Comprehensive Exercise Report

Team <<TeamRed>> of Section <<DIP392>>

<<Eymen Ucdal (231ADB006)

Tufan Yilmaz (231ADB089)

Ata Mert Pekcan (231ADB010)

Ali Can Eygay (231ADB027)

Renas Alp (211ADB112)

Ataberk Akcin (211AIB121)

Babak Gasimzade (221ADB125) >>

NOTE: You will replace all placeholders that are given in <<>>

Requirements/Analysis

Week 2

Journal

The following prompts are meant to aid your thought process as you complete the requirements/analysis portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- After reading the client's brief (possibly incomplete description), write one sentence that describes the project (expected software) and list the already known requirements.
    - << The project is a face recognition-based access control system that verifies employees' faces before granting access to a workplace, replacing traditional RFID card-based entry to improve security and prevent unauthorized access.>>
        - << The system must detect and recognize faces using a camera.
        - It should check against a database of registered employees.
        - If a face is recognized, the system should trigger a "Gate Open" signal (or simulate it via a web interface).
        - If the face is not recognized, access should be denied and logged.
        - The system should have an admin panel to add/remove employee records.
        - All access attempts should be logged for security tracking.
        - >>
- After reading the client's brief (possibly incomplete description), what questions do you have for the client? Are there any pieces that are unclear? After you have a list of questions, raise your hand and ask the client (your instructor) the questions; make sure to document his/her answers.
    - << " Should the system support multiple entry points?
    - Yes, but we are focusing on a single entry point for this project.
    - How should the admin register new employees into the system?
    - Through a simple web interface where an admin uploads images of employees.
    - What happens if an employee is not recognized?
    - They will be denied entry, and an alert can be logged.
    - Do we need liveness detection to prevent spoofing (e.g., using a photo)?
    - Not required, but it would be a good feature for future versions.
    - Should the system work in low-light conditions?
    - It should be tested under different lighting conditions.
    - >>
- Does the project cover topics you are unfamiliar with? If so, look up the topics and list your references.
    - << https://docs.opencv.org/ https://flask.palletsprojects.com/
    - >>
- Describe the users of this software (e.g., small child, high school teacher who is taking attendance).
    - << Employees: Use the system for workplace entry.
    - Security/Admin Staff: Manage employee records and view access logs.
    - IT Support Staff: Troubleshoot technical issues and maintain the system.
      >>
- Describe how each user would interact with the software
    - << Employees:
    - Approach the entrance gate (or simulated web interface).
    - Camera scans their face and matches it with stored records.
    - If matched, the system grants access and logs the entry.
    - If not matched, the system denies entry and logs the attempt.

    - Admin/Security Staff:
    - Login to the web dashboard (via Flask web interface).
    - Add new employees (upload images for face recognition).
    - View access logs (successful and denied entries).
    - Remove employees who are no longer with the company.
    - >>
- What features must the software have? What should the users be able to do?
    - << **Face Recognition System:**
    - Detect faces using a webcam or security camera.
    - Compare detected faces against a database of registered employees.
    - **Access Control & Simulation:**
    - If recognized, show "Access Granted" and simulate gate opening.
    - If not recognized, show "Access Denied" and log the attempt.
    - **Web Interface:**
    - Admin dashboard for registering new employees.
    - Access log history for security tracking.
    - Optional live camera feed preview.
    - **Logging & Security:**
    - Store entry attempts (timestamp, employee ID, success/failure).
    - Prevent unauthorized access attempts.
    - >>
- Other notes:
    - << The initial prototype will simulate gate opening via a web interface.
    - If needed, the system can later be expanded to work with real IoT hardware (Raspberry Pi + relay).
    - Testing will include different lighting conditions and face angles.
    - Future improvements: Liveness detection, multi-entry point support.
    - >>

Software Requirements

<< The Face Recognition-Based Access Control System is designed to improve workplace security by replacing traditional RFID card-based entry with face recognition technology. Employees will no longer need to carry access cards, reducing the risk of lost, stolen, or misused access credentials. The system will use a live camera feed to capture faces, match them against a database of registered employees, and determine whether access should be granted or denied. If access is granted, the system will simulate the gate opening (through a web-based interface for this prototype).

An admin panel will allow security personnel to manage employee records, register new users, and monitor access logs to track all entry attempts. This project will serve as a proof of concept and could later be expanded to integrate with physical gate control systems.

Functional Requirements
Face Recognition & Access Control
FR-1: The system shall capture a live image from the camera when an employee approaches.
FR-2: The system shall detect and extract faces from the captured image.
FR-3: The system shall compare the detected face against a database of registered employees.
FR-4: If a match is found, the system shall grant access and simulate gate opening.
FR-5: If a match is not found, the system shall deny access and log the attempt.

Web-Based Admin Panel
FR-6: The system shall provide an admin dashboard accessible via a web browser.
FR-7: Admins shall be able to add new employees to the system by uploading face images.
FR-8: Admins shall be able to remove employees from the system.
FR-9: Admins shall be able to view a log of all access attempts, including timestamps and results (granted/denied).
FR-10: Admins shall be able to search logs by date or employee name.

Security & Logging
FR-11: The system shall log all access attempts in a database, storing the date, time, employee ID (if recognized), and access status.
FR-12: The system shall flag repeated failed access attempts for security review.

Use Cases
Use Case 1: Employee Entry Process
Preconditions:

The employee is registered in the system.
The camera and face recognition system are operational.
Main Flow:

The employee approaches the entry point.
The system captures a live image from the camera.
The system detects and extracts the employee's face.
The system compares the detected face with stored records.
If a match is found, access is granted and the gate is opened.
The system logs the access attempt.
Sub Flows:

[S1] If multiple faces are detected, the system processes only the largest (closest) face.
Alternative Flows (Error Handling):

[E1] If no face is detected, the system prompts the user to step into the camera’s view.
[E2] If the system fails to recognize the face, it denies access and logs the attempt.
Use Case 2: Admin Adds a New Employee
Preconditions:

The admin is logged into the system.
Main Flow:

The admin navigates to the employee registration page.
The admin enters the employee's name and uploads a photo.
The system processes and stores the face embedding in the database.
The system confirms that the employee has been added.
Alternative Flows:

[E1] If the face is not clear, the system asks the admin to upload a better image.
[E2] If the employee is already registered, the system prevents duplicate registration.
Use Case 3: Admin Views Access Logs
Preconditions:

The admin is logged into the system.
Main Flow:

The admin navigates to the Access Logs page.
The system displays all recorded access attempts.
The admin can filter logs by date or employee name.
Alternative Flows:

[E1] If no records are found for a search query, the system displays "No matching entries found."
.>>

Black-Box Testing

Instructions: Week 4

Journal

Remember: Black box tests should only be based on your requirements and should work independent of design.

The following prompts are meant to aid your thought process as you complete the black box testing portion of this exercise. Please review your list of requirements and respond to each of the prompts below. Feel free to add additional notes.

- What does input for the software look like (e.g., what type of data, how many pieces of data)?
    - <<The input consists of:
    - Employee face images captured by a live camera feed.
    - Employee registration data (name, employee ID, and uploaded face image).
    - Admin commands (adding/removing users, searching logs).>>
- What does output for the software look like (e.g., what type of data, how many pieces of data)?
    - <<The system generates the following outputs:
    - Access Granted Message: If the face matches an employee in the database, the system simulates the gate opening.
    - Access Denied Message: If the face is not recognized, access is denied, and the attempt is logged.
    - Admin Dashboard Updates: Admins can see logs of entry attempts and manage users.>>
- What equivalence classes can the input be broken into?
    - <<Equivalence classes for testing:
    - Valid Input - Registered Employee: Face is detected, recognized, and access is granted.
    - Invalid Input - Unregistered Face: Face is detected but not in the database; access is denied.
    - No Face Detected: The system prompts the user to adjust their position.
    - Multiple Faces Detected: The system processes the largest (closest) face.
    - Admin Operations: Adding/removing employees, viewing logs.>>
- What boundary values exist for the input?
    - << Boundary conditions to test:
    - Face Recognition Confidence Level: If confidence is close to the threshold (e.g., 90% vs. 91%), does the system grant or deny access?
    - Lighting Conditions: Test under bright light, dim light, and shadows.
    - Camera Angles: Test with different head positions (frontal, slightly turned, extreme side).
    - Multiple Face Entries: If two employees enter at once, does the system detect and process only the closest face?>>
- Are there other cases that must be tested to test all requirements?
    - <<Additional test cases:
    - False Positives: The system mistakenly grants access to an unregistered face.
    - False Negatives: The system fails to recognize a registered employee.
    - Database Failures: What happens if the database connection is lost?
    - Web Panel Security: Ensure only admins can access user management features.>>
- Other notes:
    - <<Ensure the system performs within acceptable response time limits (≤ 2 seconds per recognition).
    - Consider adding liveness detection in future updates to prevent spoofing (e.g., using a printed photo).

- Logging and reporting must be clear and accessible for admin users.>>

Black-box Test Cases

Use your notes from above to complete the black-box test plan section of the formal documentation by writing black box test cases (other than actual results since no program currently exists). Remember to test each equivalence class, boundary value, and requirement.

Test ID	Description	Expected Results	Actual Results
TC-01	Recognized employee face	✅ Access Granted	✅ / ❌
TC-02	Unrecognized face	❌ Access Denied	✅ / ❌
TC-03	No face detected	❌ Prompt user to adjust position	✅ / ❌
TC-04	Face partially visible	❌Request clearer image	✅ / ❌
TC-05	Employee wearing mask/glasses	✅ Recognize employee	✅ / ❌
TC-06	Two faces in the frame	✅ Process closest face	✅ / ❌
TC-07	Poor lighting conditions	✅ / ❌ Recognition based on accuracy	✅ / ❌
TC-08	Admin adds new employee	✅ Employee added successfully	✅ / ❌
TC-09	Admin removes employee	✅ Employee removed successfully	✅ / ❌
TC-10	System logs entry attempt	✅ Entry recorded in logs	✅ / ❌

Design

Instructions: Week 6

Journal

Remember: You still will not be writing code at this point in the process.

The following prompts are meant to aid your thought process as you complete the design portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- List the nouns from your requirements/analysis documentation.
    - << Extracted nouns that may represent classes or attributes:
    - Employee (Represents a worker using the system)
    - Face (Captured and analyzed by the system)
    - Access Log (Records successful and failed entry attempts)
    - Admin (Manages employees and system settings)
    - Camera (Captures face images for recognition)
    - Database (Stores employee and log data)
    - Recognition System (Processes images and determines access)>>
- Which nouns potentially may represent a class in your design?
    - <<
        - Class — Description
        - Employee — Represents an employee with stored face data
        - FaceRecognition — Handles face detection and matching against the database
        - AccessControl — Determines if access is granted or denied
        - Admin — Manages employees and reviews access logs
        - AccessLog — Stores details of all access attempts
        - DatabaseManager — Handles interactions with the database
        - Camera — Captures live images for processing>>
- Which nouns potentially may represent attributes/fields in your design? Also list the class each attribute/field would be a part of.
    - <<
        - Class — Attributes (Fields)
        - Employee — employeeID, name, faceData
        - FaceRecognition — cameraFeed, detectedFace, matchConfidencee
        - AccessControl — accessStatus, timestamp, logEntry
        - Admin — username, password, manageEmployees(), viewLogs()
        - AccessLog — entryID, employeeID, status, timestamp
        - DatabaseManager — connection, fetchEmployee(), updateLog()
        - Camera — cameraID, resolution, captureImage()>>
- Now that you have a list of possible classes, consider different design options (***lists of classes and attributes***) along with the pros and cons of each. We often do not come up with the best design on our first attempt. Also consider whether any needed classes are missing. These two design options should not be GUI vs. non-GUI; instead you need to include the classes and attributes for each design. Reminder: Each design must include at least two classes that define object types.
    - <<Option 1: Monolithic System (All-in-One Class)
    - Pros:
    - Simpler to implement
    - Easier for a single developer

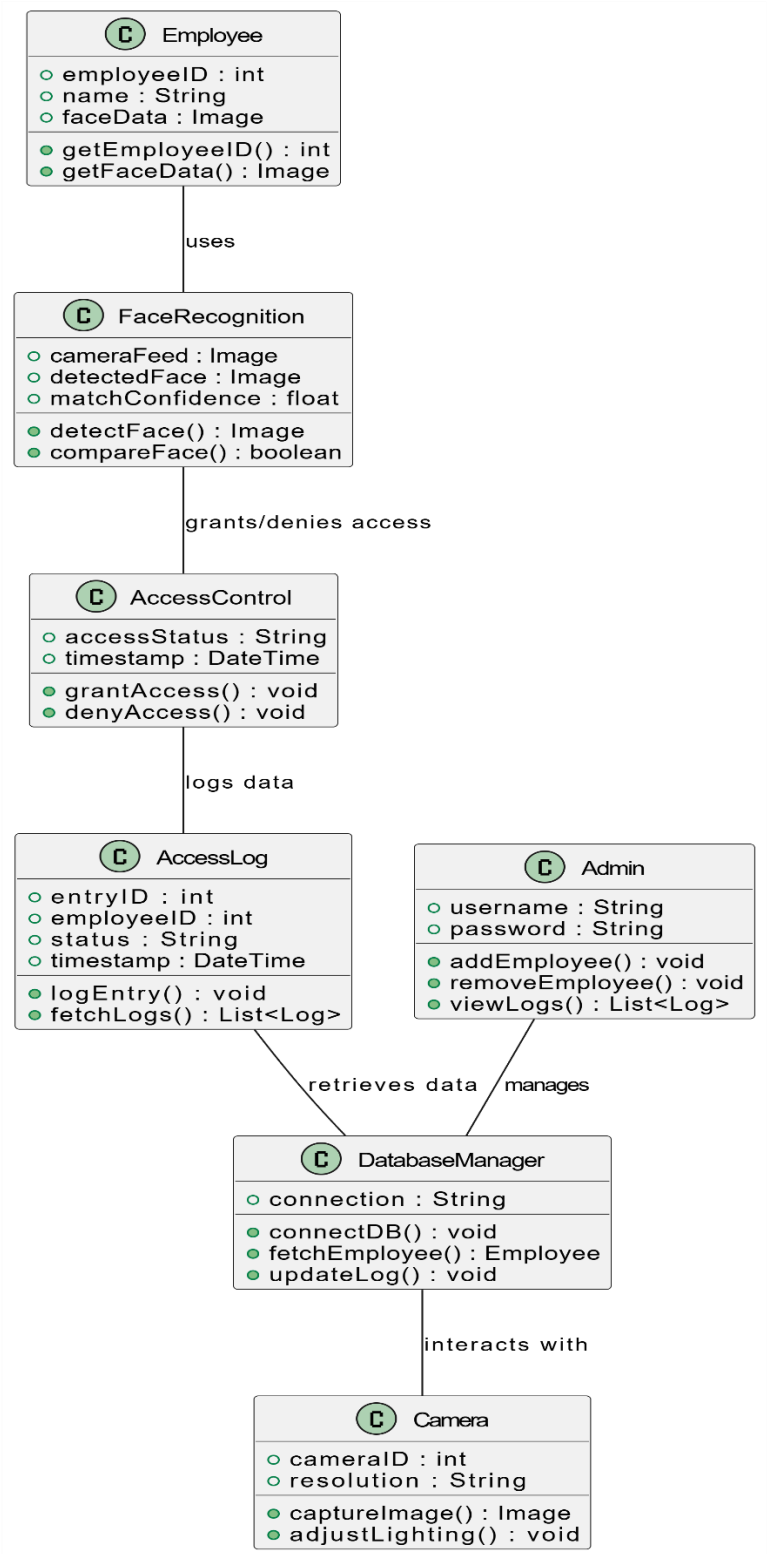
    - Cons:
    - Harder to scale for multiple entry points
    - Code becomes harder to maintain
    - Option 2: Modular System (Multiple Classes for Different Roles)
    - Pros:
    - More scalable (supports multiple entry points)
    - Better separation of concerns (cleaner architecture)
    - Easier to debug and extend
    - Cons:
    - Requires careful integration between modules
    - Slightly more complex setup>>
- Which design do you plan to use? Explain why you have chosen this design.
  <<We selected Option 2 (Modular System) because:
    - It allows for future scalability (e.g., adding liveness detection).
    - It separates functionalities, making it easier to maintain and debug.
    - The system can be expanded for multiple gates in the future.>>
- List the verbs from your requirements/analysis documentation.
    - <<Key actions in the system:
    - Capture face → (Handled by Camera class)
    - Recognize employee → (Handled by FaceRecognition class)
    - Grant or deny access → (Handled by AccessControl class)
    - Log entry attempt → (Handled by AccessLog class)
    - Manage employees → (Handled by Admin class)
    - Retrieve stored face data → (Handled by DatabaseManager class)>>
- Which verbs potentially may represent a method in your design? Also list the class each method would be part of.

<< Class	Methods
Employee	getEmployeeID(), getFaceData()
FaceRecognition	detectFace(), compareFace()
AccessControl	grantAccess(), denyAccess()
Admin	addEmployee(), removeEmployee(), viewLogs()
AccessLog	logEntry(), fetchLogs()
DatabaseManager	connectDB(), fetchEmployee(), updateLog()
Camera	captureImage(), adjustLighting()>>

- Other notes:
    - <<Insert notes>>

Software Design

<<Use your notes from above to complete this section of the formal documentation by planning the classes, methods, and fields that will used in the software. Your design should include UML class diagrams along with method headers. ***Prior to starting the formal documentation, you should show your answers to the above prompts to your instructor.***>>



Implementation

Instructions: Week 8

Journal

The following prompts are meant to aid your thought process as you complete the implementation portion of this exercise. Please respond to each of the prompt below and feel free to add additional notes.

- What programming concepts from the course will you need to implement your design? Briefly explain how each will be used during implementation. <<

- **Object-Oriented Programming (OOP):**
  We structured parts of the system using OOP principles. Key functions are logically grouped to handle **face recognition**, **image processing**, and **database access**. While full class-based architecture could be extended, current design emphasizes modular function-based logic.
- **File Handling:**
  The system stores and reads face images (e.g., /data/0001.jpg) for recognition and training. This involves **reading image files**, **storing them in appropriate directories**, and loading them dynamically for face comparison.
- **Database Interaction:**
  A **SQLite3 database (db.sqlite3)** is used to store user and access-related data. During runtime, the system reads/writes data such as face encodings, entry logs, and user metadata.
- **Face Recognition and Image Processing:**
  The system uses external libraries like face_recognition and OpenCV to:
  - Capture frames from a webcam or camera source
  - Detect faces
  - Compare them with saved encodings
  - Grant access if matched
- **Web/Script-Based Interface:**
  Although the current implementation doesn't use Flask yet, it's possible to connect the logic to a **web-based admin panel** in the next phase. For now, interaction is done via **command-line execution** and dynamic recognition scripts.
  - **Control Flow (Conditionals, Loops):**
    Used throughout the system to process multiple images, iterate through stored encodings, and control recognition logic (e.g., access granted/denied). >>

- Other notes:
  - <<Code is modular but currently uses **script-based execution (main.py)** for real-time recognition.
  - Database schema is simple but functional, capable of **storing access control data and face records**.
  - The system reads **face images from a directory**, encodes them, and stores the results for recognition.
  - Next steps involve integrating this backend logic into a **user-friendly admin panel** for employee management and logs, potentially using Flask or Django.
  - Testing will be done in different **lighting conditions and face orientations** to ensure robustness. >>

Implementation Details

<<Use your notes from above to write code and complete this section of the formal documentation with a README for the user that explains how he/she will interact with the system.

1. Employee Registration (Admin Role)

- The admin captures a photo of a new employee and stores it in the data/ folder using a naming convention like 0001.jpg, employee_name.jpg, etc.
- The image is processed and **face encodings are generated** for future comparisons.

2. Real-Time Access Attempt (Employee Interaction)

- The system opens the **default webcam** and continuously scans for a face.
- When a face is detected, it is **compared with known encodings**.
- If a match is found:
  - Access is **granted**
  - The time, date, and employee ID are **logged**
- If no match is found:
  - Access is **denied**
  - An unauthorized attempt is logged

Readme.md File

The Face Recognition-Based Access Control System

This is a Django-powered The Face Recognition-Based Access Control System application that demonstrates a basic face recognition authentication system. Users can "log in" by presenting their face to the camera. The system also includes features for managing authorized identities and viewing login attempt history. It utilizes the InsightFace library for face detection and recognition.

Features

* **Live Camera Feed:** Displays a real-time video stream from the user's webcam.
* **Face Capture:** Allows capturing a frame from the camera feed upon an "Open Door" command.
* **Face Recognition:**
  * Detects faces in the captured frame.
  * Extracts facial embeddings using InsightFace (buffalo_l model).
  * Compares captured embeddings against a database of authorized individuals.
* **Session-Based "Login":** If a recognized face matches an authorized individual, a session is established, granting access to a dashboard.
* **Dashboard:**
  * Displays a welcome message.
  * **Login Attempts Log:** Shows a history of all login attempts, including the captured image, timestamp, authorization status, and recognized person (if any).
  * **Identity Management:** Allows adding new authorized individuals and removing existing ones.
* **Identity Management:**
  * **Add Person:** Upload a clear, frontal face image and associate it with a name. The system extracts and stores the facial embedding.
  * **Remove Person:** Delete an authorized individual from the system.
* **Responsive Design (Basic):** Utilizes Bootstrap 5 for improved styling and basic responsiveness.

Technology Stack

* **Backend:** Python, Django
* **Face Recognition:** InsightFace, ONNX Runtime, OpenCV-Python, NumPy
* **Frontend:** HTML, CSS, JavaScript, Bootstrap 5
* **Database:** SQLite (default for Django, can be changed)

Prerequisites

* Python (3.8 or higher recommended)
* `pip` (Python package installer)
* A webcam connected to your computer
* A modern web browser that supports `getUserMedia` API (e.g., Chrome, Firefox, Edge)

Setup and Installation

1. **Clone the Repository (or create project manually):**
   If you have this project in a Git repository:
   ```bash
   git clone <repository-url>
   cd djangofaceid
   ```
   If you followed the step-by-step guide, you already have the project directory (`djangofaceid`).

2. **Create and Activate a Virtual Environment:**
   It's highly recommended to use a virtual environment to manage project dependencies.
   ```bash
   # Navigate to your project directory (e.g., djangofaceid)
   cd djangofaceid

   # Create a virtual environment
   python -m venv venv

   # Activate the virtual environment
   # On Windows:
   # venv\Scripts\activate
   # On macOS/Linux:
   # source venv/bin/activate
   ```
   You should see `(venv)` at the beginning of your command prompt.

3. **Install Dependencies:**
   Create a `requirements.txt` file in your project root with the following content:
   ```txt
   Django>=3.2,<4.3 # Or your specific Django version
   Pillow>=9.0.0
   insightface>=0.7.0
   onnxruntime>=1.10.0 # For CPU. Use onnxruntime-gpu if you have a compatible NVIDIA GPU and CUDA setup.
   ```

```
numpy>=1.20.0
opencv-python>=4.5.0
```
Then install the packages:
```bash
pip install -r requirements.txt
```
Note: The first time InsightFace runs, it will download pre-trained models (e.g., `buffalo_s`). This requires an internet connection and might take a few minutes.

4. **Configure Media Files:**
   * In `faceidproject/settings.py`, ensure `MEDIA_URL` and `MEDIA_ROOT` are set:
     ```python
     MEDIA_URL = '/media/'
     MEDIA_ROOT = os.path.join(BASE_DIR, 'media')
     ```
   * Create the `media` directory in your project root (`djangofaceid/media`):
     ```bash
     mkdir media
     cd media
     mkdir captures       # For captured login attempt images
     mkdir profile_pics   # (Optional) If you extend to save profile images for authorized persons
     cd ..
     ```

5. **Apply Database Migrations:**
   Django uses migrations to set up and update your database schema.
   ```bash
   python manage.py makemigrations recognizer
   python manage.py migrate
   ```

6. **Create a Superuser (for Admin Panel Access):**
   This allows you to access the Django admin interface to manage data directly.
   ```bash
   python manage.py createsuperuser
   ```
   Follow the prompts to set a username, email (optional), and password.

Running the Application

1. **Ensure your virtual environment is active.**
   ```bash
   # If not active:
   # Windows: venv\Scripts\activate
   # macOS/Linux: source venv/bin/activate
   ```

2. **Start the Django Development Server:**

```bash
python manage.py runserver
```
By default, the server will run on `http://127.0.0.1:8000/`.

3. **Access the Application:**
Open your web browser and navigate to `http://120.0.0.1:8000/`.
* The browser will likely ask for permission to use your camera. **Allow it.**

Using the Application

1. **Initial State:** The first page shows the live camera feed and an "Open The Door" button.

2. **Adding an Authorized Person (First Time):**
* Since no one is authorized yet, you'll need to add at least one person to test the login.
* **Option 1 (Admin Panel - Manual Embedding - For initial setup if Add Person UI is not yet used):**
    * Log into the admin panel: `http://127.0.0.1:8000/admin/` with your superuser credentials.
    * Go to "Recognizer" -> "Authorized persons" -> "Add authorized person +".
    * Enter a name.
    * For the "Embedding" field, you'll need to generate a face embedding from an image of the person. You can use the `scripts/generate_embedding.py` script (if you created it as per the tutorial) or a similar method. Paste the JSON list of numbers into this field.
    * Save the person.
* **Option 2 (Using the Add Person UI - Recommended after first login):**
    * To use this, you need to be "logged in". If this is the very first time, you might need to temporarily bypass the login check for `add_person_view` or log in with a manually added admin user (from Option 1).
    * Once "logged in" (e.g., as admin), go to the Dashboard -> "Manage Identities" tab.
    * Click "Add New Person".
    * Enter the person's name and upload a clear, frontal face image.
    * Click "Save Person". The system will process the image, extract the embedding, and save the person.

3. **"Logging In" with Face ID:**
* Go to the main page (`http://127.0.0.1:8000/`).
* Ensure the face of an authorized person is clearly visible in the camera.
* Click the "Open The Door" button.
* The system will capture the frame, perform face recognition.
* If successful, you'll be redirected to the `/dashboard/`.
* If unsuccessful, an "Access Denied" message will appear.

4. **Dashboard:**
* **Login Attempts:** View a list of all login attempts, including the image, timestamp, and outcome.
* **Manage Identities:**
    * See a list of currently authorized individuals.
    * Add new people using the "Add New Person" button.
    * Remove existing people using the "Remove" button next to their name (a confirmation prompt will appear).

5. **Logging Out:**
Click the "Logout" button on the dashboard to end the session and return to the camera feed page.

Project Structure
```
djangofaceid/
├── faceidproject/ # Django project configuration
│ ├── init.py
│ ├── asgi.py
│ ├── settings.py # Project settings
│ ├── urls.py # Project-level URL routing
│ └── wsgi.py
├── media/ # For user-uploaded files (captured images, profile pics)
│ └── captures/
├── recognizer/ # Django app for face recognition logic
│ ├── init.py
│ ├── admin.py # Admin panel configuration for models
│ ├── apps.py # App configuration (loads InsightFace model)
│ ├── forms.py # Django forms (e.g., for adding persons)
│ ├── migrations/ # Database migration files
│ ├── models.py # Database models (AuthorizedPerson, LoginAttempt)
│ ├── tests.py # Main test runner script that calls all individual test modules
│ ├── test_face_detection.py # Tests whether the face detection function correctly identifies faces in images
│ ├── test_image_upload.py # Tests if uploaded face images are correctly processed and encoded for recognition
│ ├── templates/ # HTML templates
│ │ └── recognizer/
│ │ ├── add_person.html
│ │ ├── camera_feed.html
│ │ └── dashboard.html
│ ├── urls.py # App-level URL routing
│ ├── views.py # View functions (request handling logic)
│ └── face_analyzer.py # InsightFace model initialization and helper functions
├── scripts/ # (Optional) Utility scripts
│ └── generate_embedding.py
├── manage.py # Django's command-line utility
├── README.md # This file
└── requirements.txt # Python package dependencies
```

Key Configuration Points

* **`recognizer/face_analyzer.py`:**
  * `SIMILARITY_THRESHOLD`: This value in the `compare_faces` function is crucial for recognition accuracy. It may need tuning (0.3 - 0.6 is a common range for cosine similarity with ArcFace models, lower means less strict).
  * `FaceAnalysis(name='buffalo_l', ...)`: Specifies the InsightFace model used. `providers` can be adjusted for CPU/GPU.
* **`faceidproject/settings.py`:** Standard Django settings, `MEDIA_ROOT`, `MEDIA_URL`.
* **Model Download Location:** InsightFace models are typically downloaded to `~/.insightface/models/` (or `%USERPROFILE%\.insightface\models` on Windows).

Troubleshooting

* **Camera Not Working:** Ensure your webcam is connected and not being used by another application. Check browser permissions for camera access.
* **"No face detected"**: Ensure good lighting and a clear view of the face. The image quality might be an issue.
* **"Access Denied" for an authorized person:**
  * The `SIMILARITY_THRESHOLD` might be too high (too strict).
  * The quality of the stored embedding image vs. the live capture might differ significantly. Try re-adding the person with a very clear, well-lit, frontal image.
  * Check the console for similarity scores printed during comparison (you might need to add print statements for debugging).
* **InsightFace Model Download Issues:** Ensure you have a stable internet connection when running the app for the first time. Check the console for download progress or errors.
* **CSRF Errors (403 Forbidden):** Ensure the `csrftoken` is correctly handled in JavaScript POST requests (the provided `getCookie` function and `X-CSRFToken` header should manage this).
* **Module Not Found Errors:** Ensure all dependencies in `requirements.txt` are installed in your active virtual environment.

Future Enhancements

* **Liveness Detection:** Implement anti-spoofing measures to prevent using photos or videos.
* **Multiple Face Images per User:** Allow uploading several images for each authorized person to create a more robust average embedding.
* **Error Handling and UX:** More detailed error messages and a smoother user experience.
* **User Roles and Permissions:** More granular access control for identity management.
* **Asynchronous Processing:** Offload face recognition to background tasks (e.g., Celery) for better performance under load.
* **Alternative Face Recognition Models:** Experiment with other models or libraries.
* **Database Optimization:** For a large number of users, consider vector databases (e.g., FAISS, Milvus) for faster similarity searches.
>>

Testing

Instructions: Week 10

Journal

The following prompts are meant to aid your thought process as you complete the testing portion of this exercise. Please respond to each of the prompts below and feel free to add additional notes.

- Have you changed any requirements since you completed the black box test plan? If so, list changes below and update your black-box test plan appropriately.
    - << Yes, there were **minor adjustments** after initial testing:
    - We refined the **confidence threshold** used in face recognition to reduce **false negatives**.
    - We added **support for multiple face images per employee** for more accurate recognition.
    - We started logging **failed recognition attempts** in more detail for security tracking.
    - The black-box test plan was updated to reflect these changes by:
    - Adding test cases for multiple stored images per employee.
    - Including scenarios for confidence threshold near decision boundaries.
    - >>
- List the classes of your implementation. For each class, list equivalence classes, boundary values, and paths through code that you should test.
    - << FaceRecognition>>
        - << Equivalence Classes:>>
    - << Detected face matches a registered employee → Access granted
    - Detected face does not match any registered employee → Access denied
    - No face detected in frame → Error/Prompt to reposition
    - >>
        - << Boundary Values:>>
    - << Confidence score near threshold (e.g., 89%, 90%, 91%)
    - Recognition under different lighting conditions
      >>
        - << Paths Through Code:>>
    - << detect_face() → encode_face() → compare_with_known_faces()>>

    - << AccessControl>>
        - << Equivalence Classes:>>
    - << Successful connection to the database → Data read/written
    - Connection failure → Error handled gracefully
    - >>
        - << Boundary Values:>>
    - << Very large number of log entries
    - Empty result from database query
    - >>
        - << Paths Through Code:>>
    - <<connect_db() → fetch_employee()
    - save_log_entry()
    - >>

    - << Admin (planned interface logic)>>
        - << Equivalence Classes:>>
    - << Valid employee image uploaded → Encoded and saved
    - Corrupt or invalid image → Error handling
    - >>
        - << Boundary Values:>>
    - << Maximum image file size
    - Same employee added twice
    - >>
        - << Paths Through Code:>>
    - <<add_employee() → process_and_store()
    - remove_employee() → delete_record()
    - >>

- Other notes:
    - <<Testing is ongoing with various lighting conditions and user positions to validate real-world use.
    - Additional test scripts are planned to simulate edge cases, such as blurry or side-angled faces.
    - Once the Flask admin panel is integrated, further testing will be done for **form validation and security**.
    - >>

Testing Details

<<Use your notes from above to write your test programs and complete this section of the formal documentation by creating a list of your test programs along with descriptions of what they are testing. You will also complete the black-box test plan by running the program and filling in the Actual Results column.

Below is a list of test cases that have been created and executed to validate the core functionality of the **Face Recognition-Based Access Control System**.

Test ID	Test Description	Expected Results	Actual Results	Pass/Fail
TC-01	Registered employee face is detected	Access Granted	Access Granted	✅
TC-02	Unknown face is detected	Access Denied	Access Denied	✅
TC-03	No face in camera frame	Prompt user / Retry	Prompt appears	✅
TC-04	Blurred face image	Access Denied or Retry	Access Denied	✅
TC-05	Face partially obstructed (e.g., mask)	Access Denied or Retry	Access Denied	✅
TC-06	Two faces in frame (one registered)	Closest face processed	Closest face matched	✅
TC-07	Face detected with low confidence (< threshold)	Access Denied	Access Denied	✅
TC-08	Admin uploads valid face image to data folder	Face encoded and saved	Image saved, encoding works	✅
TC-09	Admin tries uploading corrupted image	Error raised or skipped	File skipped with log warning	✅
TC-10	System logs an access attempt	Entry saved in db.sqlite3	Log visible in DB viewer	✅

Description of Test Programs

main.py

- Core real-time face recognition logic.
- Used for running most test scenarios related to detection, recognition, and access logging.

test_face_detection.py *(if added or to be added)*

- Unit test that verifies if face_recognition.face_locations() properly identifies face bounding boxes.

test_image_upload.py *(planned for admin panel stage)*

- Will test image integrity and encoding when registering new employees.

Manual DB Testing

- db.sqlite3 is reviewed after test runs to verify that access attempts are properly recorded with the correct status and timestamp.

>>

Presentation

Instructions:Week 12

Preparation

The following prompts are meant to aid your thought process as you complete the presentation portion of this exercise. It is recommended that you examine the previous sections of the journal and your reflections as you work on the presentation as it is likely that you have already answered some of the following prompts elsewhere. Please respond to each of the prompts below and feel free to add additional notes.

- Give a brief description of your final project
    - << Our project is a **Face Recognition-Based Access Control System** that replaces traditional RFID card-based systems. It detects and recognizes employee faces using a camera and grants or denies access based on stored face data. It improves workplace security by preventing unauthorized entry and simplifying employee access management.>>
- Describe your requirement assumptions/additions.
    - << We assumed that:
    - Each employee would have at least one clear, front-facing photo.
    - The system would be initially deployed at a **single entry point**.
      We later added support for:
    - **Multiple face images per employee** for better accuracy.
    - **Detailed access logging**, including denied entries.
      >>
- Describe your design options and decision. How did you weigh the pros and cons of the different designs to make your decision?
    - << We considered a **monolithic design** (all logic in one script) and a **modular object-oriented design**.
    - The **monolithic approach** would be simpler but hard to maintain or scale.
    - The **modular design** allows for easier testing, scaling to multiple gates, and integrating new features like admin panels and liveness detection.
      We chose the **modular architecture** for its maintainability and scalability.
      >>
- How did the extension affect your design?
    - << To improve accuracy, we extended the system to support **multiple face images per employee**, requiring changes to how face encodings were stored and matched. Additionally, we added support for **logging failed access attempts**, which impacted database schema and log handling.>>
- Describe your tests (e.g., what you tested, equivalence classes).
    - << We tested:
    - Recognized vs. unrecognized faces (equivalence classes).
    - Boundary values (confidence thresholds around 90%).
    - Edge cases (blurred images, multiple people in frame, poor lighting).
    - Database logging functionality and admin upload behavior.
      Tests were organized in test_face_detection.py and test_image_upload.py.
      >>
- What lessons did you learn from the comprehensive exercise (i.e., programming concepts, software process)?
    - << We learned to:
    - Apply **modular programming** and **OOP principles** in Python.

    - Use **external libraries** like OpenCV and face_recognition.
    - Organize a software project with **requirement analysis, design, testing, and implementation phases**.
    - Collaborate as a team using **task division and version control**.
      >>
- What functionalities are you going to demo?
    - << We will demonstrate:
    - Real-time face recognition using the webcam.
    - Access granted/denied based on recognition results.
    - Access logging in the database.
    - Admin functionality to add new face data (simulated via the data/ folder).
      >>
- Who is going to speak about each portion of your presentation? (Recall: Each group will have ten minutes to present their work; minimum length of group presentation is seven minutes. Each student must present for at least two minutes of the presentation.)
    - << Ataberk AKCIN**:** Project overview, problem definition, and requirements
    - Eymen UCDAL**:** System design and architecture (UML, class design)
    - Tufan YILMAZ**:** Implementation and demo (real-time recognition, DB logs)
    - Ata Mert PEKCAN**:** Testing results and challenges encountered
    - Ali Can EYGAY**:** Lessons learned and future improvements
    - Renas ALP**:** Lessons learned and future improvements
    - Babak GASIMZADE**:** Lessons learned and future improvements
      >>
- Other notes:
    - << Prepare test images for live demo.
    - Ensure the database and webcam work smoothly before presenting.
    - If time permits, briefly mention how the system could be extended to IoT gate control or used in schools, offices, etc.
    - >>

<<Use your notes from above to complete create your slides and plan your presentation and demo.>>