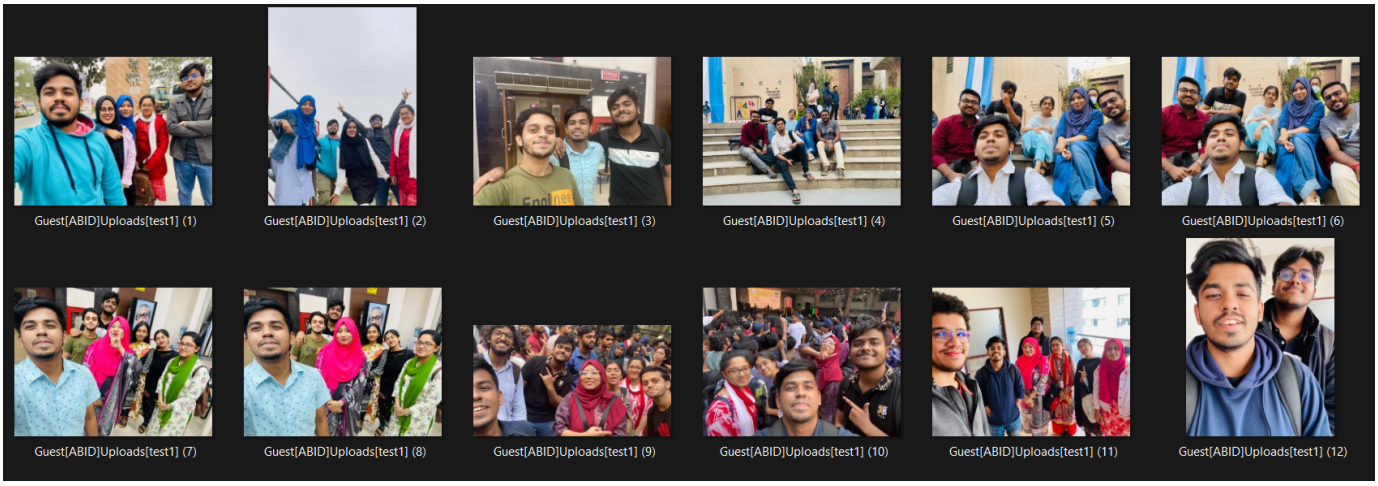

Fig. 8: Guest 1 Uploaded Images in Test 1 Dataset

setup restricts full automation capabilities, as continuous 24/7 operation on a local machine poses significant challenges [21].

3. Data Characteristics and Associated Challenges

a. Variability in Image Quality:

- **Lighting Conditions:** Images captured in different lighting environments (e.g., bright sunlight, dimly lit rooms) can affect the accuracy of facial detection and recognition.

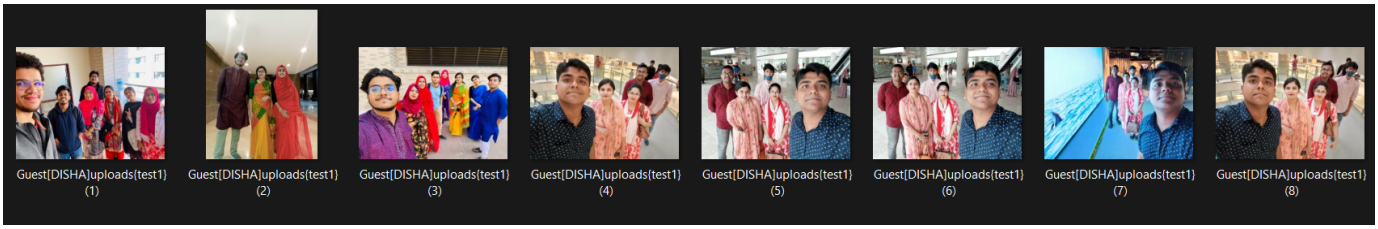

Fig. 9: Guest 2 Uploaded Images in Test 1 Dataset

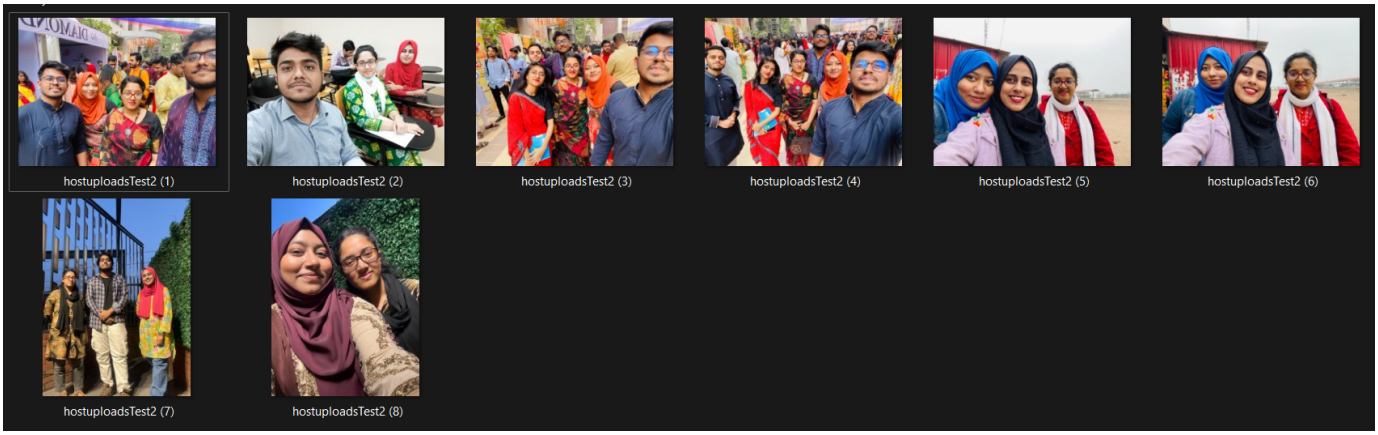

Fig. 10: Host Uploaded Images for Test 2 Dataset

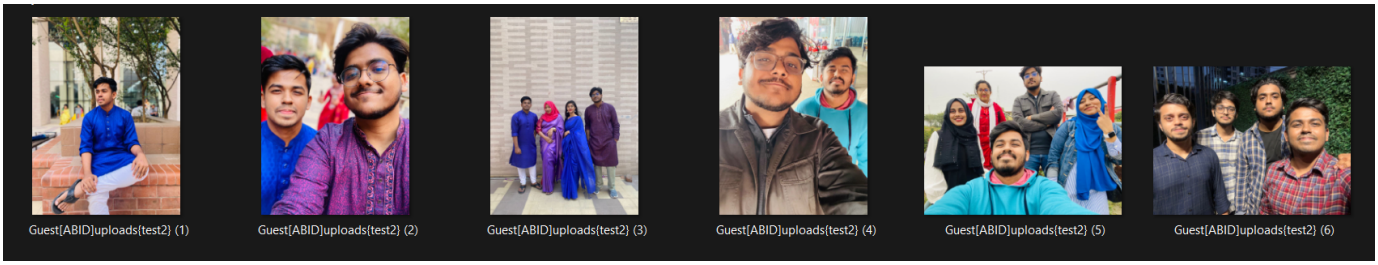

Fig. 11: Guest 1 Uploaded Images in Test 2 Dataset

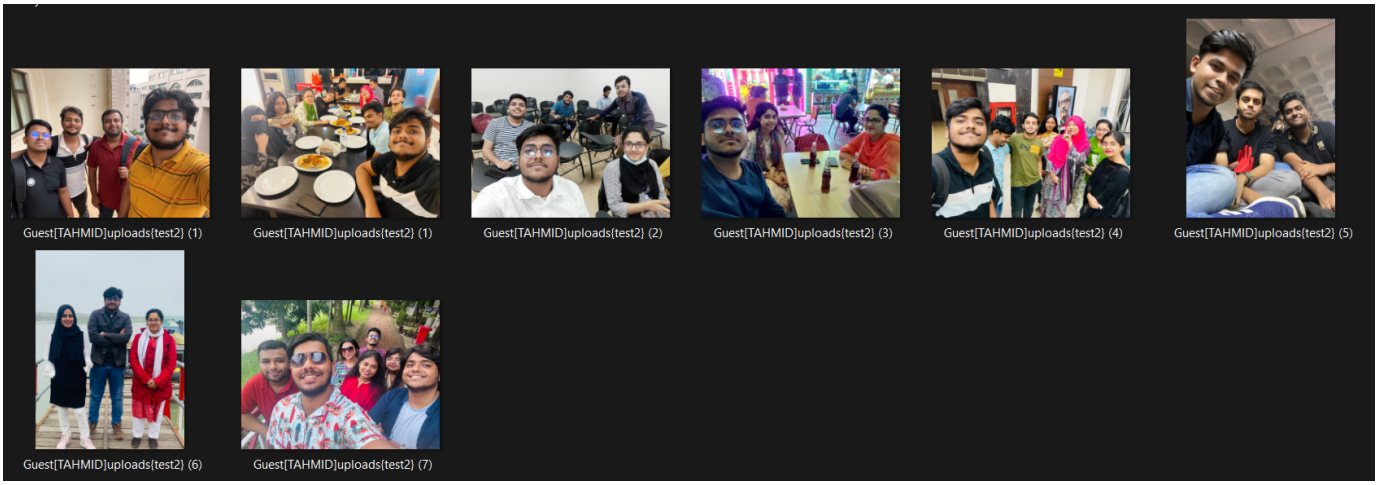

Fig. 12: Guest 2 Uploaded Images in Test 2 Dataset

- **Image Resolution:** Low-resolution images may lack the necessary detail for effective face recognition, while high-resolution images require more processing power and time.

b. Diversity in Participant Presentation:

- **Facial Expressions and Poses:** Participants appear in various facial expressions and poses, which can impact the consistency of recognition.
- **Occlusions:** Accessories such as glasses or masks partially covering faces introduce additional challenges for accurate facial feature extraction.

c. Dynamic and Unstructured Data:

- **Unstructured Uploads:** Images are uploaded at different times and by different users without a standardized format or naming convention, necessitating robust preprocessing to handle inconsistencies.
- **Real-Time Processing Needs:** The system must efficiently process incoming images in real-time or near-real-time to provide timely organization and access.

C. Methodological Approach to Solving the Problem

Our approach encompasses a comprehensive pipeline designed to automate the organization of event photos through facial recognition. The methodology is divided into five primary components:

a. Data Preprocessing
b. Data Storage and Management
c. Automated Photo Identification and Sorting
d. Handling Data Processing Challenges
e. User Accessibility and Feedback

Additionally, we conducted a backend model analysis to evaluate system performance and identify potential optimizations.

a. Data Preprocessing:

- **Face Detection:** Utilizing OpenCV's Haar Cascades, the system detects and localizes faces within each image. This step involves converting images to grayscale to enhance detection accuracy and applying the Haar Cascade classifier [20].

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
↪ 'haarcascade_frontalface_default.xml')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
↪ minNeighbors=5, minSize=(30, 30))
```

Listing 1: Face Detection using OpenCV

```
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
↪ 'haarcascade_frontalface_default.xml')
```

Listing 2: Initializing Classifier

This line initializes a Haar Cascade classifier for detecting faces using the `haarcascade_frontalface_default.xml` model. The XML file contains pre-trained data for face detection and is part of OpenCV's standard datasets.

```
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
```

Listing 3: BGR to Gray Scale Conversion

This converts the input image (image) from the BGR color space (used by OpenCV) to grayscale. Haar Cascades work better with grayscale images because they are computationally simpler and focus only on intensity differences, ignoring color information.

```
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1,
↪ minNeighbors=5, minSize=(30, 30))
```

Listing 4: Detects Faces in Gray Scale

This detects faces in the grayscale image using the classifier.
scaleFactor=1.1: Specifies how much the image size is reduced at each image scale. Smaller values detect smaller faces but increase computation.
minNeighbors=5: Specifies how many neighbors each candidate rectangle should have to retain it. Higher values reduce false positives.
minSize=(30, 30): Specifies the minimum size of detected faces. Smaller faces are ignored.
The result `faces` is a list of rectangles where each rectangle is defined by **(x, y, w, h)** — the coordinates of the top-left corner and the width and height of the rectangle enclosing a detected face.

- **Face Cropping and Embedding Generation:** Once faces are detected, they are cropped from the original image. The `face_recognition` library then generates

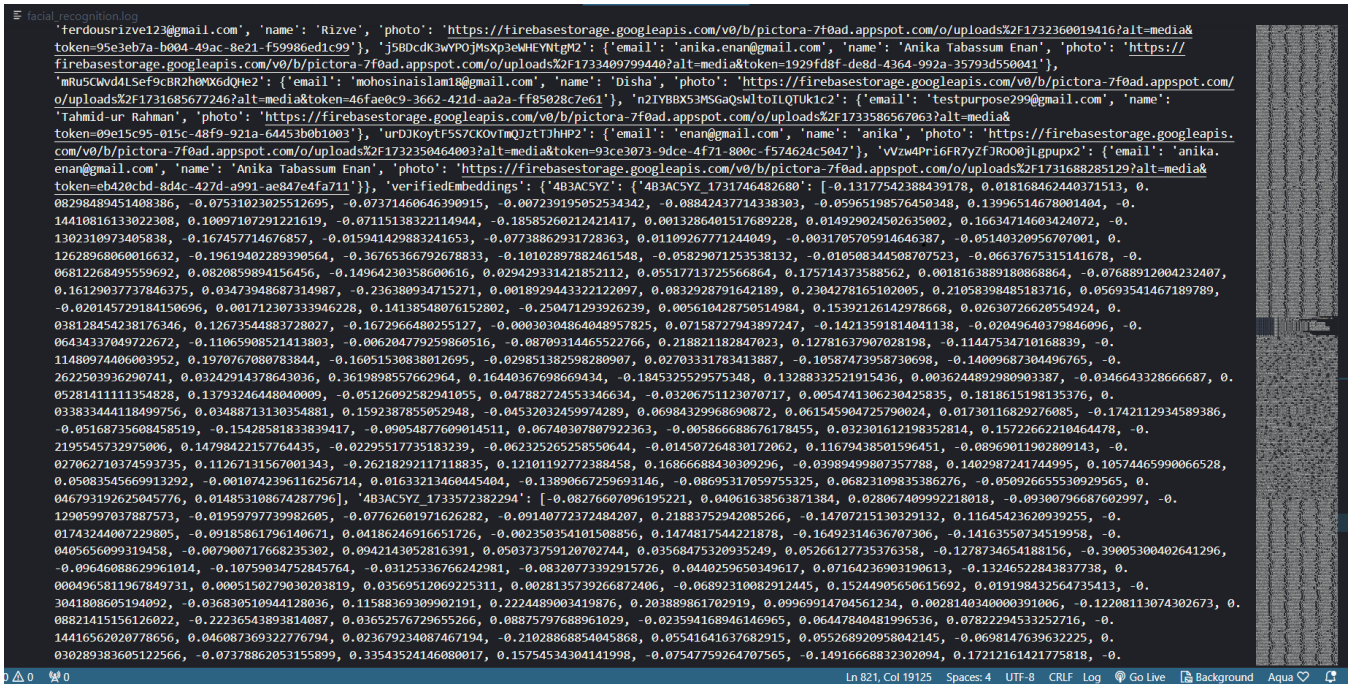

Fig. 13: Local Machine Log Message for 128-Dimensional Embeddings

128-dimensional embeddings13 and stores the embeddings in Firebase Realtime Database 14 15 [2] for each detected face, serving as unique identifiers for individuals [3]. The admins can see the output console implementations of the embedding generations where they can verify if an embedding for a profile is generated explicitly or not.16 17

```
rgb_image = cv2.cvtColor(face, cv2.COLOR_BGR2RGB)
encodings = face_recognition.face_encodings(rgb_image)
if encodings:
    embedding = encodings[0]
```

Listing 5: Face Cropping and Embedding Generation

- **Normalization:** Embeddings are normalized to ensure consistency and improve comparison accuracy [11].

b. Data Storage and Management:

- **Firebase Integration:** Firebase Realtime Database and Firebase Storage are employed to store participant profile images 18, event photos 19 20, and their corresponding facial embeddings 14 15. This ensures scalable and secure data management [2], [13].

```
cred =
↪ credentials.Certificate("path_to_firebase_credentials.json")
firebase_admin.initialize_app(cred, {
    'storageBucket': 'firebase-storage-bucket',
    'databaseURL': 'firebase-database-url'
})
```

Listing 6: Firebase Integration

- **Data Organization:** Profile images are stored in dedicated folders for each participant, while event photos are centralized and categorized based on embeddings [21].

c. Automated Photo Identification and Sorting:

- **Embedding Comparison:** The system compares embeddings from event photos with those of participant profiles using Euclidean distance. A predefined threshold (e.g., 0.45) determines whether a match is considered valid [11].

```
distance = np.linalg.norm(embedding - verified_embedding)
if distance < 0.45:
    matched_participants.add(participant_id)
```

Listing 7: Embedding Comparison

- **Categorization:** Matched photos are uploaded to participant-specific folders in Firebase Storage, while unmatched 22 photos are stored separately for review [2].

```
if matched_participants:
    for participant_id in matched_participants:
        guest_folder =
↪ f"rooms/{event_code}/{participant_id}/photos/"
        matched_blob =
↪ bucket.blob(f"{guest_folder}{os.path.basename(blob.name)}")
```

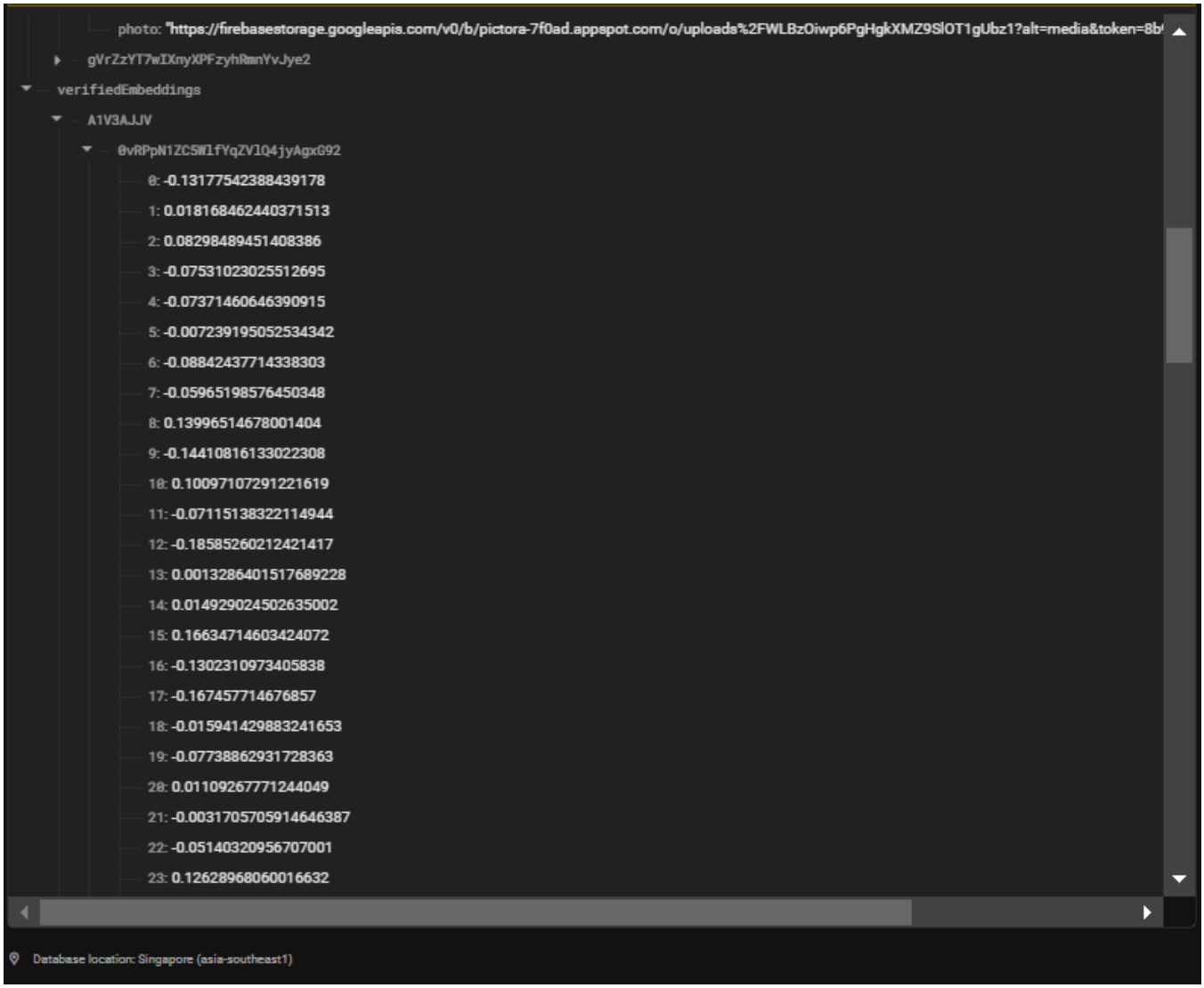

Fig. 14: Firebase Realtime Database Stores the 128 Embedded Values: Part 1

```
↪ matched_blob.upload_from_filename(str(local_image_path))
else:
    unmatched_path =
↪ f"{unmatched_folder}unmatched_{unmatched_counter}.jpg"
    unmatched_blob = bucket.blob(unmatched_path)
    unmatched_blob.upload_from_filename(str(local_image_path))
```

Listing 8: Photo Categorization

```
            return func(file_path)
        except OSError as e:
            if hasattr(e, 'errno') and e.errno == 32:
                time.sleep(delay)
            else:
                break
    return []
```

Listing 9: Retry Mechanism

d. Handling Data Processing Challenges:

- **Retry Mechanisms:** To address issues such as file locks or transient errors during image processing, the system implements retry logic with delays, ensuring robustness and reliability [21].

```
def process_image_with_retry(file_path, func, retries=5,
↪ delay=1):
    for attempt in range(retries):
        try:
```

- **Error Handling and Logging:** Comprehensive error handling ensures that any issues encountered during processing are logged, facilitating debugging and system maintenance.

e. User Accessibility and Feedback:

- **Progress Indicators:** The system provides real-time progress updates to admins through console-based progress bars, informing users

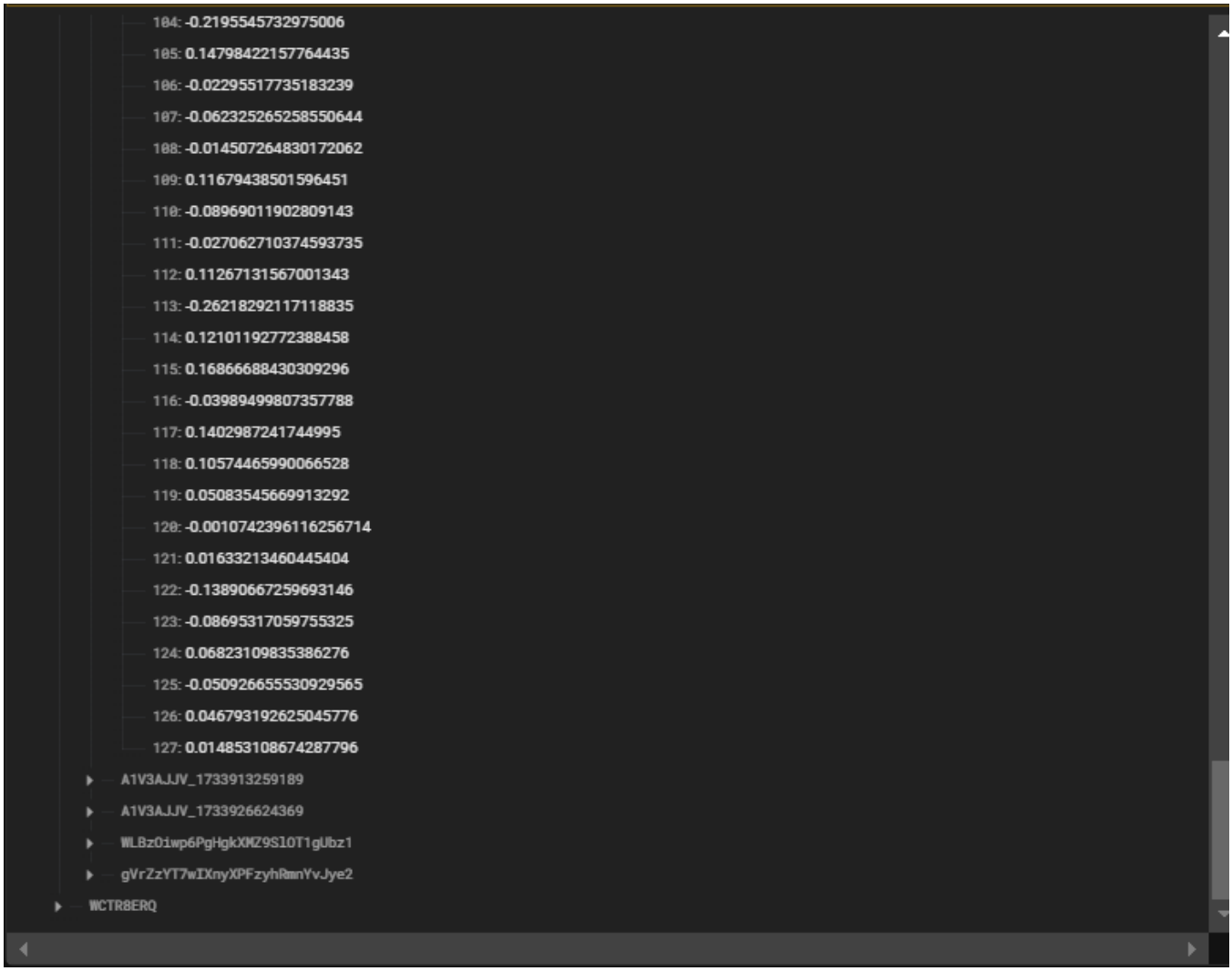

Fig. 15: Firebase Realtime Database Stores the 128 Embedded Values: Part 2

about the processing status and accuracy metrics **??**.

```
print_progress_bar(idx, total_photos, prefix='Facial
↪ Recognition Progress', length=50, accuracy=accuracy)
```

Listing 10: Progress Indicators

- **Final Summary and Metrics:** Upon completion, the system generates a summary of processed photos, including matched and unmatched counts, overall accuracy, and a confusion matrix for performance evaluation 23 24 [21].

```
print(f"Total Photos Processed: {total_photos}")
print(f"Matched Photos: {matched_photos}")
print(f"Unmatched Photos: {unmatched_photos}")
print(f"Overall Accuracy: {overall_accuracy:.2f}%")
```

Listing 11: Final Summary and Metrics

5. Problem Resolution through the Proposed System

By leveraging established libraries such as OpenCV for face detection [20] and `face_recognition` for embedding generation [3], the system efficiently addresses the challenges posed by the dataset's variability and scale. The integration with Firebase [2], [13] ensures scalable storage and real-time database management, facilitating seamless photo sorting and retrieval. Automated embedding comparison and categorization eliminate the need for manual intervention, significantly reducing the time and effort required to organize event photos. Additionally, robust error handling and progress tracking enhance the system's reliability and user

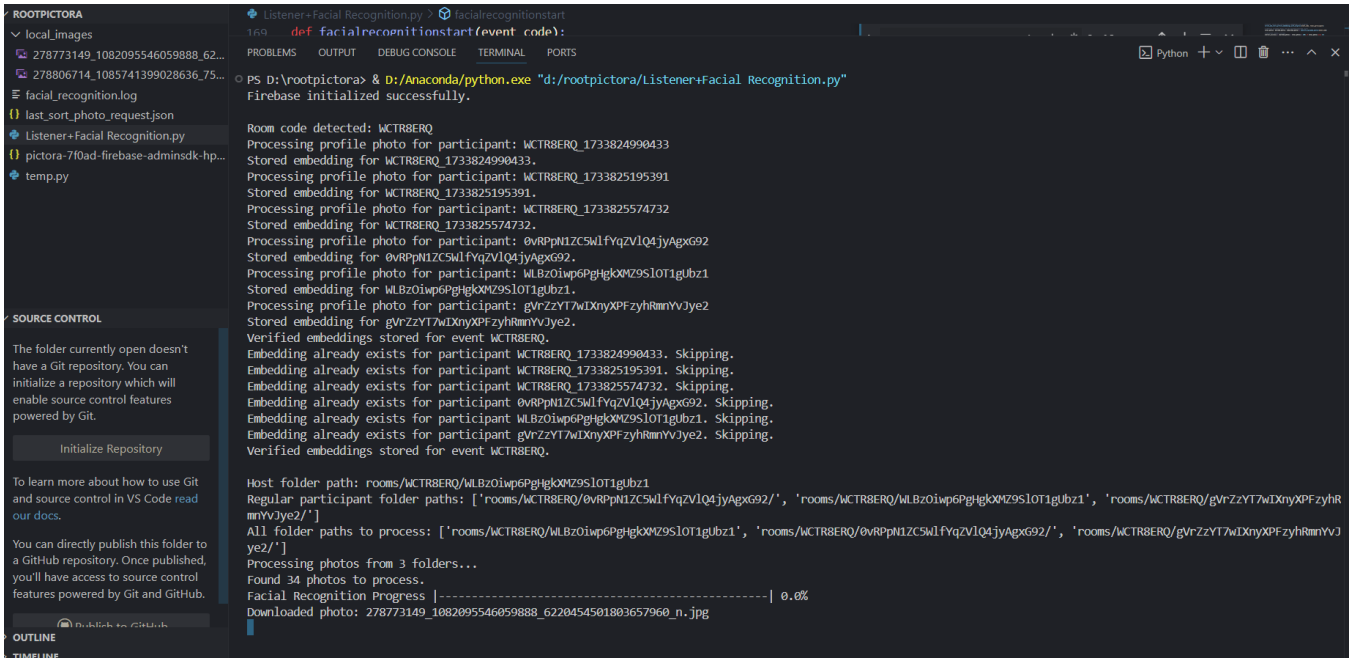

Fig. 16: Generated Embedding Console Message for Test 1

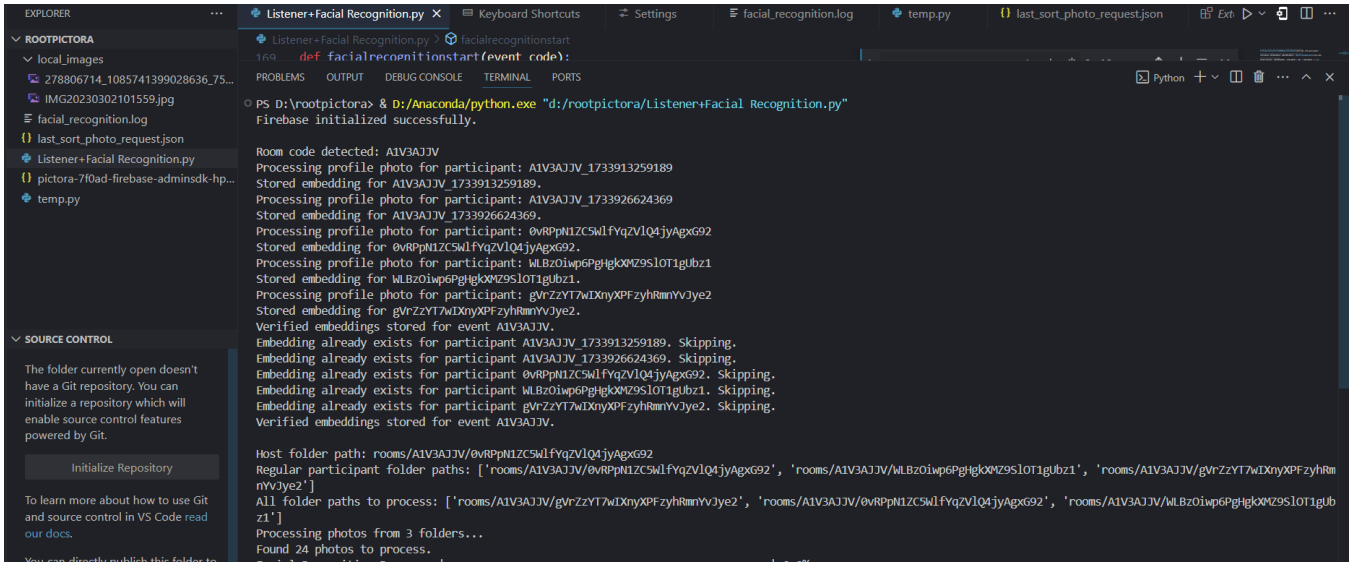

Fig. 17: Generated Embedding Console Message for Test 2

experience, ensuring that participants can access their curated photo collections promptly and effortlessly.

Backend Model Analysis for Limitations:

An integral part of our methodology is the analysis of the backend system's limitations, which is responsible for processing, storing, and retrieving data. Given the constraints of operating on a local machine without dedicated GPUs, several challenges arise:

1. **Hardware Limitations:**

- **No Dedicated GPU:** The absence of a GPU restricts the use of certain libraries and limits the processing speed, particularly for computationally intensive tasks like embedding generation [21].

- **Local Machine Hosting:** Hosting the backend on a local machine poses scalability issues and requires the system to be operational 24/7 to ensure

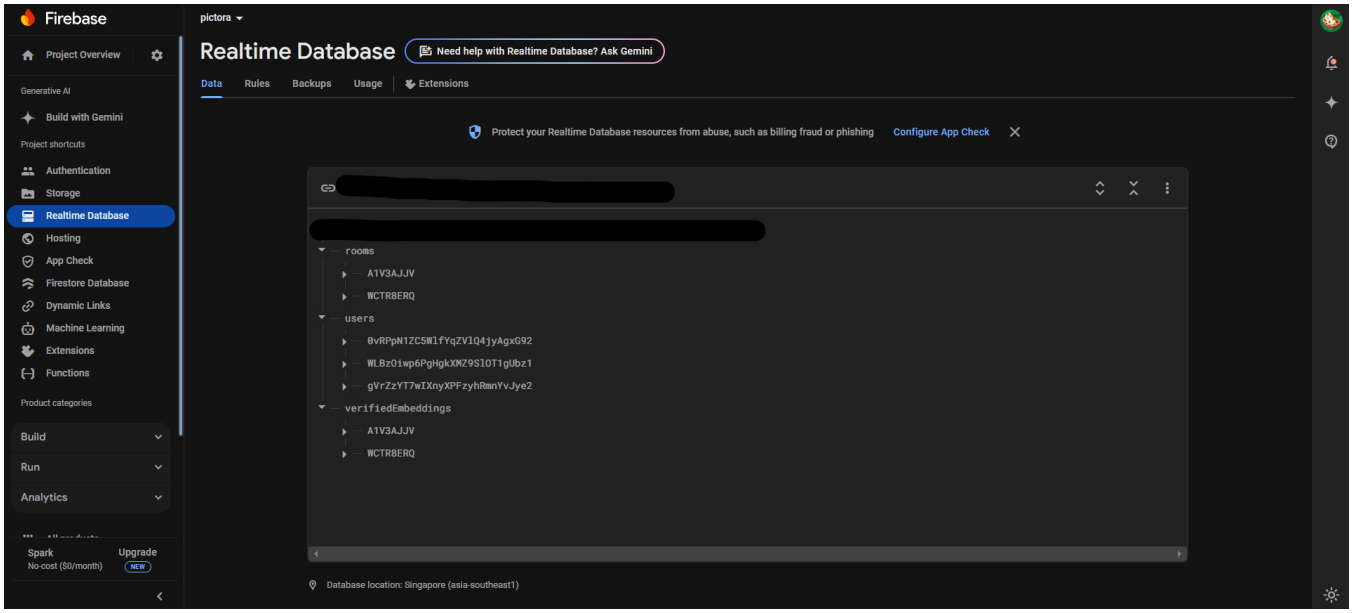

Fig. 18: Firebase Realtime Database Structure

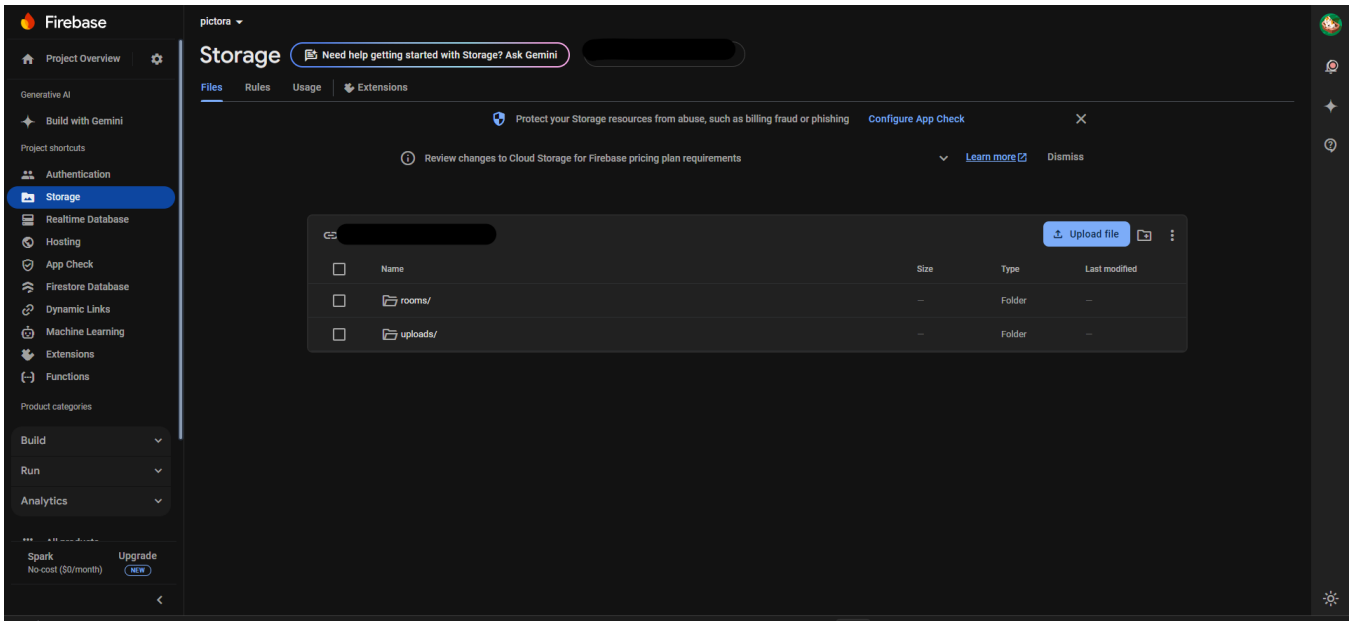

Fig. 19: Firebase Storage Structure: Root

full automation, which is impractical [21].

2. **Software Constraints:**

- **HEIC File Handling:** The inability to process HEIC files due to hardware limitations prevents the use of the `pyheif` library, limiting the range of supported image formats [**?**].

3. **Performance Challenges:**

- **Unmatched Outputs:** The system struggles with unmatched outputs in images 25 captured under extreme low-light conditions and unconventional angles, highlighting the need for improved image preprocessing and more robust recognition models [16].

4. **Scalability and Automation:**

- **Continuous Operation:** The requirement for the backend to run continuously on a local machine to achieve full automation is a significant hurdle, as it increases the risk of system downtime and

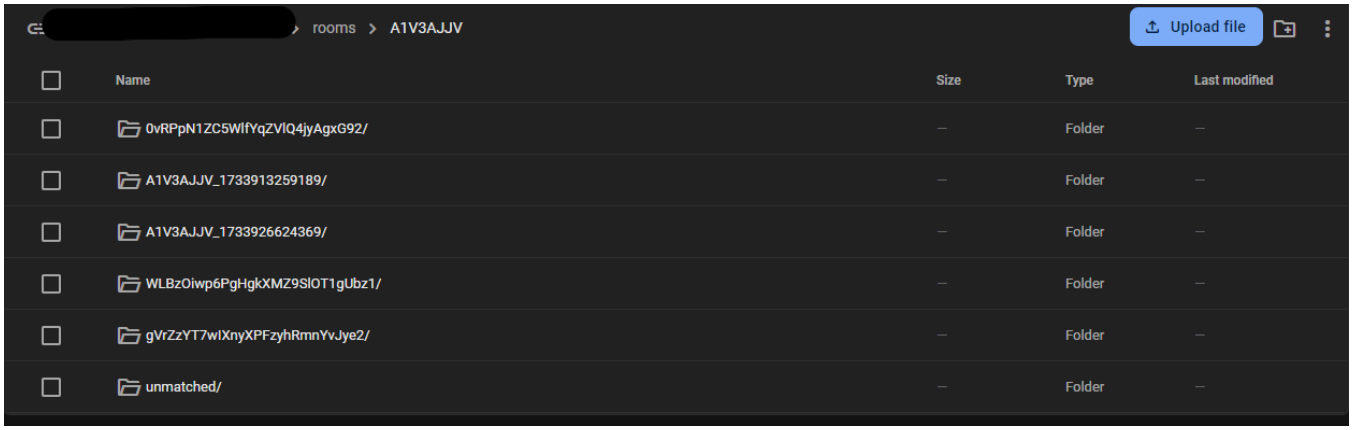

Fig. 20: Firebase Storage Structure: Depth 1

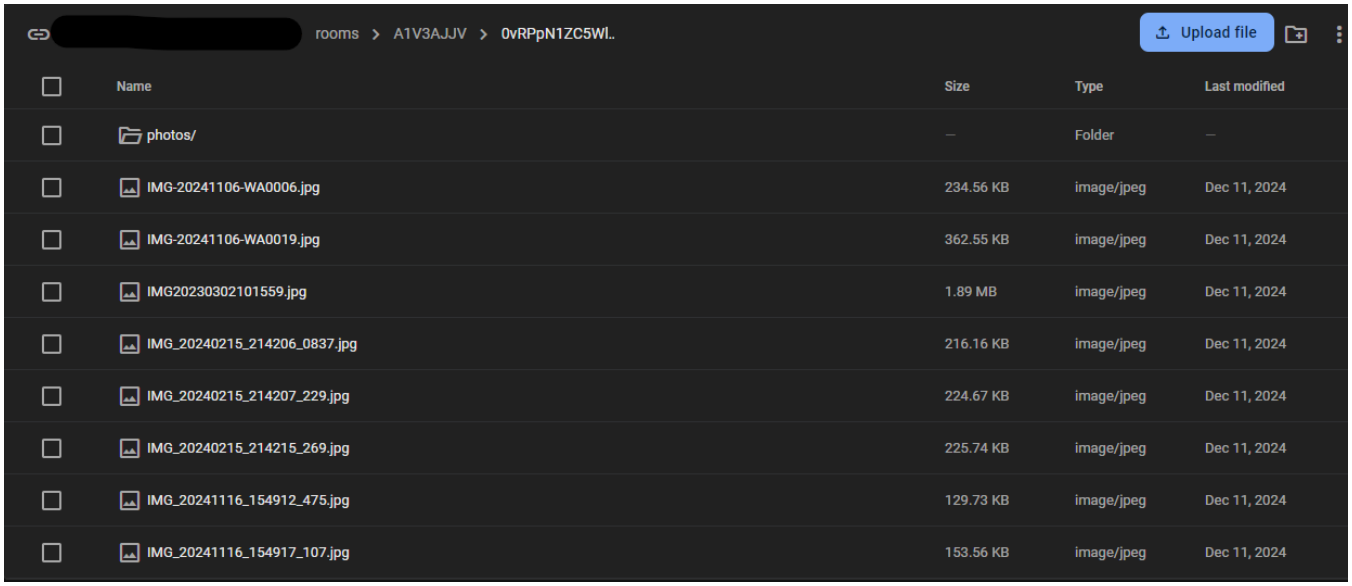

Fig. 21: Firebase Storage Structure: Depth 2

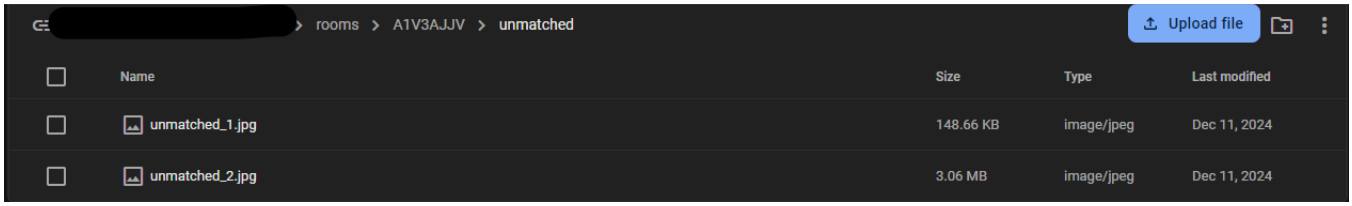

Fig. 22: Unmatched Photos Get Stored in Storage Under Room Code Path Appended with `unmatched` Path

complicates maintenance [21].

5. **Potential Solutions:**

- **Cloud-Based Hosting:** Migrating the backend to cloud servers with dedicated GPUs could alleviate hardware constraints, enable the use of advanced libraries, and support scalable, 24/7 operation [22].

- **Enhanced Preprocessing:** Implementing advanced image enhancement techniques such as adaptive histogram equalization, noise reduction filters, and alignment algorithms can improve face detection and embedding accuracy in challenging conditions [18].

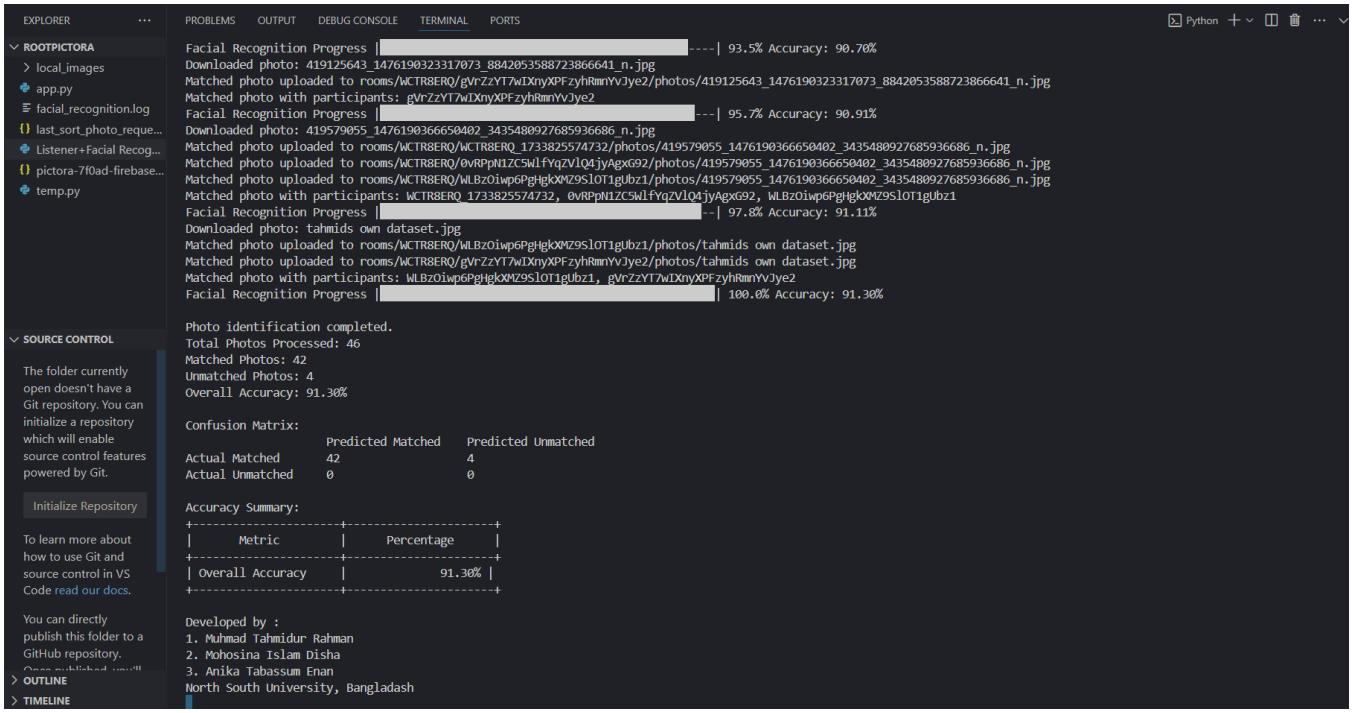

Fig. 23: Test Room 1 Console Printouts

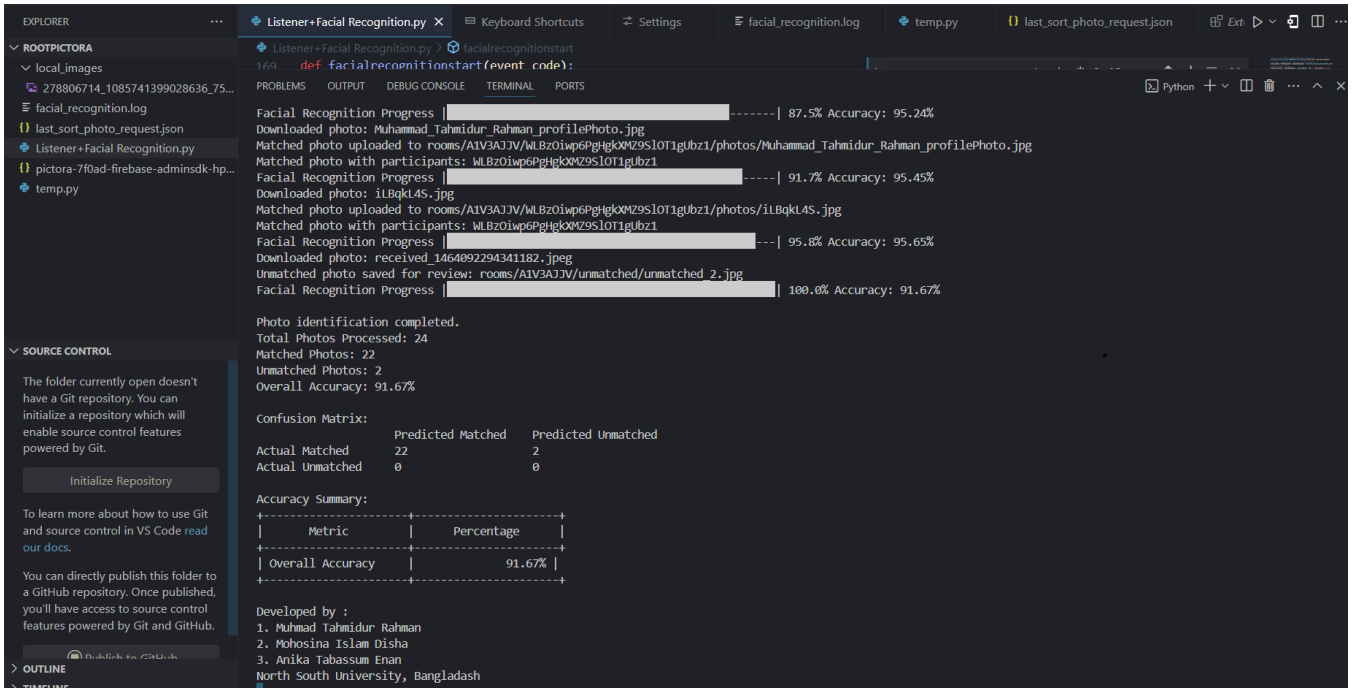

Fig. 24: Test Room 2 Console Printouts

- **Model Optimization:** Employing lightweight and efficient facial recognition models optimized for CPU execution can enhance performance without necessitating GPU support [23].

Final Summary The methodological approach effectively addresses the challenges of organizing event photos by leveraging facial recognition technologies and integrating scalable storage solutions.

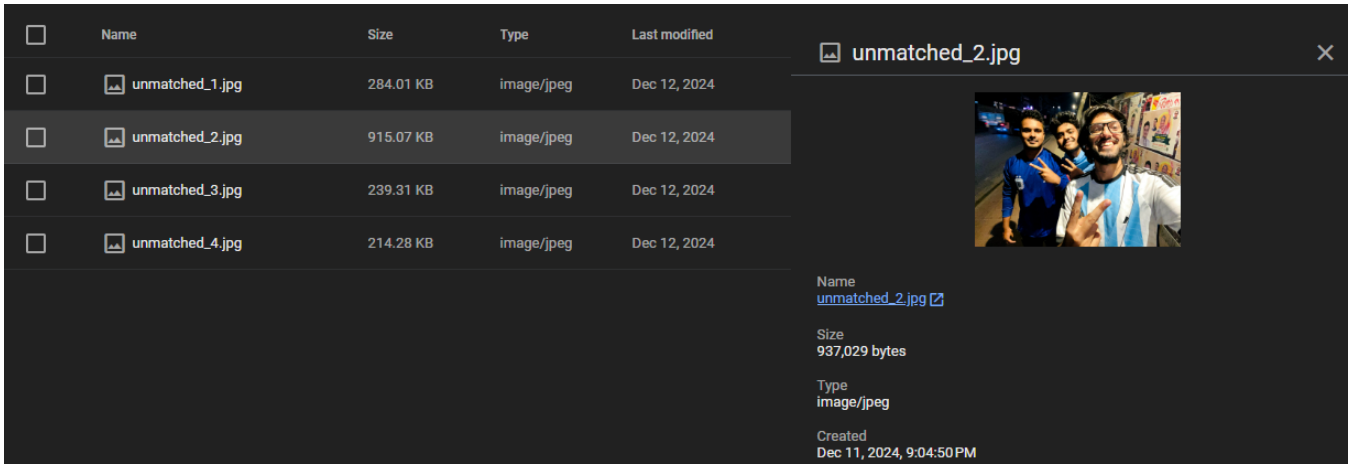

Fig. 25: Unmatched Images for Low-Light Conditions and Angle Variations

Despite hardware and software constraints, the system demonstrates robust performance in standard conditions, with identified areas for improvement in handling diverse and challenging image scenarios.

D. User Interface and Its Feasibility

The user interface (UI) of pictora web platform [5] and mobile platform [6] plays a crucial role in ensuring an intuitive and seamless user experience for both event hosts and participants. The UI is designed to be user-friendly, responsive, and accessible across various devices, including desktops, tablets, and smartphones. This subsection elaborates on the design principles, key features, and feasibility aspects of the UI for both the web platform and the mobile application.

1) Design Principles: The UI design adheres to the following principles to enhance usability and accessibility:

- **Simplicity:** The interface is kept clean and uncluttered, presenting only essential features to avoid overwhelming users.
- **Consistency:** Uniform design elements and navigation patterns are maintained across both platforms to provide a cohesive user experience.
- **Responsiveness:** The UI is optimized for various screen sizes and resolutions, ensuring functionality on desktops, tablets, and mobile devices.
- **Accessibility:** Features such as keyboard navigation, screen reader support, and appropriate contrast ratios are incorporated to accommodate users with disabilities.
- **Feedback:** Interactive elements provide immediate feedback (e.g., button presses, form submissions) to inform users of their actions.

2) Web Platform Interface: The web platform [5] serves as the primary interface for event hosts to manage rooms, upload photos, and oversee the sorting process. Key features were discussed earlier on web flow diagram sectionIV-A.Key UI components include:

3) Mobile Application Interface: The mobile application named Pictora [6]caters to both hosts and guests, offering a soothing experience for touch interactions and on-the-go access. key features were discussed earlier on the flow diagram sections IV-A.Key UI components include:

4) Feasibility Considerations: The feasibility of the UI design is ensured through careful selection of technologies and adherence to best practices in UI/UX design:

- **Tools used:** The web platform is developed using javascripts enabling dynamic and responsive interfaces. The mobile application leverages cross-platform frameworks like Flutter using Android studio framework ensuring consistent performance across devices.
- **Performance Optimization:** Lazy loading techniques and efficient state management are implemented to ensure quick load times and smooth interactions, even with large photo libraries.

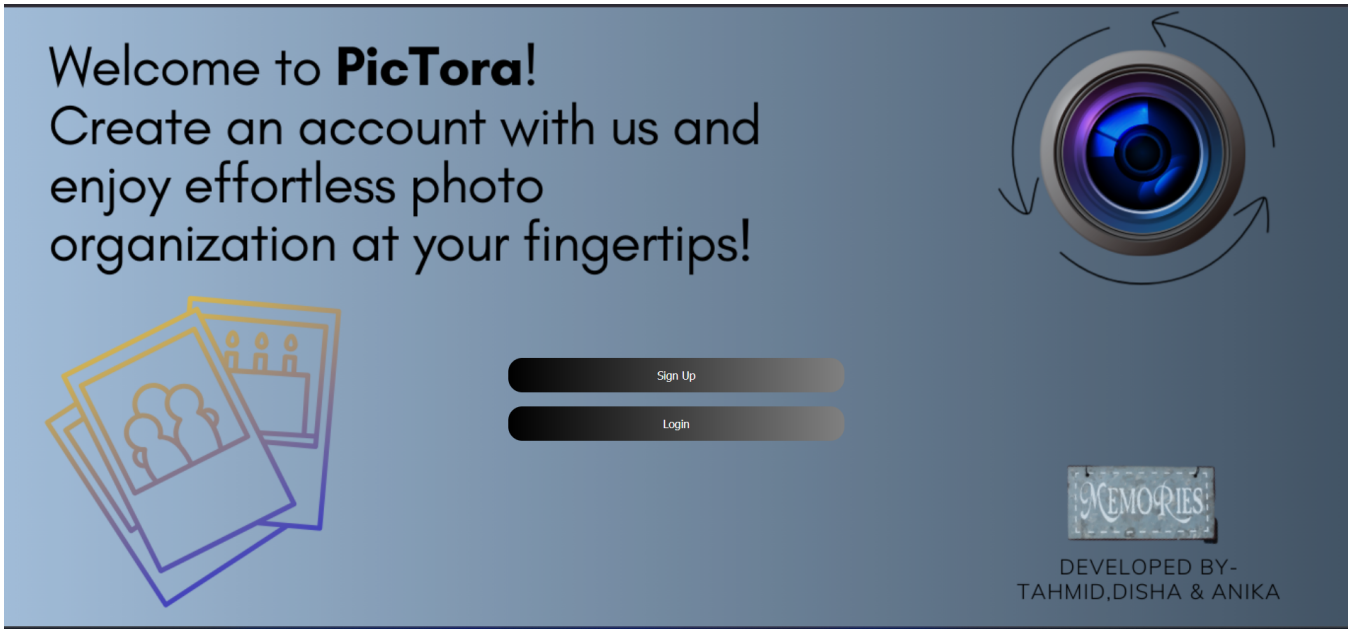

Fig. 26: Web Platform Welcome Page

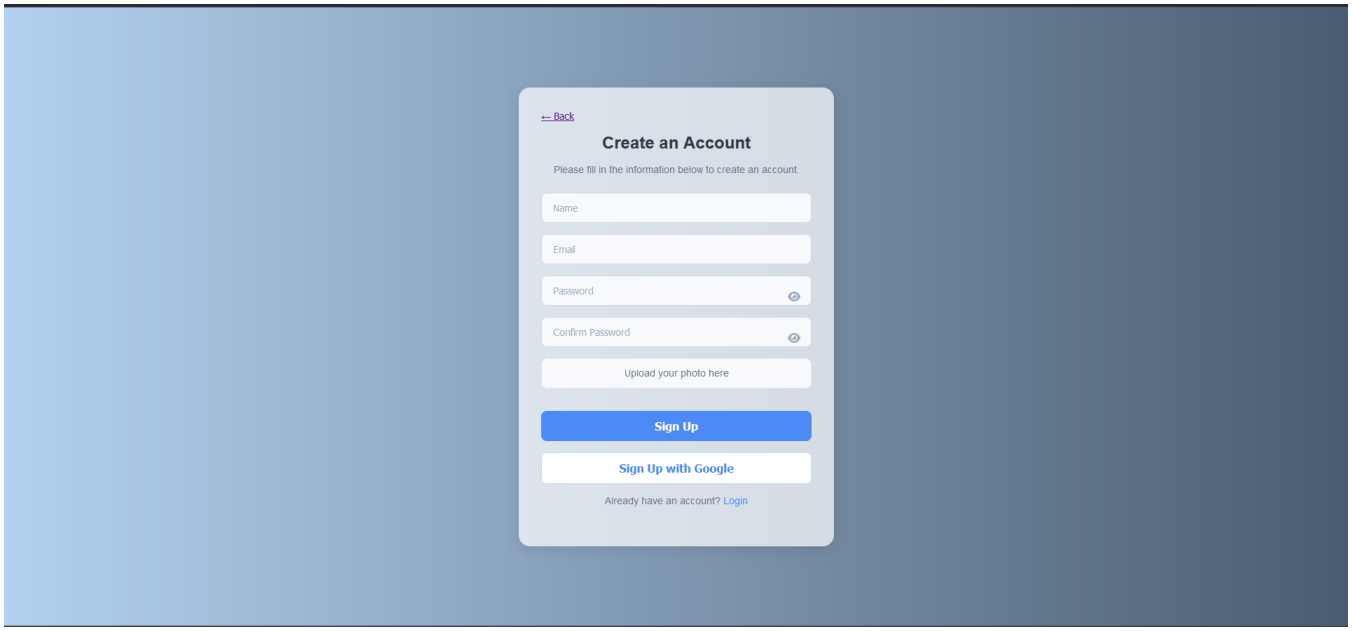

Fig. 27: Web Platform SignUp Page

- **Scalability:** The UI components are designed to handle increasing numbers of users and photos without compromising performance, supported by scalable backend services.
- **User Testing and Iteration:** Prototypes of the UI were subjected to user testing sessions with a diverse group of participants. Feedback from these sessions informed iterative improvements, enhancing usability and satisfaction.
- **Security Measures:** Secure authentication protocols and data encryption are integrated into the UI to protect user data and privacy, aligning with industry standards and regulations.

Overall, the UI design of the proposed system balances aesthetic appeal with functional efficiency, ensuring that users can effortlessly manage and

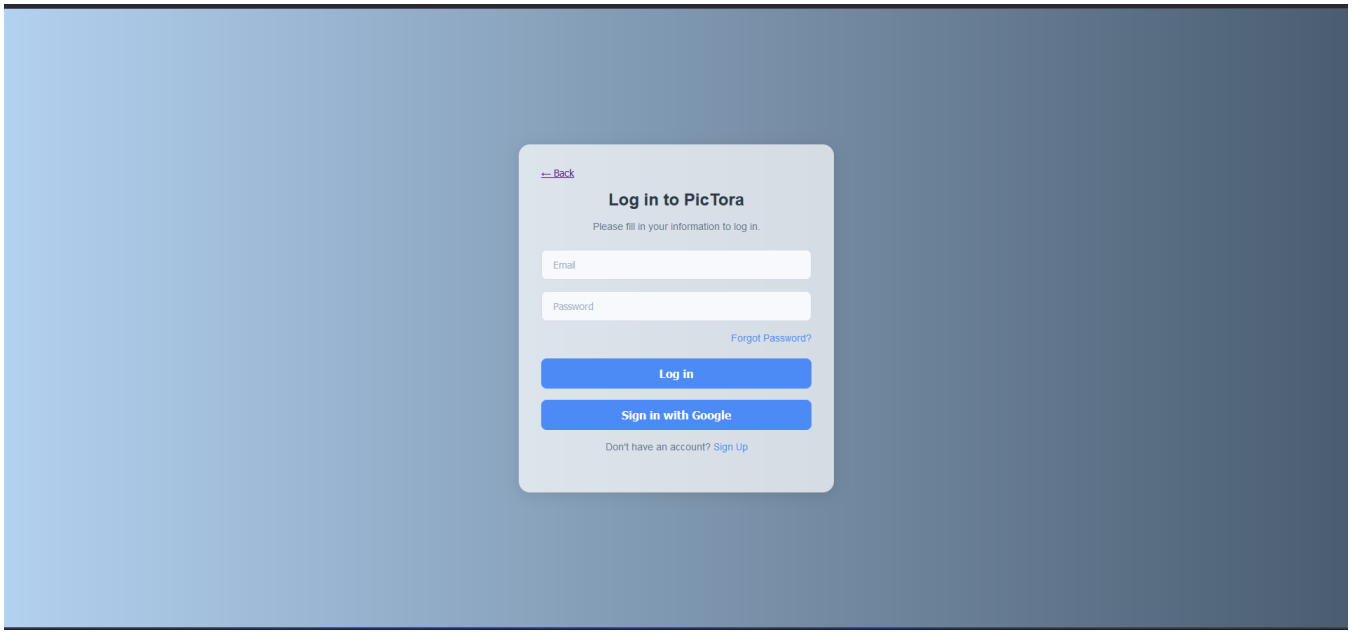

Fig. 28: Web Platform Login Page

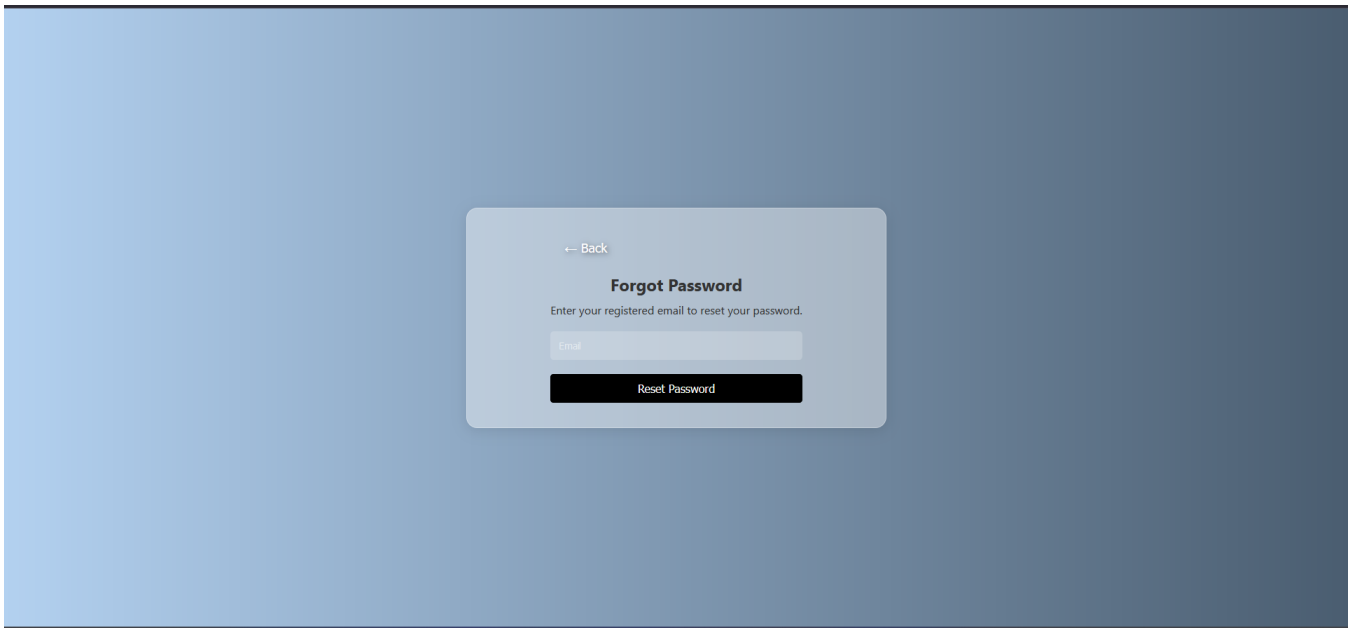

Fig. 29: Web Platform ForgotPassword Page

access their event photos. The feasibility of the UI is underpinned by a robust technology stack, user-centered design practices, and scalable architectural choices, making the system both practical and user-friendly for diverse event scenarios.

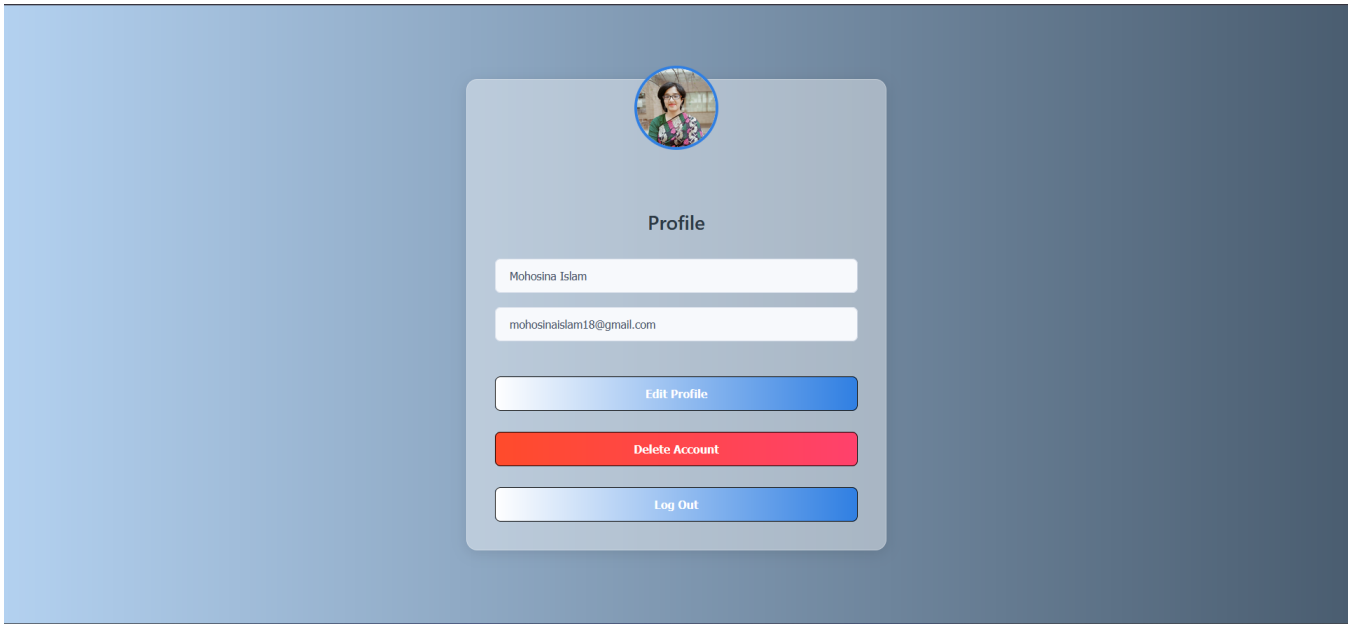

Fig. 30: Web Platform Profile Page

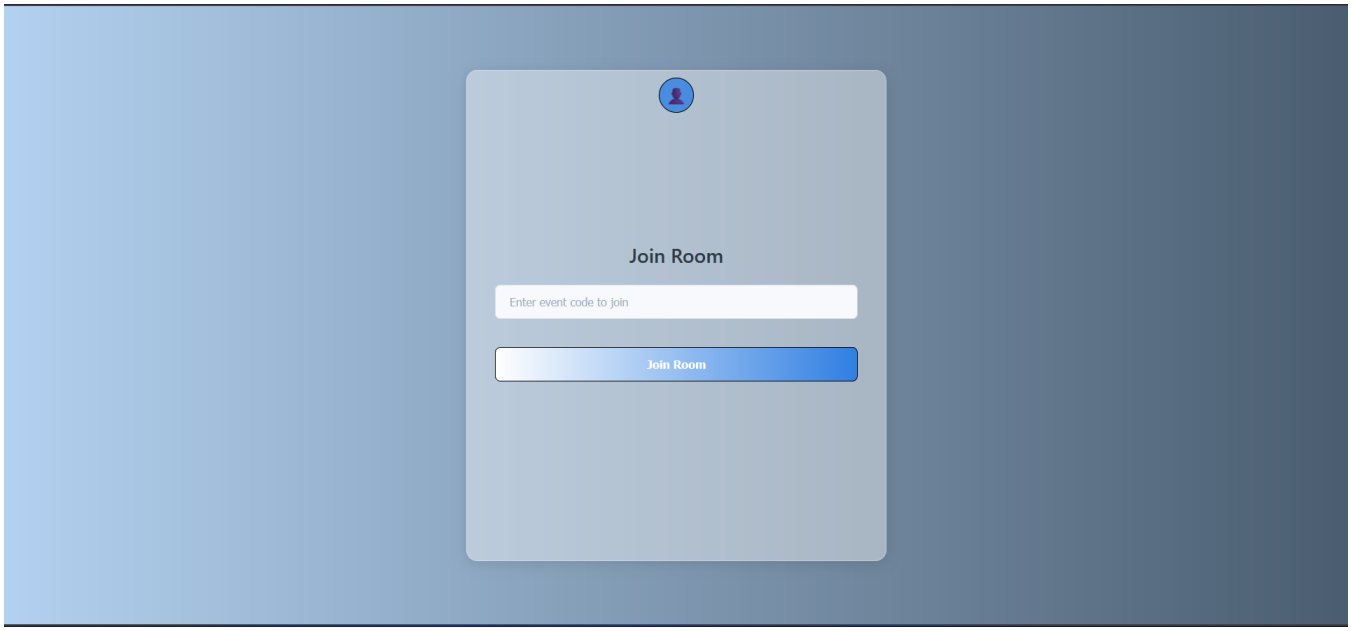

Fig. 31: Web Platform JoinEvent Page

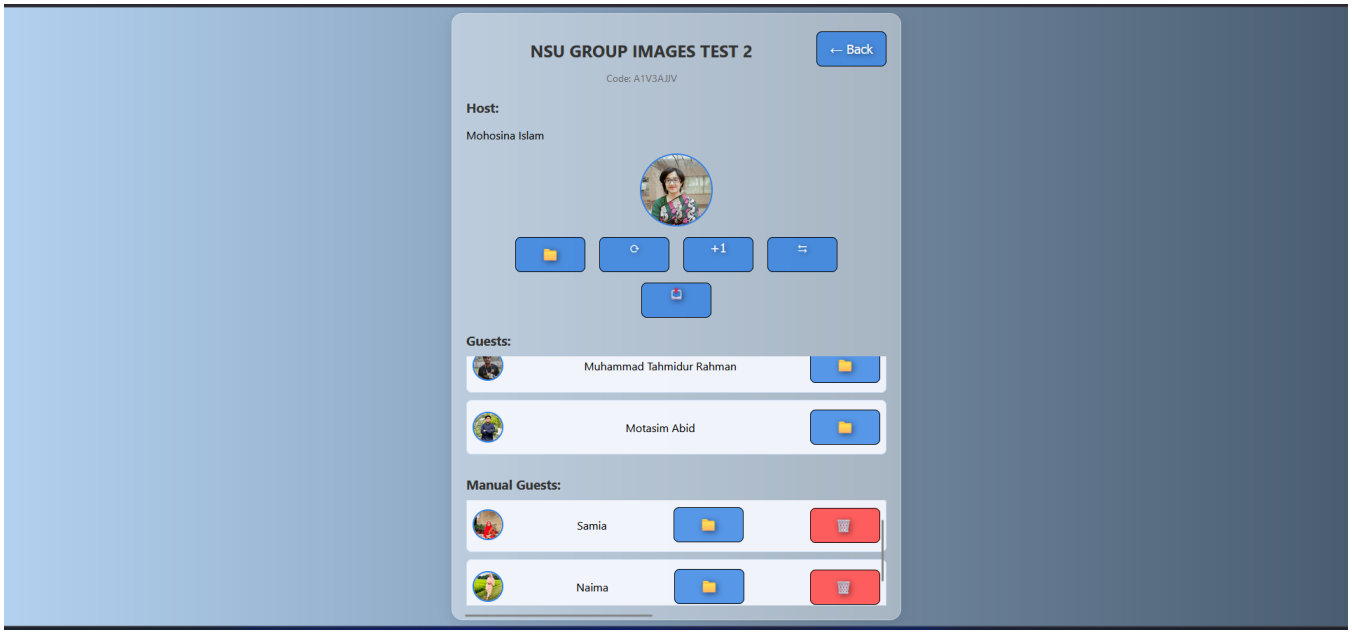

Fig. 32: Web Platform Event Room Host POV Page

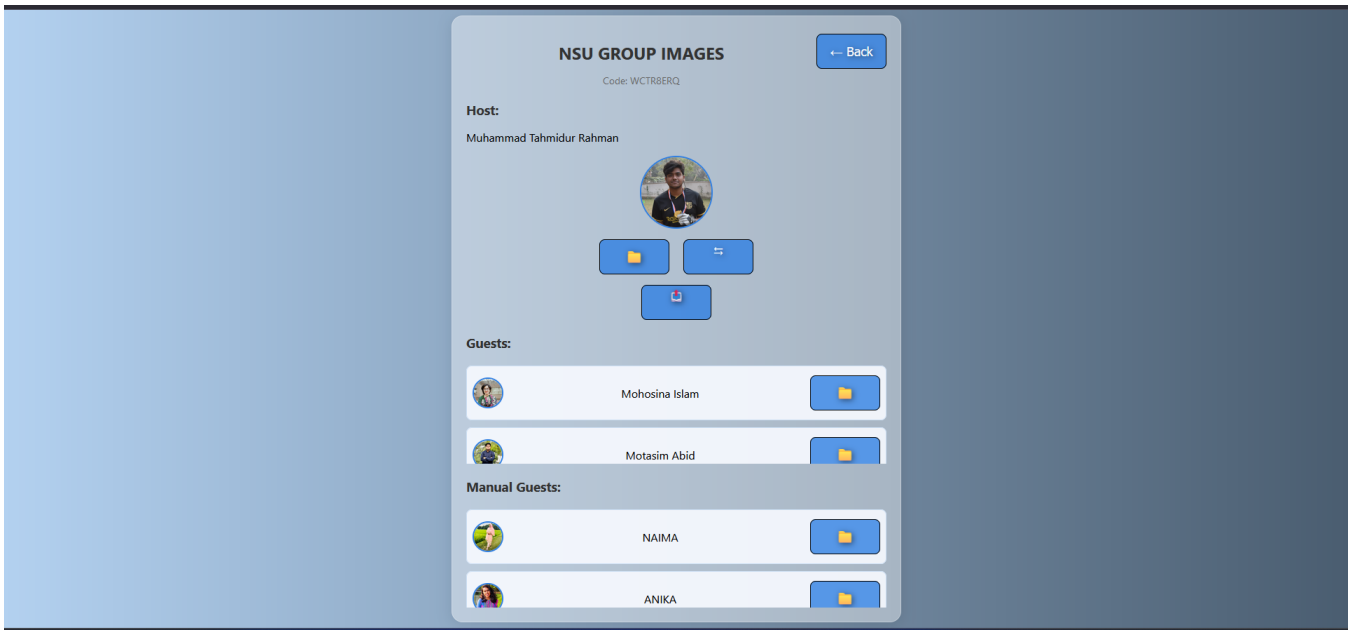

Fig. 33: Web Platform Event Room Guest POV Page

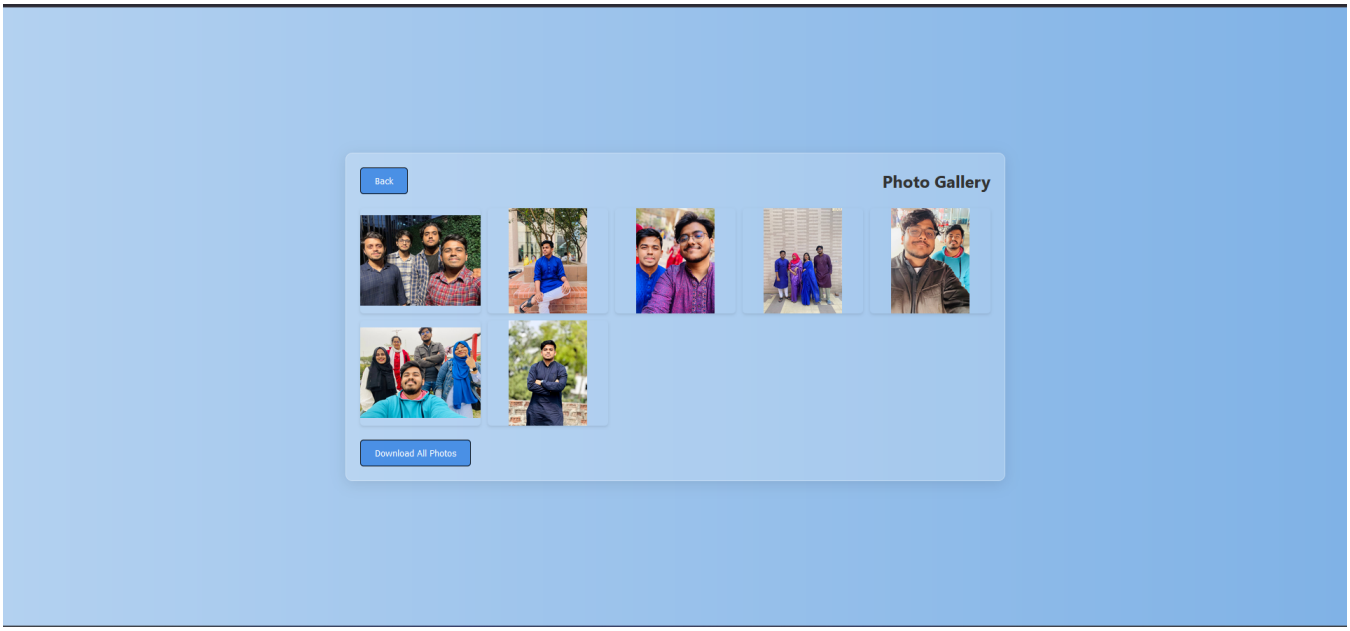

Fig. 34: Web Platform Uploaded Photo Gallery Page

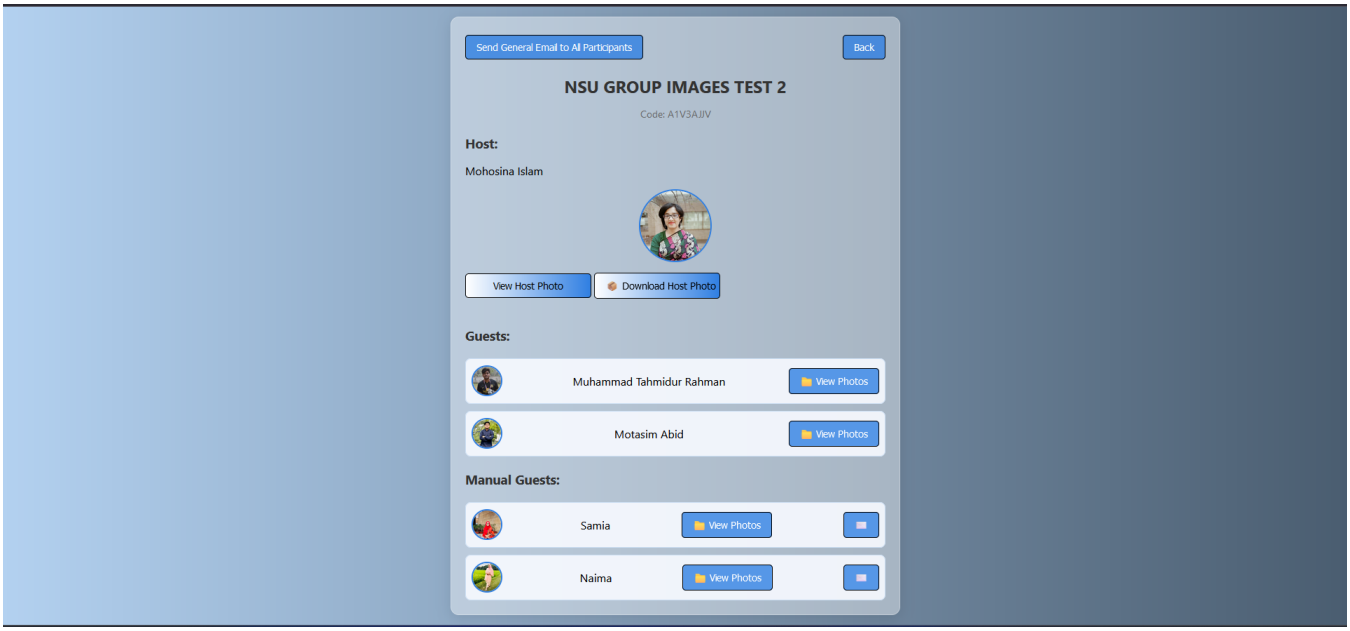

Fig. 35: Web Platform Arranged Photo Page

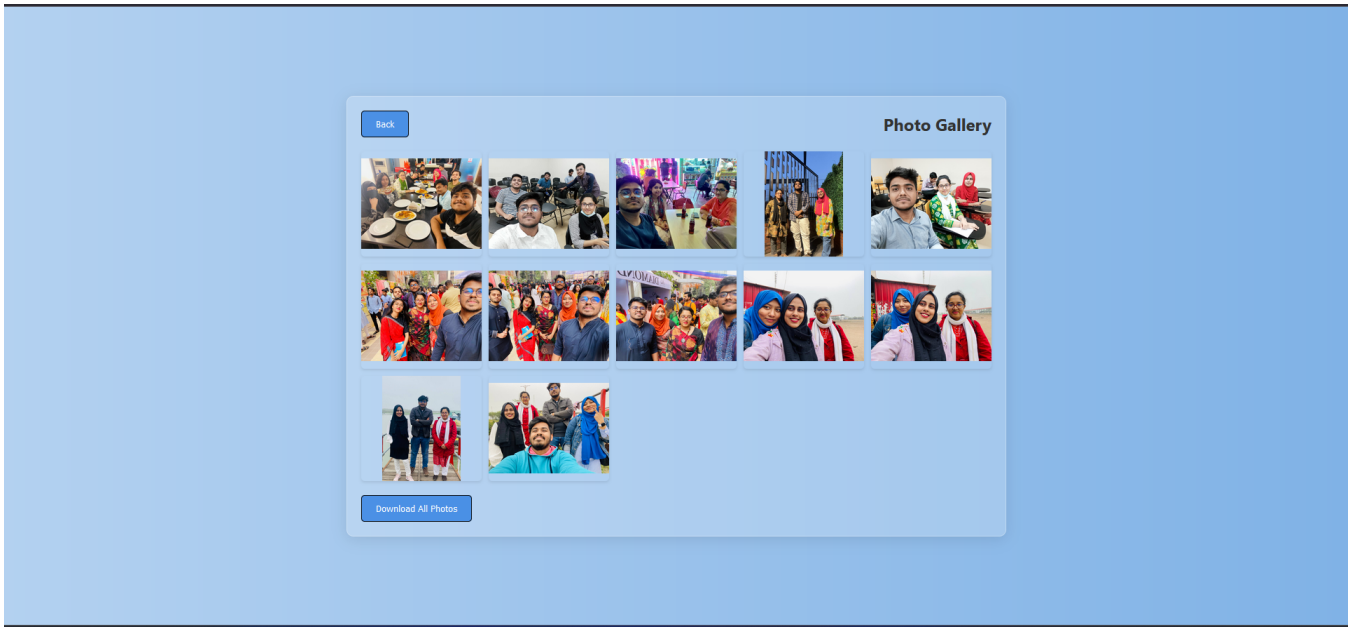

Fig. 36: Web Platform Sorted Photo Gallery Page

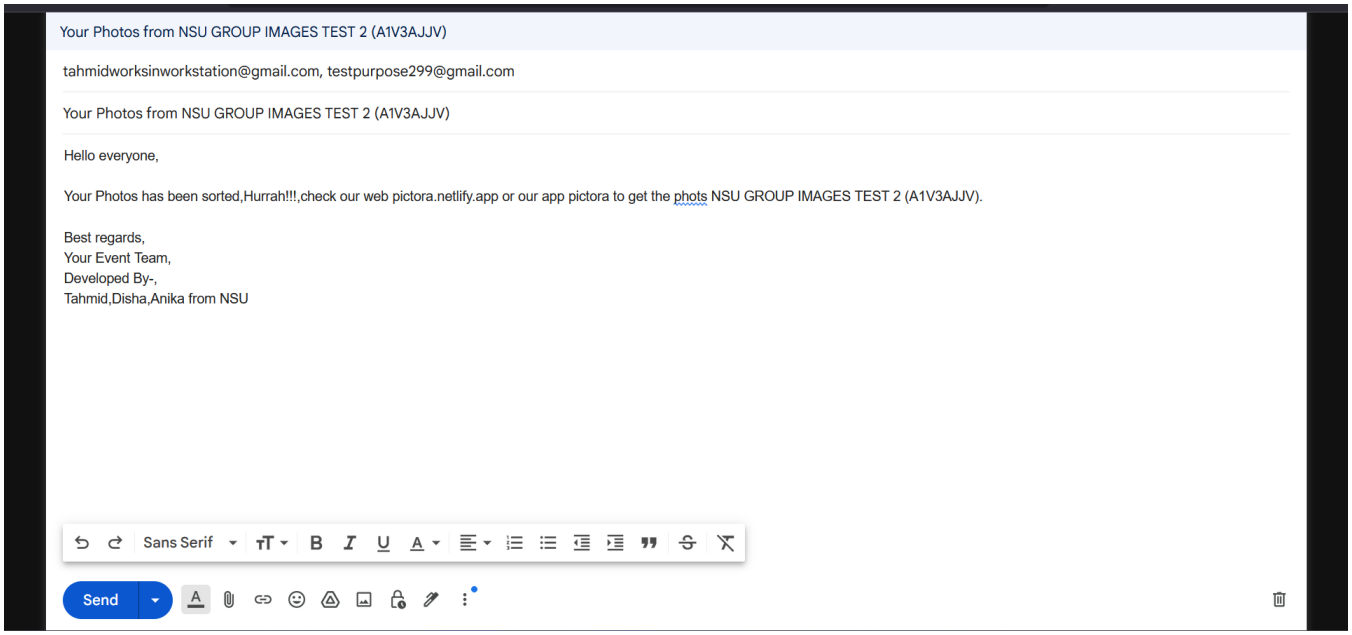

Fig. 37: Web Platform Generalised Email generate Page

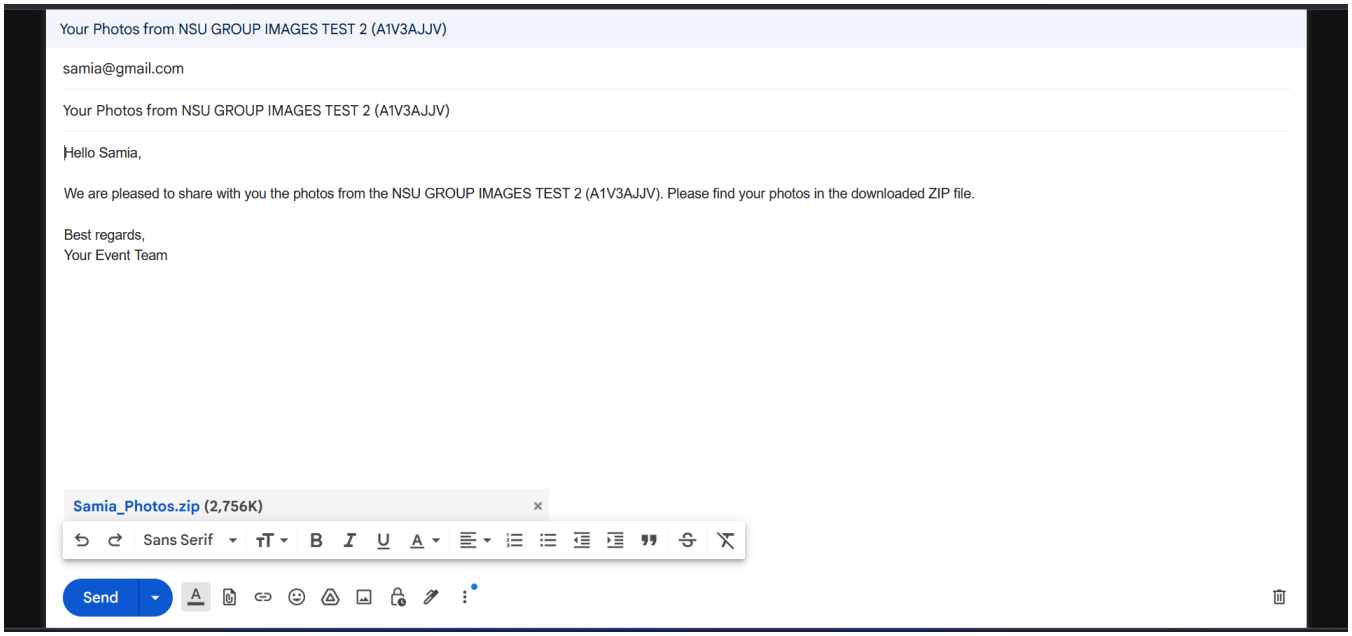

Fig. 38: Web Platform Added Zip to email Page

Fig. 39: App Platform Welcome Page

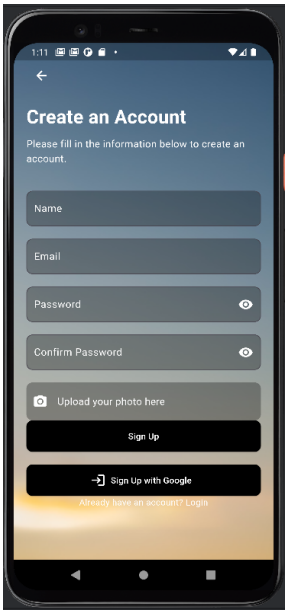

Fig. 40: App Platform SignUp Page

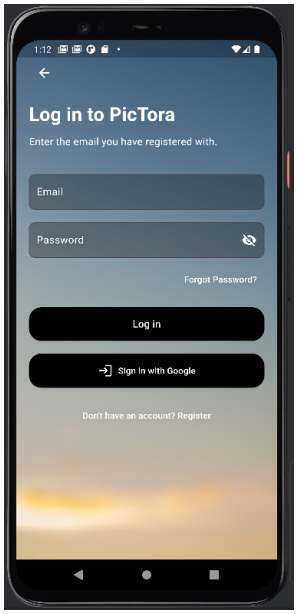

Fig. 41: App Platform Login Page

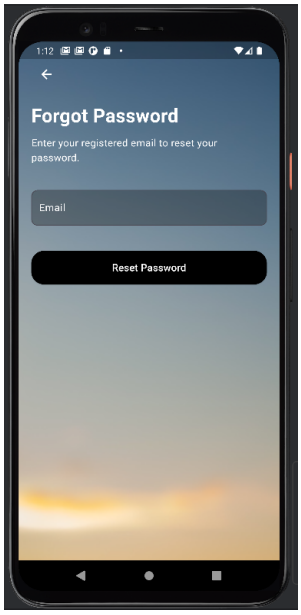

Fig. 42: App Platform ForgotPassword Page

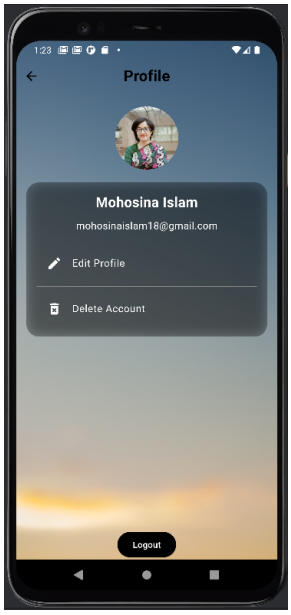

Fig. 43: App Platform Profile Page

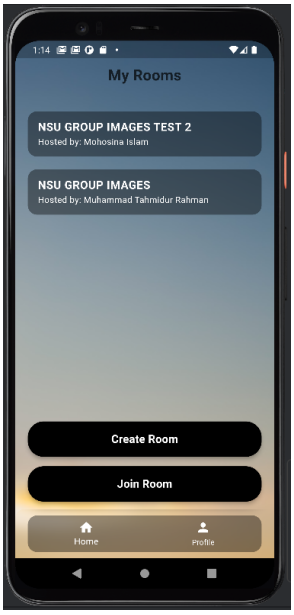

Fig. 44: App Platform Create Or Join Room Page

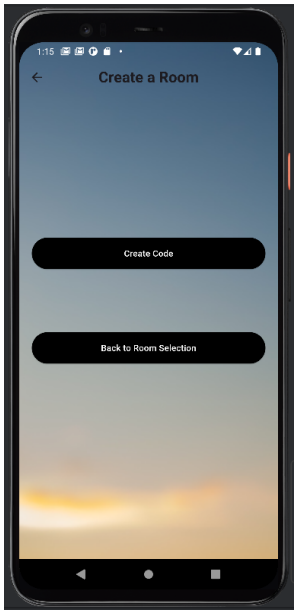

Fig. 45: App Platform Create Room Page:Part1

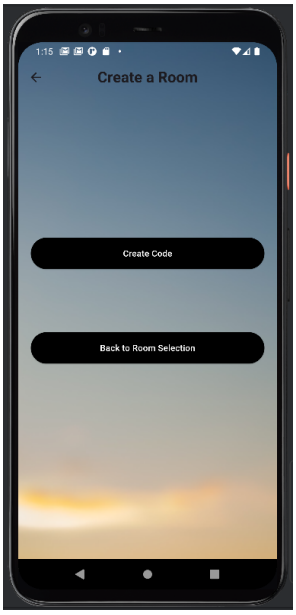

Fig. 46: App Platform Create Room Page:Part 2

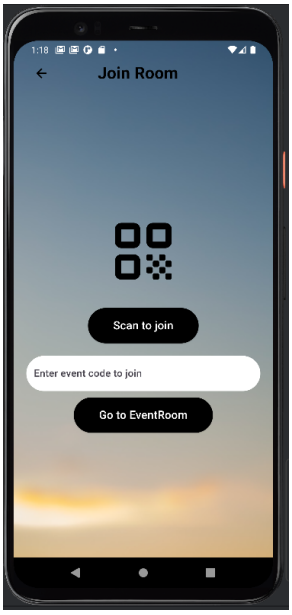

Fig. 47: App Platform Join Room Page

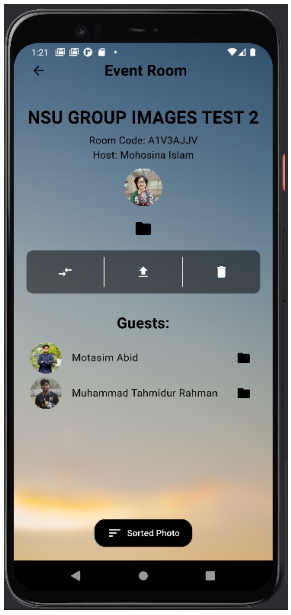

Fig. 48: App Platform Event Room as Host Page

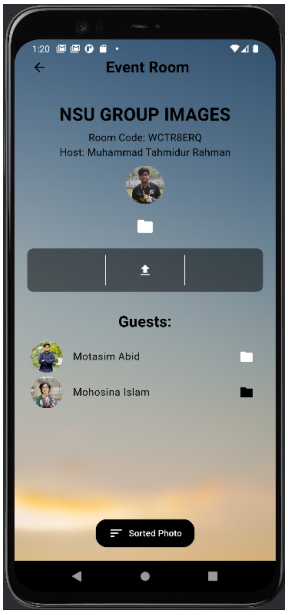

Fig. 49: App Platform Event Room as Guest Page

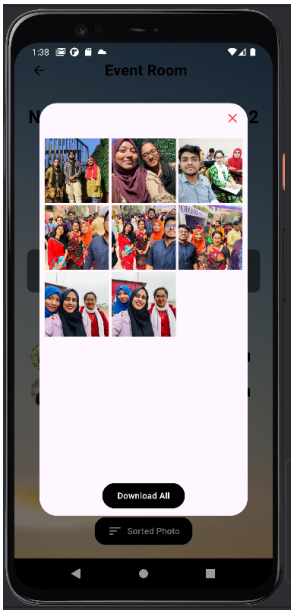

Fig. 50: App Platform Uploaded Photo Page

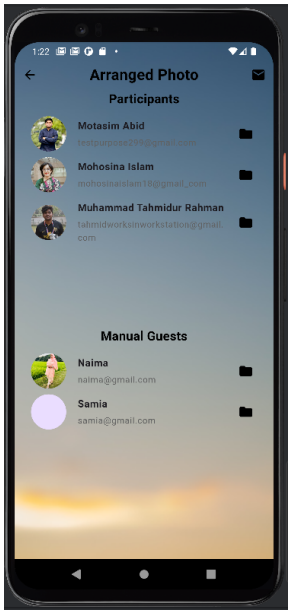

Fig. 51: App Platform Arranged Photo Page

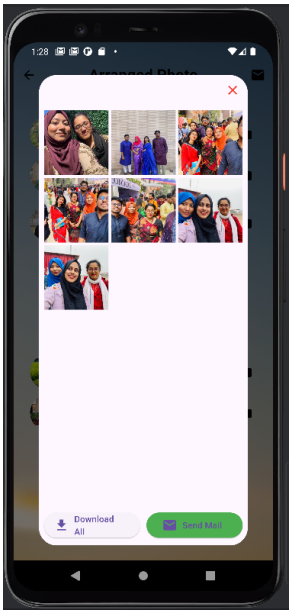

Fig. 52: App Platform Sorted Photo Page

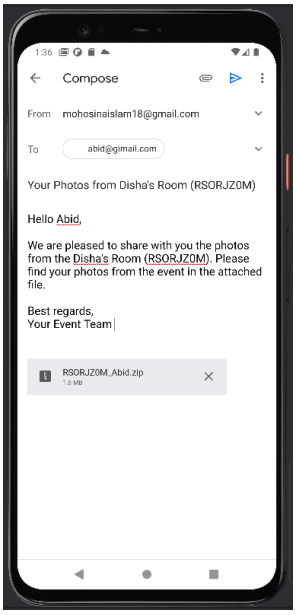

Fig. 53: App Platform Manual Mail Page

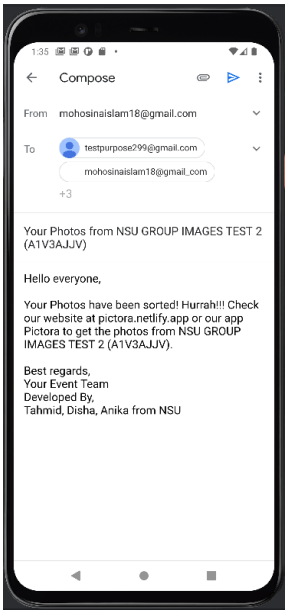

Fig. 54: App Platform Generalized Mail Page

E. Tech Stack

55

1)Languages

- Python
- Dart, Flutter (environment)
- JavaScript, HTML, CSS (for web app)

Libraries/Frameworks

- OpenCV
- facerecognition
- Pillow (library)
- NumPy (package)

2) Database

- Firebase
  - Storage
  - Realtime Database

3)Tools

- JSON
- pathlib
- Android Studio (for app development)

V. Results

The proposed automated facial recognition system was evaluated using two distinct test datasets, referred to as Test 1 and Test 2. These datasets were designed to assess the system's performance under varying conditions, including differences in image quality, participant profiles, and event settings. The results are presented through a combination of quantitative metrics and graphical representations to provide a comprehensive overview of the system's efficacy.

A. Performance Metrics

To quantitatively evaluate the system, the following metrics were employed:

- **Accuracy:** The proportion of correctly identified faces out of the total number of faces processed.
- **Matched Photos:** The number of photos successfully matched to participants.
- **Unmatched Photos:** The number of photos that could not be matched to any participant.

B. Test 1 Results

Test 1 was conducted using *Room 1: NSU GROUP IMAGES*, which included a total of 42 processed images. The system's performance on this dataset is summarized in Table I.

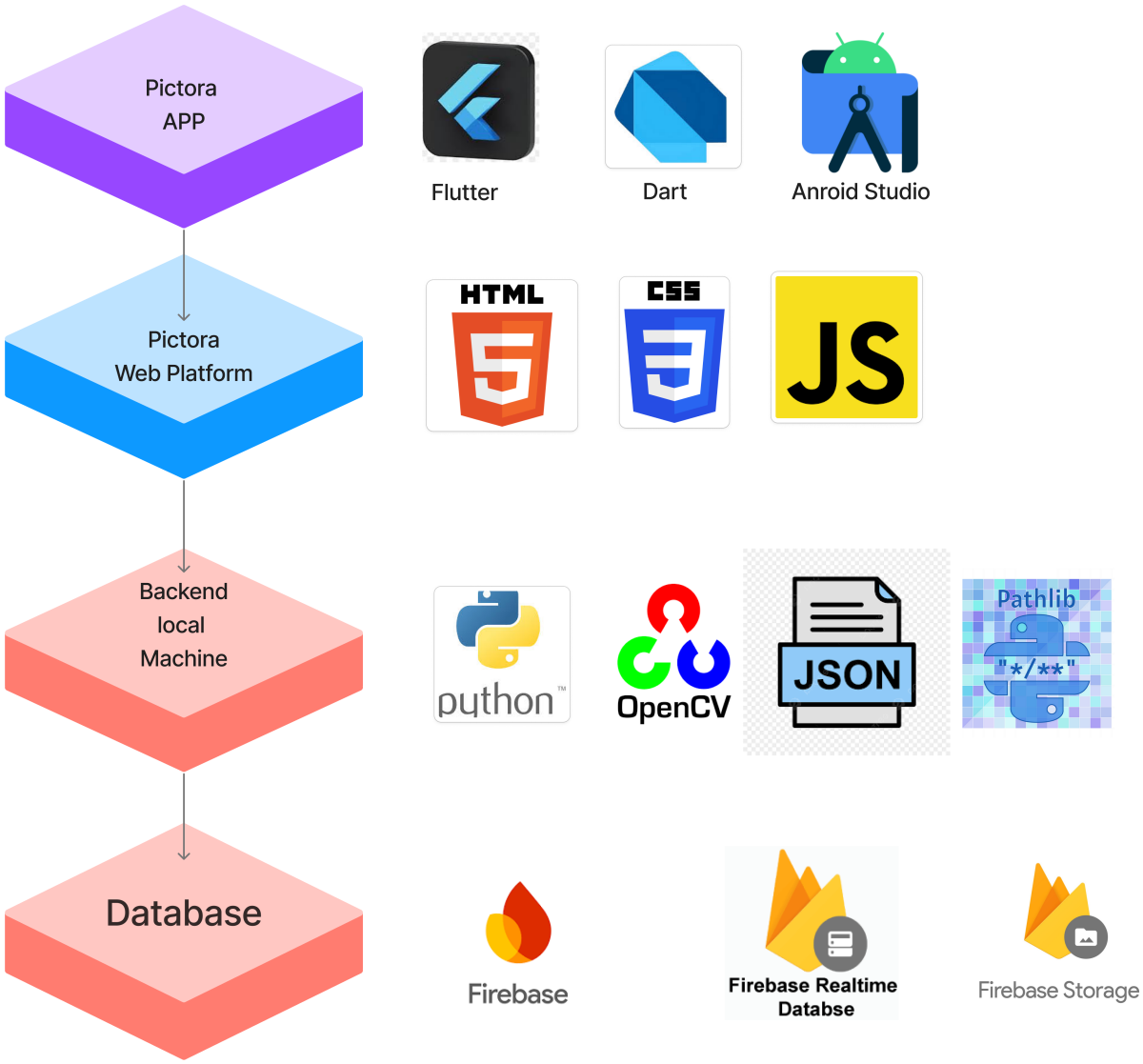

Fig. 55: Tech Stack for Entire Project

TABLE I: Performance Metrics for Test 1

Metric	Value
Total Images Processed	42
Matched Photos	40 (95.24%)
Unmatched Photos	2 (4.76%)
Accuracy	91.30%

1) Confusion Matrix for Test 1: Figure 56 illustrates the confusion matrix for Test 1 using a heatmap to represent the True Positives (TP), False Positives (FP), True Negatives (TN), and False Negatives (FN).

C. Test 2 Results

Test 2 utilized *Room 2: NSU GROUP IMAGES Test 2*, comprising a total of 24 processed images. The system's performance on this dataset is presented in Table II.

1) Confusion Matrix for Test 2: Figure 57 presents the confusion matrix for Test 2 using a heatmap to represent the True Positives (TP), False

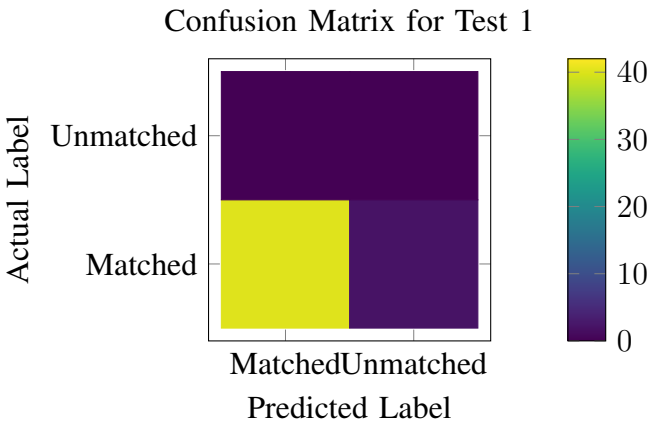

Fig. 56: Confusion Matrix Heatmap for Test 1

TABLE II: Performance Metrics for Test 2

Metric	Value
Total Images Processed	24
Matched Photos	22 (91.67%)
Unmatched Photos	2 (8.33%)
Accuracy	91.67%

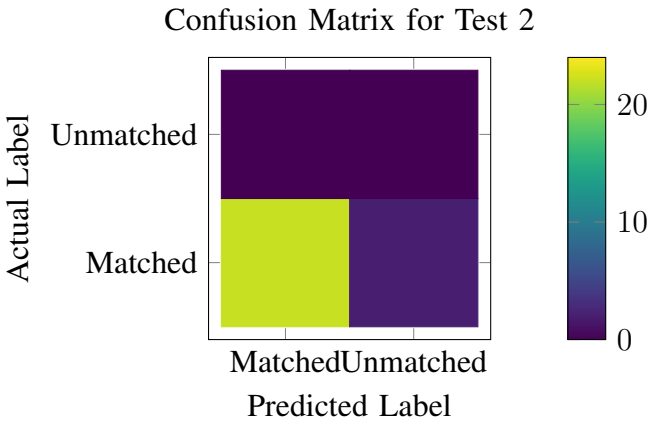

Fig. 57: Confusion Matrix Heatmap for Test 2

Positives (FP), True Negatives (TN), and False Negatives (FN).

D. Comparison of Test 1 and Test 2

To analyze the system's consistency and reliability, a comparative analysis between Test 1 and Test 2 was performed. Figure 58 provides a visual comparison of the accuracy and matched photo percentages between both tests.

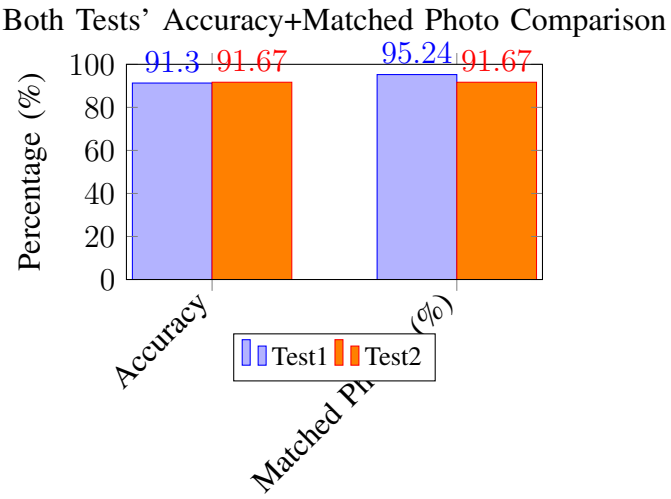

Fig. 58: Comparison of Accuracy and Matched Photos Between Test 1 and Test 2

VI. Conclusion and Future Work

In this project, we developed an automated facial recognition system integrated with a web platform [5] and a mobile application [6], collectively named Pictora [24], [25]. The system effectively addresses the challenges associated with organizing and managing event photos by leveraging advanced facial recognition technologies and scalable cloud-based infrastructure. Through the utilization of OpenCV for face detection, the `face_recognition` library [21] for embedding generation, and Firebase for real-time database management and storage [2] [13], our solution achieved high accuracy rates of over 90% in both test scenarios. This demonstrates the system's capability to accurately identify and categorize individual participants, thereby streamlining the photo retrieval process and enhancing user satisfaction.

Despite the promising results, there are several areas where the system can be further improved and expanded. Future work will focus on the following aspects:

- **Enhanced Facial Recognition Models:** Integrating more robust and scalable facial recognition models, such as those optimized for low-light conditions and varying angles, can significantly improve the system's accuracy and reliability in diverse event environments.
- **Support for Additional Image Formats:** Expanding the system's compatibility to include formats like HEIC by leveraging cloud-based processing resources can enhance its versatility and user accessibility.
- **Cloud-Based Backend Deployment:** Migrating the backend infrastructure from a local machine to cloud servers will address scalability and availability issues, ensuring continuous and efficient operation during large-scale events [22].
- **Real-Time Processing and Notifications:** Implementing real-time photo processing and instant notifications can provide users with immediate access to their curated photo collections, thereby improving the overall user experience.
- **User Interface Enhancements:** Refining the UI/UX design based on user feedback can further enhance the intuitiveness and functionality of both the web platform and mobile application [25].
- **Privacy and Security Enhancements:** Incorporating advanced security measures and privacy controls will ensure that user data is protected and that the system complies with relevant data protection regulations.
- **Integration with Social Media Platforms:** Facilitating seamless sharing of sorted photos to popular social media platforms can increase the system's utility and user engagement.
- **Automated Quality Assessment:** Developing algorithms to assess and enhance the quality of uploaded images can reduce the number of unmatched photos and improve overall system performance.

By addressing these areas, future iterations of Pictora will offer a more comprehensive and user-centric solution for event photo management. The ongoing development and optimization will ensure that the system remains robust, scalable, and adaptable to the evolving needs of event organizers and participants alike.

REFERENCES

[1] G. Bradski, "The OpenCV Library," *Dr. Dobb's Journal of Software Tools*, 2000. [Online]. Available: https://opencv.org

[2] Firebase, "Firebase Realtime Database," Google. [Online]. Available: https://firebase.google.com/products/realtime-database

[3] A. Geitgey, "face_recognition: Simple facial recognition library for Python," *GitHub*. [Online]. Available: https://github.com/ageitgey/face_recognition

[4] P. Viola and M. Jones, "Robust Real-Time Face Detection," *Int. J. Comput. Vis.*, vol. 57, no. 2, pp. 137–154, 2004.

[5] Pictora, "Pictora Web Platform." [Online]. Available: https://pictora.netlify.app

[6] Pictora, "Pictora Mobile Application." [Online]. Available: https://pictora-mobile-app-link.com

[7] Google Photos, "Organize and search your photos with AI." [Online]. Available: https://photos.google.com

[8] Apple, "Photos on iCloud." [Online]. Available: https://www.apple.com/icloud/photos/

[9] Shutterfly, "Event Photo Management." [Online]. Available: https://www.shutterfly.com

[10] SmugMug, "Photo Organization and Sharing." [Online]. Available: https://www.smugmug.com

[11] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 815–823.

[12] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "DeepFace: Closing the Gap to Human-Level Performance in Face Verification," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 1701–1708.

[13] Firebase, "Storage," Google. [Online]. Available: https://firebase.google.com/products/Storage

[14] S. Lee and J. Kim, "Performance Evaluation of Facial Recognition Models for Real-Time Applications," *IEEE Access*, vol. 8, pp. 123456–123465, 2020.

[15] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment using Multitask Cascaded Convolutional Networks," *IEEE Signal Processing Letters*, vol. 23, no. 10, pp. 1499–1503, 2016.

[16] J. Deng, J. Guo, Y. Zhou, J. Yu, I. Kotsia, and S. Zafeiriou, "RetinaFace: Single-Shot Multi-Level Face Localization in the Wild," in *Proc. IEEE/CVF Conf. Computer Vision and Pattern Recognition (CVPR)*, 2020, pp. 5203–5212.

[17] G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," *NeurIPS Deep Learning Workshop*, 2015.

[18] X. Hua, T. Mei, and S. Li, "Scalable Multimedia Retrieval," *Proc. IEEE*, vol. 101, no. 1, pp. 11–31, 2013.

[19] X. Hu, A. Singh, and D. Forsyth, "Natural Language-based Retrieval of Complex Video Content," *NeurIPS*, 2019, pp. 15754–15764.

[20] OpenCV, "Cascade Classifier." [Online]. Available: https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

[21] M. T. Rahman and M. I. Disha, "Face Recognition Using OpenCV Haar Cascade and Face Recognition Backend for Pictora." [Online]. Available: https://github.com/MuhammadTahmidurRahman/Face_Recognition_UsingOPENCV_HARCASCADE_FACE_RECOGNITION-pictorabackend-

[22] PythonAnywhere, "Host, run, and code Python in the cloud." [Online]. Available: https://www.pythonanywhere.com/

[23] BestFaceDetector, "What is the Best Face Detector?" *Medium*. (Accessed: Dec. 12, 2024). Available: https://medium.com/pythons-gurus/what-is-the-best-face-detector-ab650d8c1225

[24] M. T. Rahman, M. I. Disha, and A. Tabassum, "Pictora: Website Repository," *GitHub*. (Accessed: Dec. 12, 2024). Available: https://github.com/MuhammadTahmidurRahman/pictora

[25] M. T. Rahman, M. I. Disha, and A. Tabassum, "CSE-299-Face-Recognition-App," *GitHub*. (Accessed: Dec. 12, 2024). Available: https://github.com/MuhammadTahmidurRahman/CSE-299-FACE-RECPGNITION-APP