GNG1103

Design Project User and Product Manual

The E-Key Two-Factor Authentication for the Modern Office

Submitted by:

E CUBED GROUP 2

EMMA BELAL 300029994

LOUIS-PHILIPPE KEITH, 300445603

RARES CONSTANTIN SERBAN, 300418847

December 3rd, 2024

University of Ottawa

Table of Contents

Table of Contents ii

List of Figures 3

List of Tables vi

List of Acronyms and Glossary viii

1 Introduction 1

2 Overview 1

2.1 Conventions 2

2.2 Cautions & Warnings 2

3 Getting started 2

3.1 User Interface 4

3.2 Employee Registration 4

3.3 Configuration Considerations 11

3.4 User Access Considerations 11

3.5 Accessing/setting up the System 11

3.6 System Organization & Navigation 12

How to register an employee 13

1. Click ‘Admin Access’ 13

2. Log into Admin 13

3. Click ‘Register User’ 14

4. Enter all the information for the new user 14

5. Click register and wait for confirmation pop up 23

How to update user information 15
1. Click ‘Admin Access’ to log in 15
2. Log into Admin 16
3. Click ‘Update User’ 16
4. Enter username 17
5. Change the information and click ‘Update’ 17
How to check employee location 18
1. Click ‘Admin Access’ to log in 18
2. Log into Admin 19
3. Click ‘Locations’ 19
4. Click ‘View Full Map’ 20
How to Test facial recognition 20
1. Click ‘Admin Access’ to log in 20
2. Log into Admin 21
3. Click ‘Facial Recognition’ 25
4. Enter credentials 22
How to check employee logs 23
1. Click ‘Admin Access’ 23
2. Log into Admin 23
3. Click ‘Logs’ 24
3. Open the Downloaded Document with the Logs. 24
Employee access 25
1. Click ‘Employee Access’ 25

2. Log into Employee Login with Credentials 25

3. You can choose between facial recognition or location which works the same way as administration. If you want further detail look at the administration location and administration facial recognition. 26

3.7 Exiting the System 26

4 Using the System 28

4.1 Location Tracking 29

4.2 Facial Recognition 30

4.3 Notification 31

4.4 UI/Web Application 31

5 Troubleshooting & Support 31

5.1 Error Messages or Behaviors 32

5.2 Special Considerations 33

5.3 Maintenance 33

5.4 Support 33

6 Product Documentation 33

6.1 main.py 34

6.2 admin_locations.py 40

6.3 employee_location.py 43

6.4 facial_recognition.py 44

6.5 bandwidth.py 50

6.6 homepage.html 51

6.7 admin_login.html 54

6.8 admin_dashboard.html 56
6.9 register_user.html 59
6.10 update_user.html 63
6.11 locations.html 69
6.12 employee_login.html 71
6.13 employee_dashboard.html 74
6.14 config.json 77
6.15 Bill of Materials (BOM) 78
BOM 78
Equipment list 78
6.16 Testing & Validation 78
7 Conclusions and Recommendations for Future Work 82
8 Bibliography 83
APPENDIx 84
9 APPENDIX I: Design Files 84

List of Figures

Figure 1. Image of the Homepage of the Web Application. 2

Figure 2. Home Screen 5

Figure 3. Admin Access Login 5

Figure 4. Admin Dashboard 6

Figure 5. Register a User 6

Figure 6. User Data Base 7

Figure 7. Admin Updating the User 7

Figure 8. Admin Updating User Part 2 8

Figure 9. Locations 8

Figure 10. Facial Recognition 9

Figure 11. Access Attempt Logs 9

Figure 12. Employee Login 10

Figure 13. Employee Dashboard 10

Figure 14. Highlighting the exit button of the system 27

Figure 15. Example of Correct Access Assignment 79

Figure 16. Example of Correct UAAE Assignment 79

Figure 17. Response Time to Unauthorized and Authorized Access Attempts 80

Figure 18. Image of Access Log after Consecutive Log in Attempts 80

List of Tables

Table 1. Acronyms viii

Table 2. Glossary viii

Table 3. Results from Medium Fidelity Comprehensive Prototype 81

Table 4. Referenced Documents 84

List of Acronyms and Glossary

Table 1. Acronyms

Acronym	Definition
SMS	Short Message Service
UAAE	Unauthorized Access Event
UAE	Un-Authorized Employee
UI	User Interface

Table 2. Glossary

Term	Acronym	Definition
Minimal viable product	MiVP	A product that is python code that can notify the administration of an unauthorized access attempt in an Enterprise that has a good notification.
Unauthorized Access Attempt	UAA	is when someone tries to enter a room with a door code either using someone else's information or without a code at all.

1 Introduction

This User and Product Manual (UPM) provides the information necessary for administrative staff for office enterprises, and their bosses. For administrators to effectively use the E Cubed Security System and for prototype documentation.

Industries have many reasons to restrict access to an area like hazardous materials, valuables, and sensitive instruments. E-Cubed has developed an advanced security system to help protect against unauthorized people from entering restricted areas.

In this user manual you will find instructions on how to use our user interphase. The description of our three subsystems and how they work together to deliver our final product. The two cases. Case 1: a person tries to have access to a door but is not recognized by the facial recognition system then a message will be sent to administration. Case 2: a person tries to access a door with our system in place and the camera system recognizes them, then access will be granted, and the person will be let in.

2 Overview

Enterprises often need to be able to restrict and permit access to multiple areas for their employees. Many of the ways to restrict areas are rudimentary, such as signage and a simple door pin. There is a lack of authentication in these systems and can be subjected to fraud. Someone stealing an employee's door pin to enter an area can happen without consequence. Having a security system that can confirm that the information enters corresponds to a specific employee, as well as tracking their location, can provide the Enterprise with confidence.

The user needs a system that can authentic each employee and log their information including an image to a directory. Additionally, the user needs to be able to use existing systems, such as a door pin, and integrate a two-factor authentication system to prevent code theft. This prevents the user from needed to implement an entirely new system just to be secure. The user also needs a simple but effect way of logging, tracking and registration of employees. Since many Enterprises would not want to allocate parts of the budget to train administrators in python just to operate a security system. This system has a simple interface and clear visuals to help the admin fulfill the required tasks to allow and restrict access as they please. In addition to this, Enterprise might need to track their employee's location at specific times.

Our product can seamlessly fit into existing door pin systems. Which allows for the companies to simply upgrade their security systems with the addition of cameras. Our security system

Figure 1. Image of the Homepage of the Web Application.

2.1 Conventions

In this manual when the user needs to start a task it will be found in a numbered list. Each number indicating the step that they are on.

2.2 Cautions & Warnings

There needs to be the use of phone numbers, and images of users. In addition, you also allow us to take videos to send notifications to administration. Our system should only be used in a public space where people expect to be filmed.

3 Getting started

For the first time you use the product, your database will be empty, so there'll be no registered user to sign in with. At that time, please contact us, and we'll be giving you some temporary admin credentials, enough so you can register yourself properly in your newly created database. We'll be providing you with all the fields' information. For privacy and security purposes, we strongly encourage you to delete this given data as soon as you can properly register an ADMIN user.

Once you open the application, you will see a homepage screen with two buttons: "Admin Access" and "Employee Access". Every time you press the green "Home" button throughout the application, you will be redirected to this page.

If you press "Admin Access", you will have to log in. There you will enter your username and password. If you press the red "Cancel" button, you will go back to the homepage. After logging in, you will enter the "admin dashboard". The buttons you see represent all the actions you can take as an Admin user. Firstly, you can register a new user. Only admin can do that. If you're unsure if the user already exists or about some of his credentials (if he/she exists), you can first click on "Update User" and retrieve all the information of the username you entered, apart from some private credentials.

For the location feature of the application, admin can see the locations of all registered devices. Please refer to location instructions for more details about the setup. When you click on the "View Full Map" button, you will either see a map with some pins or a blank page saying, "Waiting for location data". In that case, simply run the Shortcut on your mobile device. If the page cannot be found, make sure to run all the corresponding Python files in separate terminals. When you hover over a pin, you will see the device name (e.g. 12), along with a timestamp of the latest location update. Note that with the Shortcut automation admin can simply write a text (e.g. Send location) to the employee and the Shortcut automatically runs.

For Facial Recognition, admin and employees use the same version of the feature. After clicking on the "Facial Recognition" button, you will see a pop-up with a username and pin slots to fill. When doing so, the webcam will open and ty to recognize the presented face. If it does it successfully, an Access Granted window will be displayed. Otherwise, a short video of the camera's visual field will be captured and sent to admin via email. Please refer to Twilio instructions for more details about the setup.

The “Logs” button shows the admin a file with all the access attempts for the Face ID system and with appropriate information about each entrance event.

After logging in as an employee, the user can access the same Face ID system as previously explained. However, after accessing the location system, the employee will see only his location on the map.

This was a brief overview of some of the main application functions. Please look forward to more information on how to set your system up and running.

3.1 User Interface

This allows the administrator to use the product without any knowledge of coding. The addition of a UI also fulfils the client's need to have an application. Our interface is a web application, we chose this to avoid having system integration errors and increase the flexibility of the product. In fact, even though it’s harder to develop, a web application allows for easier access across a wider range of devices. The compatibility with the operating system of the device is not a concern. It is also easier to push updates and debug than a desktop application. In summary, for applications that prioritize accessibility, ease of use, and real-time updates, web applications are a compelling choice.

Features of the interface include:

- Login page for both administrators and employees
- Administrator’s employee registration form
- Administrator's employee location request page
- Log files of all access attempts for the week
- Door access pin prompt
- Facial recognition verification system

3.2 Employee Registration

This allows the administrator to register and unregister employees.
Every security system should have this feature. After clicking on the “Register User” button from the admin dashboard, a registration form will appear, as you can see on *Figure 5*. All fields are mandatory. If an empty field is left out, an error pop-up message will appear on your screen. Similarly, if the username entered already exists in the database, a UNIQUE CONSTRAINT message will also appear, restricting you from registering the user. If the registration is successful, you will see this notification: “User <_> has been registered successfully”. Note that the “Access Level” in the registration form is a scroll down menu.

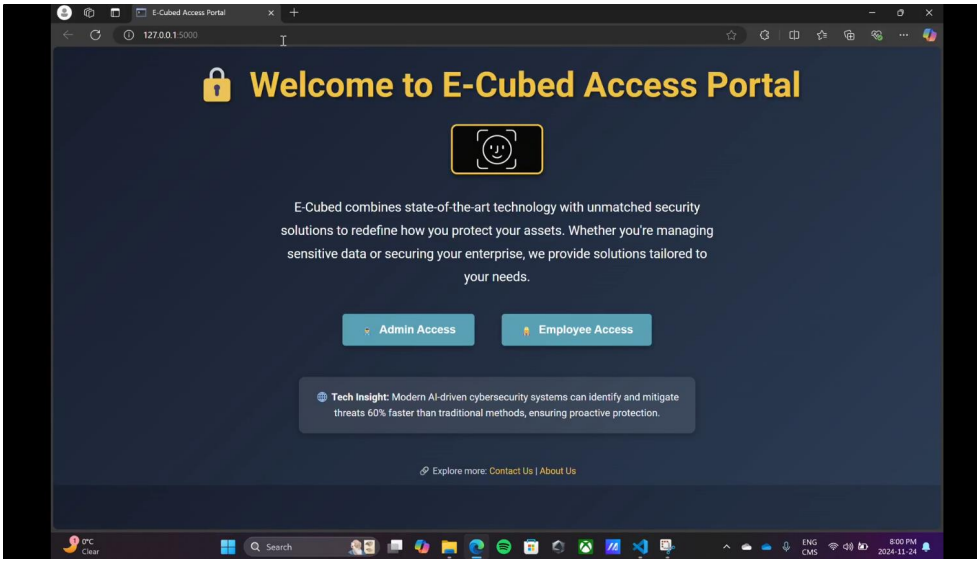


Figure 2. Home Screen

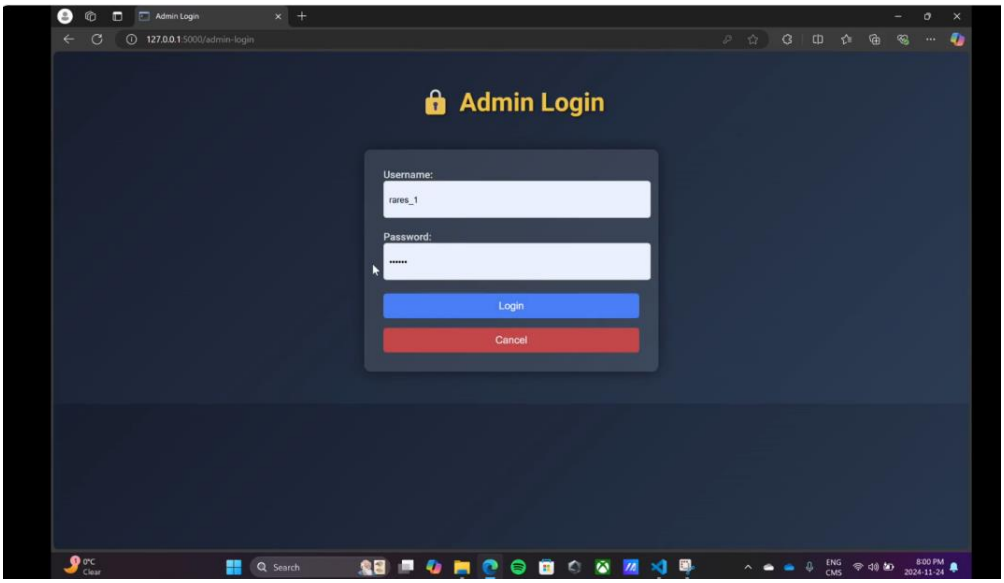


Figure 3. Admin Access Login

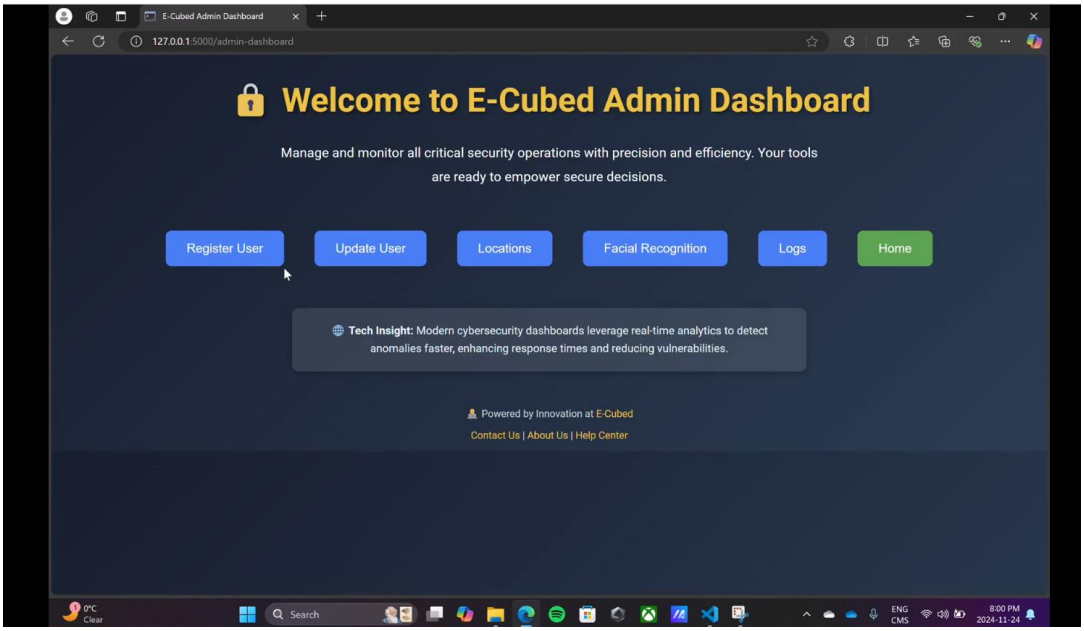


Figure 4. Admin Dashboard

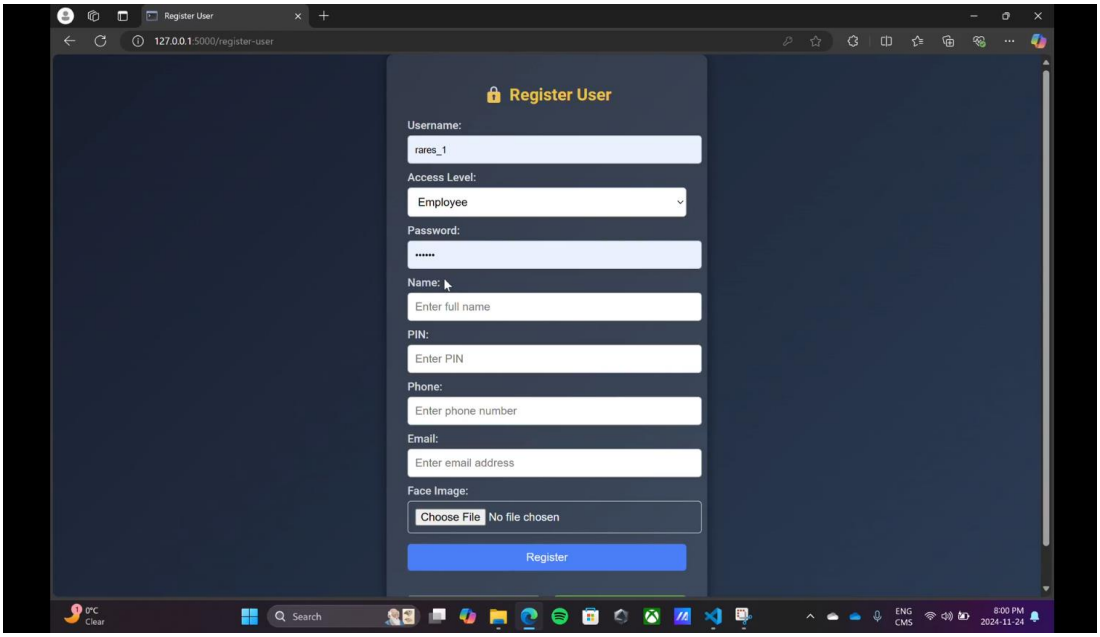


Figure 5. Register a User

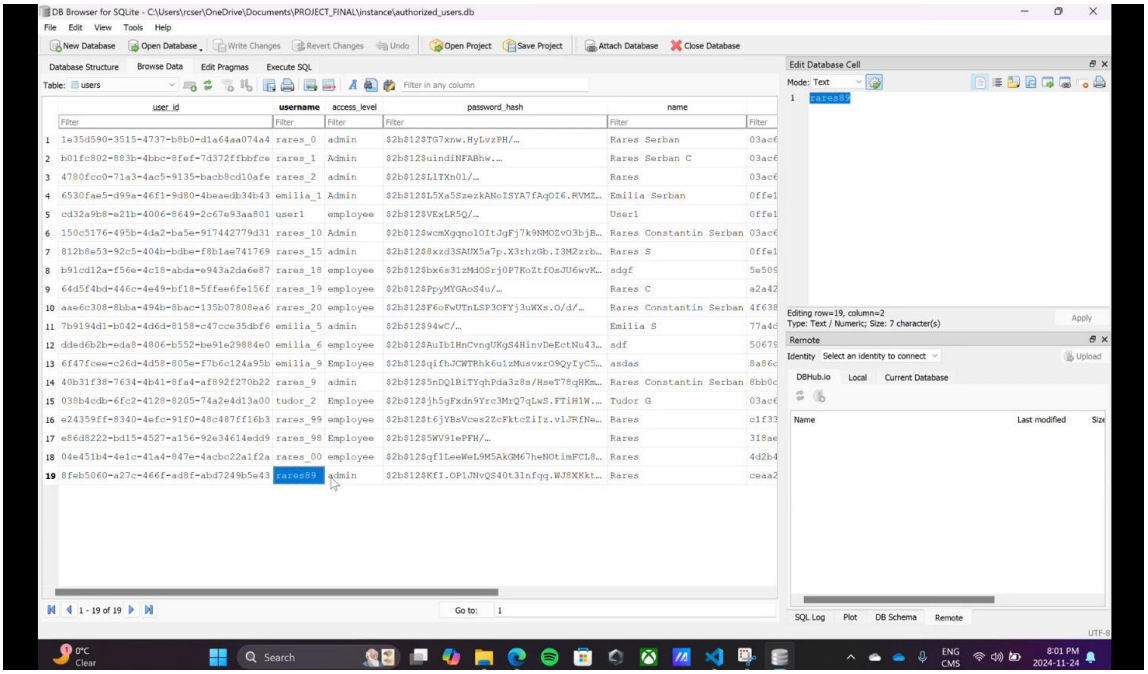


Figure 6. User Data Base

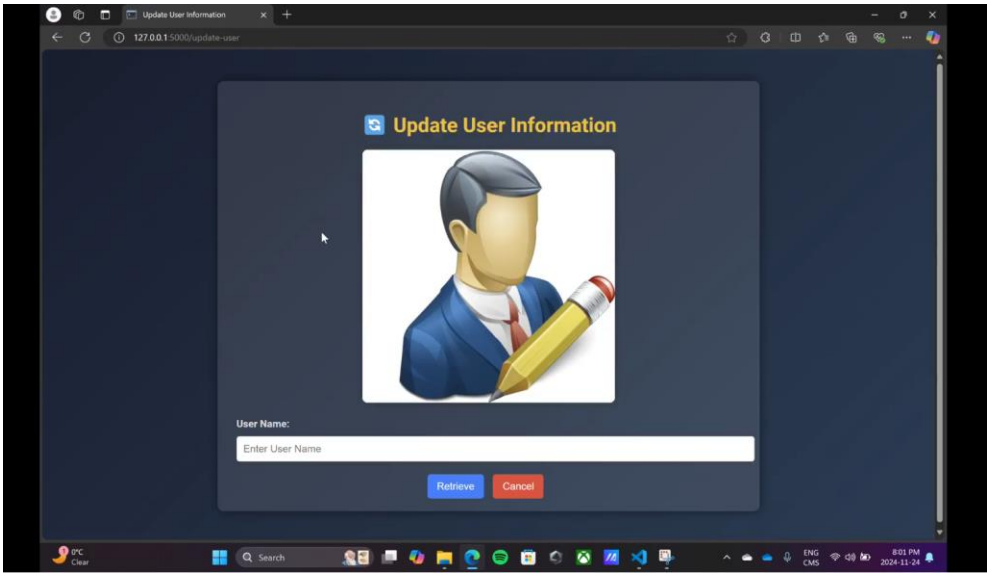


Figure 7. Admin Updating the User

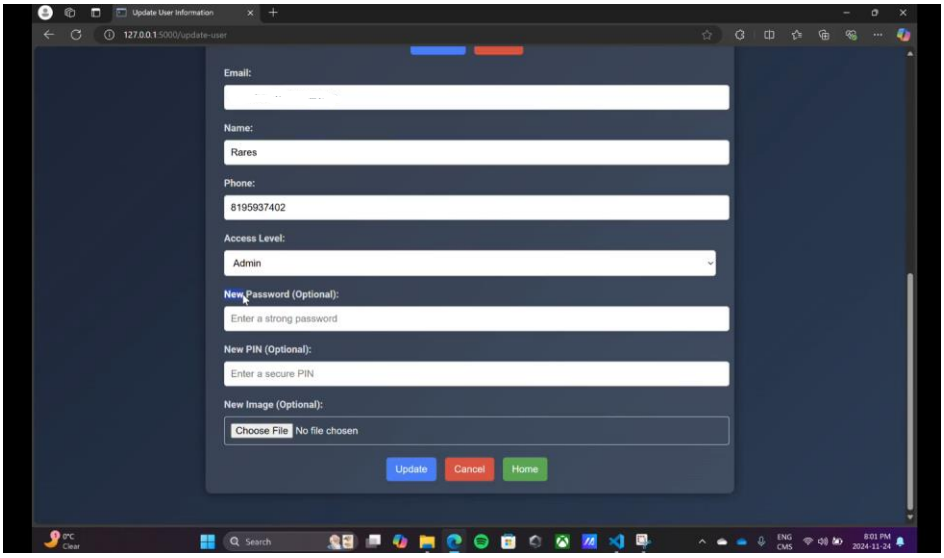


Figure 8. Admin Updating User Part 2

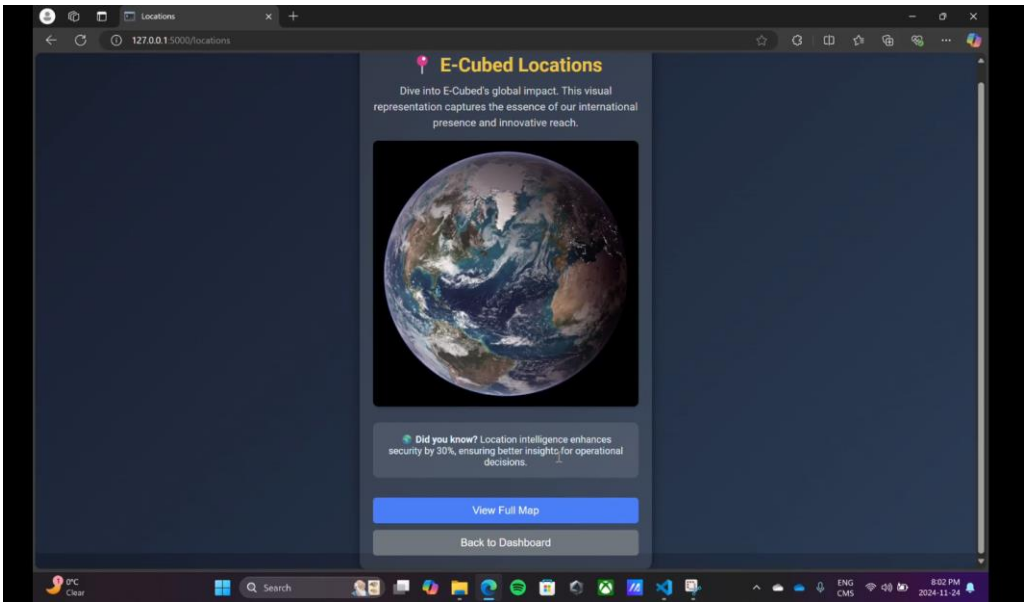


Figure 9. Locations

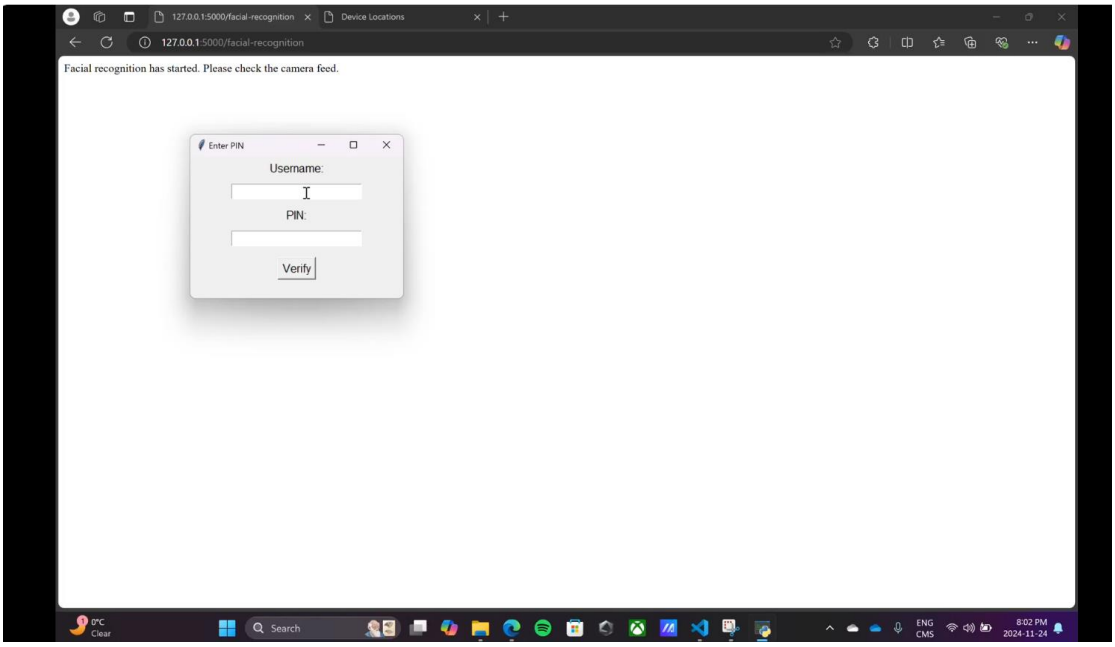


Figure 10. Facial Recognition

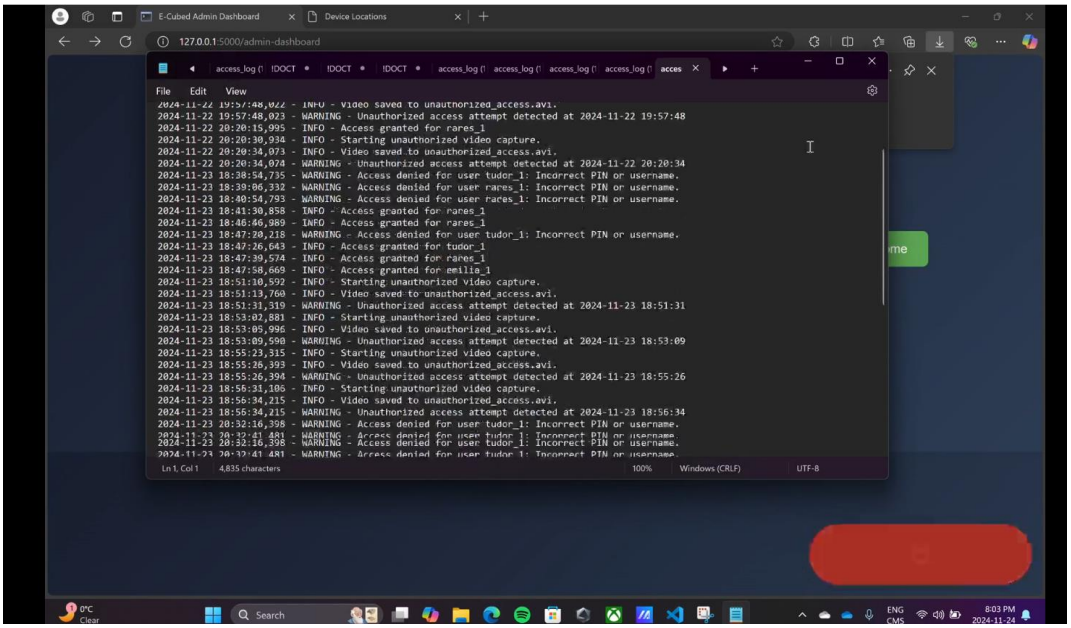


Figure 11. Access Attempt Logs

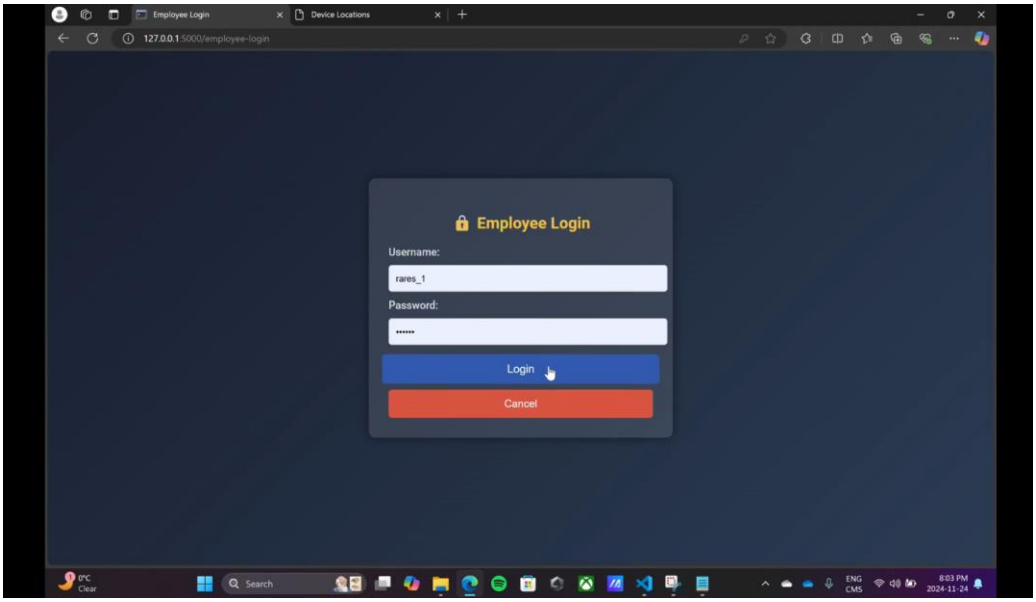


Figure 12. Employee Login

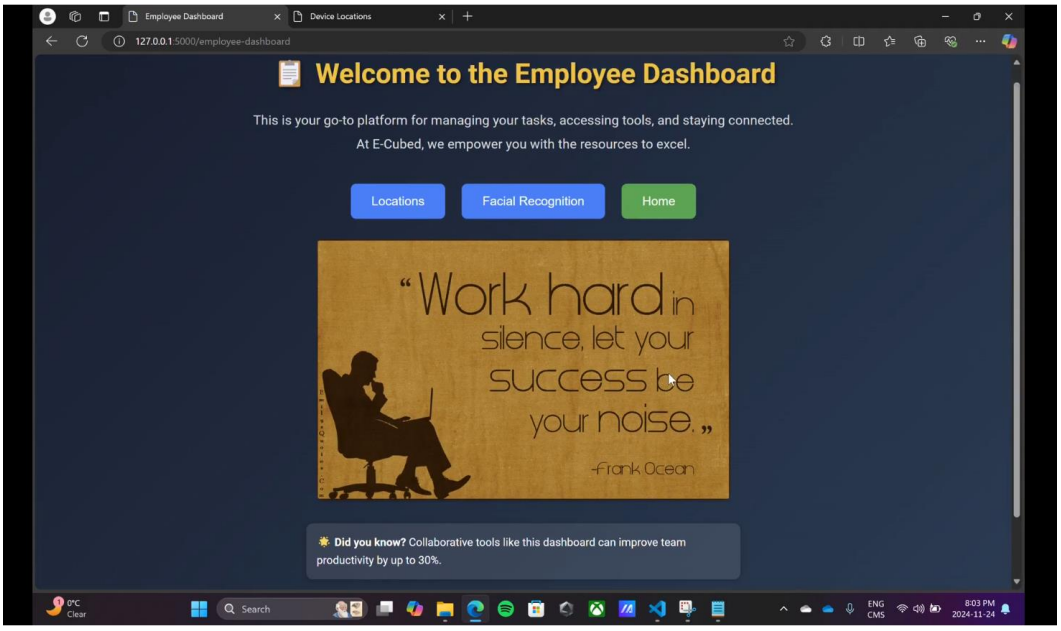


Figure 13. Employee Dashboard

Note: Location and Face ID work the same as administration.

3.3 Configuration Considerations

This system was created via the python coding language due to its accessibility and expansive libraries. Doing so lets us have multiple different systems that allow for the best security possible and preventing UAAEs. The inputs required are common pieces of information when setting up accounts like usernames and passwords. The outputs are created when they receive the inputs and are logged into a directory.

When creating the user interface, we considered both employees and administration. The administrator can add or change employee information. The Employee can only use the location and facial recognition part of our design. In addition, our user interface uses recognizable buttons like the “home” button to make it easier to navigate. E-cubed is mostly branded in blue so the website application has a lot of blue. We also use images to remind the user what page they are currently on.

3.4 User Access Considerations

The user that this prototype is designed for is administrative staff for enterprises as the primary user. The secondary user are the employees. The UI is simple with clearly defined sections to prevent any confusion for the administrative staff and employees. The administration can access all parts of the system since they are able to log in to both the employee and administration sections. Employees are only able to log into the employee section. The respective groups are prompted to log in upon clicking on the button.

3.5 Accessing/setting up the System

To run this application, you need to have three python files running and have two directories. One directory is for location tracking, where it stores the employees' device ID, latitude and longitude, as well as a timestamp. The location system needs this to operate. The second directory is to store all the employees' user IDs, usernames, access level, passwords, names, pins, phone numbers, emails, and images. This is needed for the facial recognition system, including the pin authentication and the update user and register user functions

User IDs will be automatically generated upon registration. During the same process, they can upload a clear photo (JPEG, JPG, PNG, GIF, BMP, TIFF, WEBP, HEIF/HEIC, RAW, SVG) file to the system in the correct area and then the facial recognition system will work. To update or change any information the administrator needs to select the option labeled “Update User”, retrieve current user information, and change the desired fields. There, they can change the employee's information when desired.

3.6 System Organization & Navigation

Starting from the home page of the UI there is a two-button display, one listed as ‘Admin’ and the other as ‘Employee’. Once selecting one of these options the user will be prompted to enter their username and password.

If on the administration page there are 6 buttons they can choose from; Register Employee, Update Employee, Locations, Facial Recognition, Logs and Home. Register employee links to the directories and the input information is saved automatically. Update employees work similarly and send information to the same directories. The Locations option connects to the location directory. Once the Shortcut has successfully completed, the phone’s information will be sent to the main program. After the user reloads the location page, the device location will appear on a map.

The register employee tab is where the administration can add employee accounts. This page will ask for the following information:

- Username
- Access Level
- Full Name
- Pin
- Phone Number
- Email
- Face Image

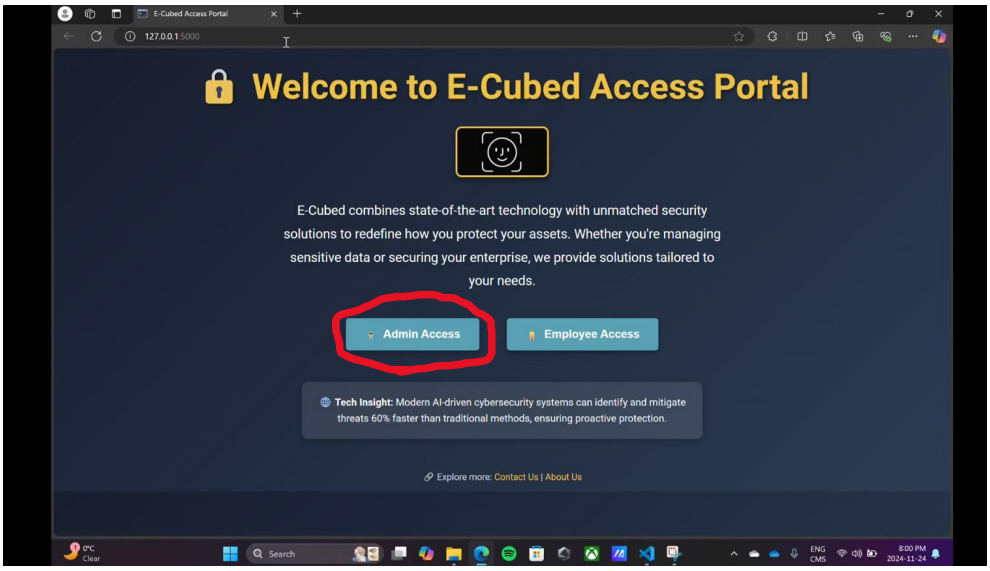
The username must be unique for each employee. The face image section is where they can attach a file via the “choose file” button. The image must be a front-facing image of an employee like a passport photo. See above section for specific file formats

The update user tab is where they can make modifications to existing employees. It first requires you to enter the employee's username. Then you click on ‘retrieve’. Once entered it shows you the same sections found in the register employee tab.

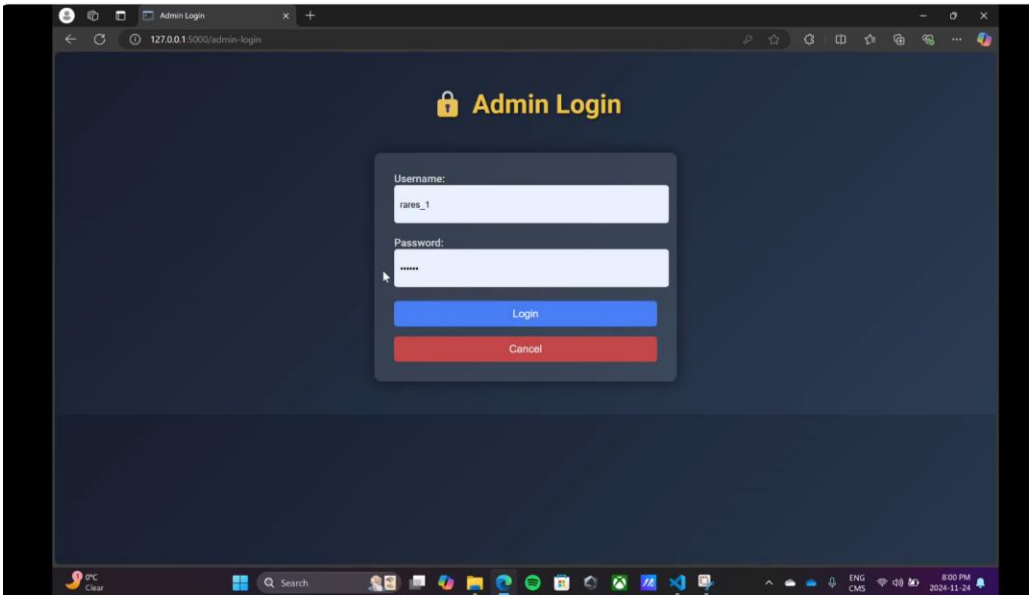
The facial recognition tab works in conjunction with the python file and camera system. Upon clicking the tab, it simulates a door pin by prompting the user in a pop-up window to enter their username and password. The camera begins to operate and carries out the appropriate result.

How to register an employee

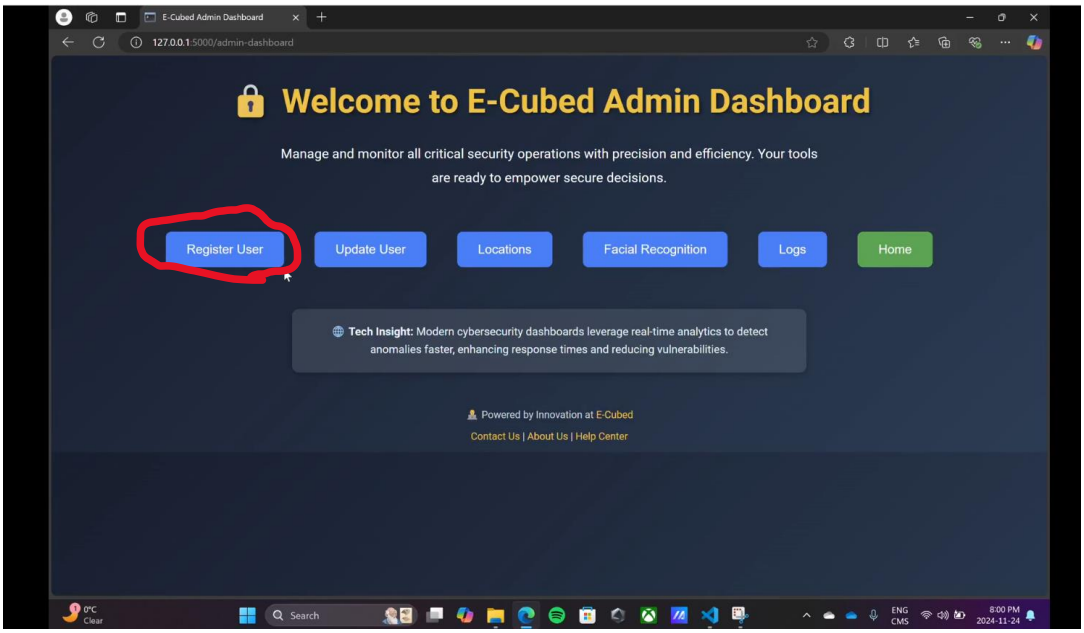
1. Click ‘Admin Access’



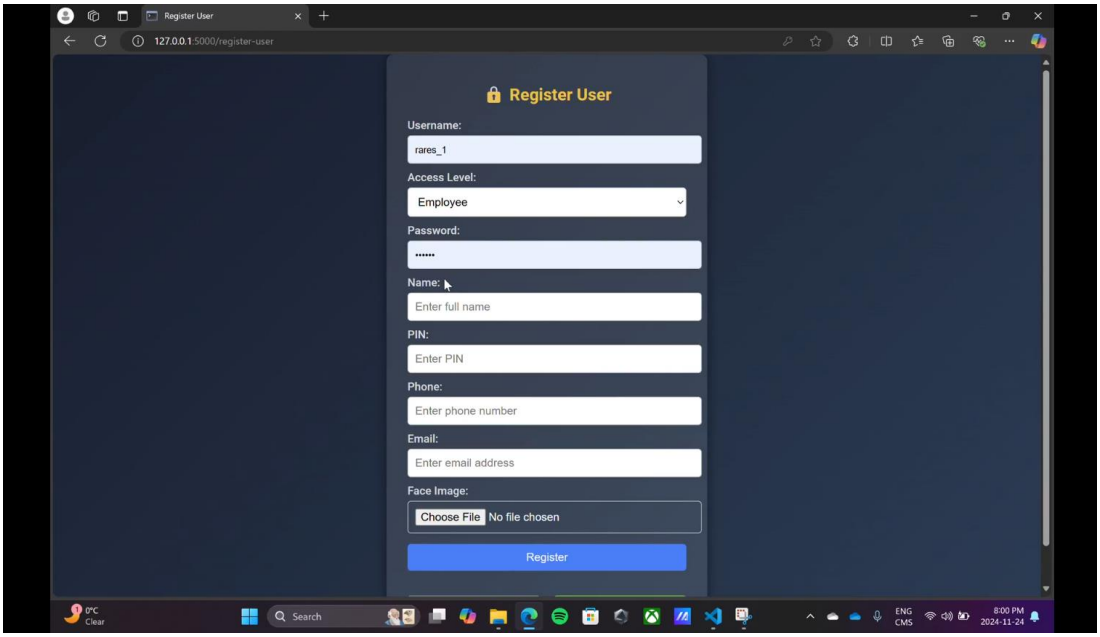
2. Log into Admin



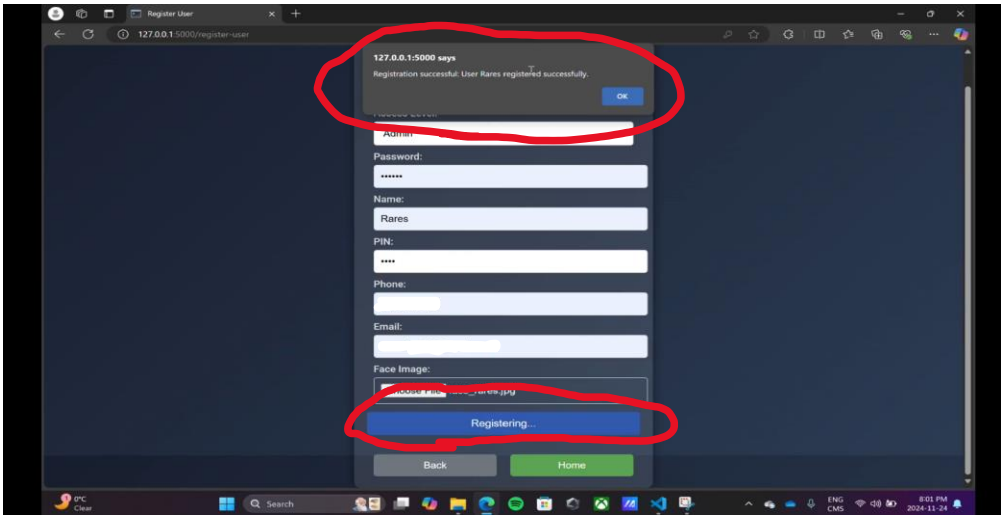
3. Click ‘Register User’



4. Enter all the information for the new user

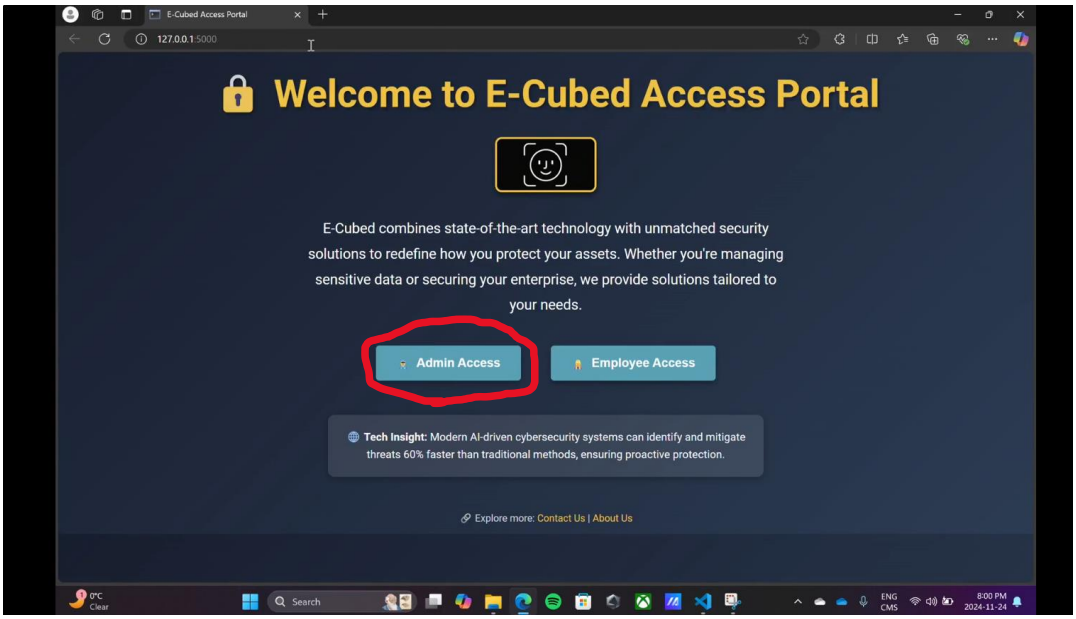


5. Click register and wait for confirmation pop up

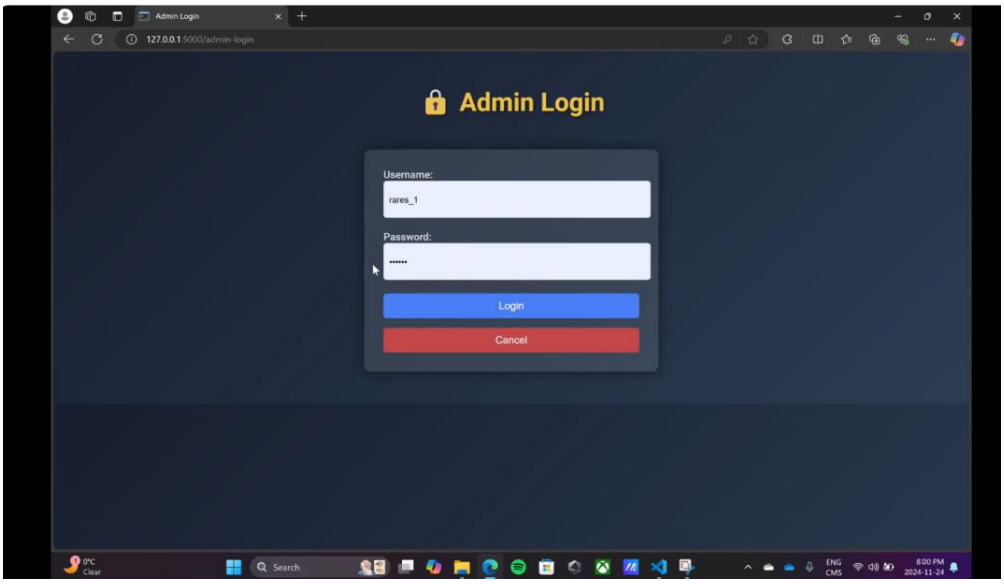


How to update user information

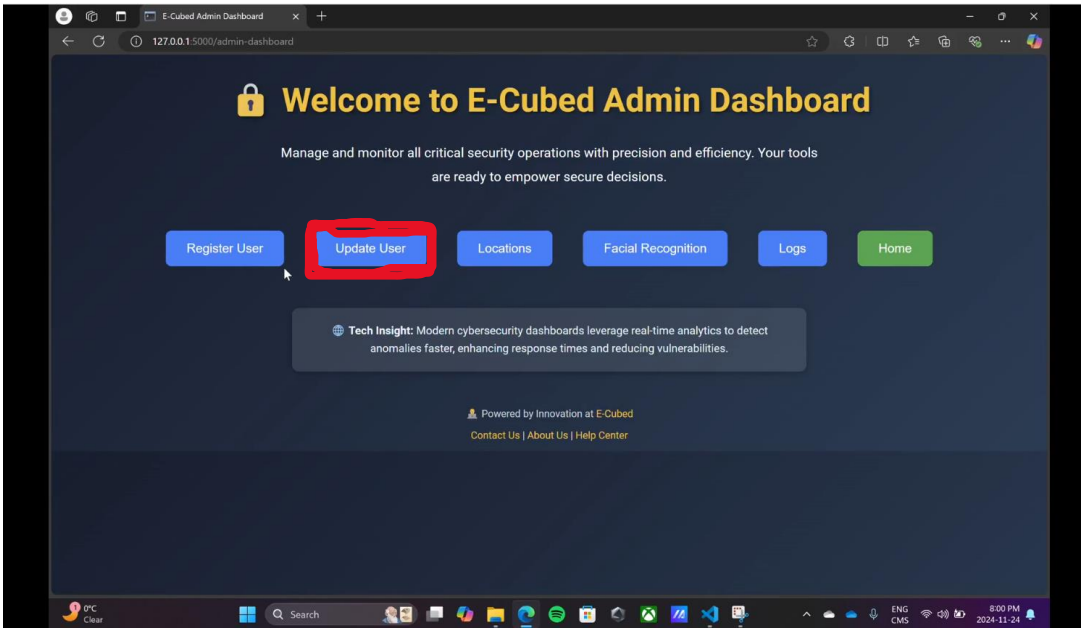
1. Click 'Admin Access' to log in



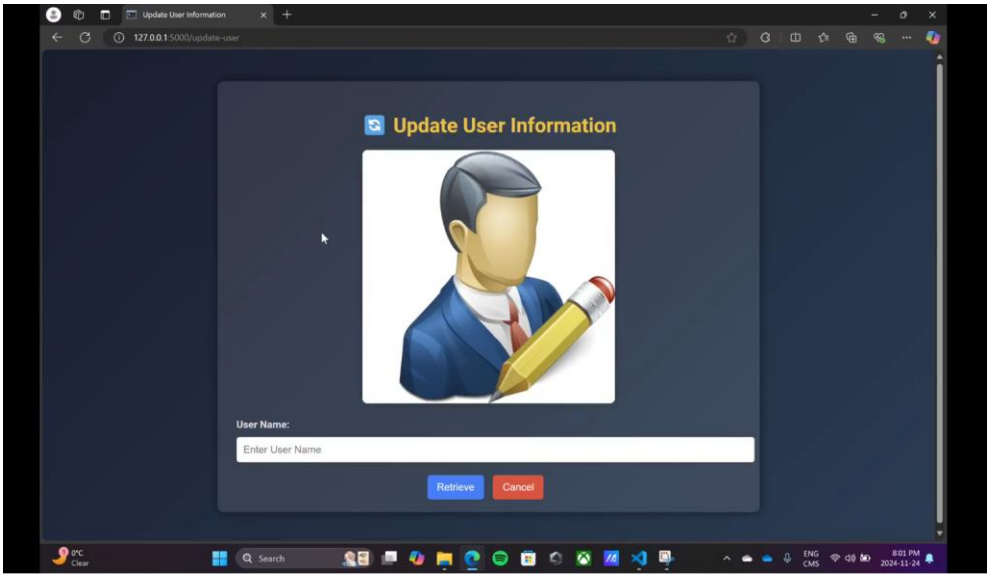
2. **Log into Admin**



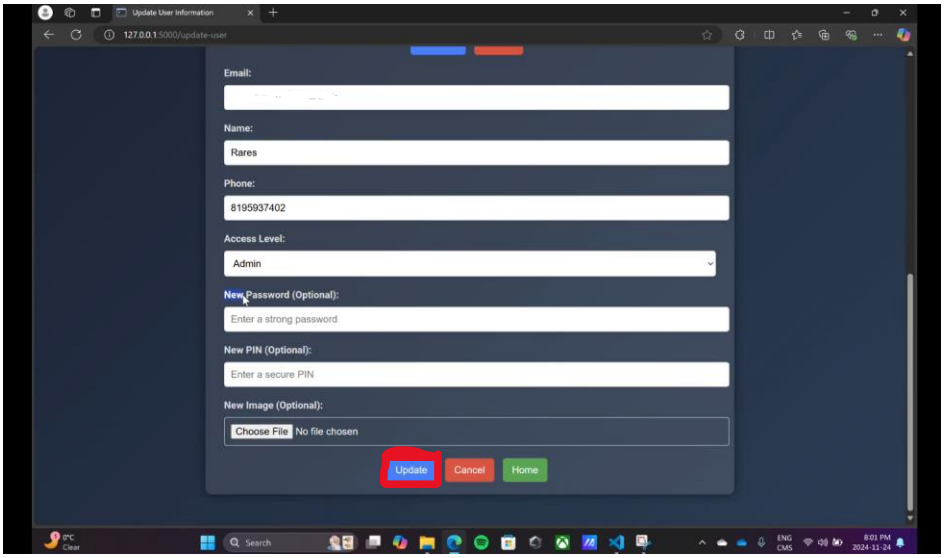
3. **Click 'Update User'**



4. Enter username

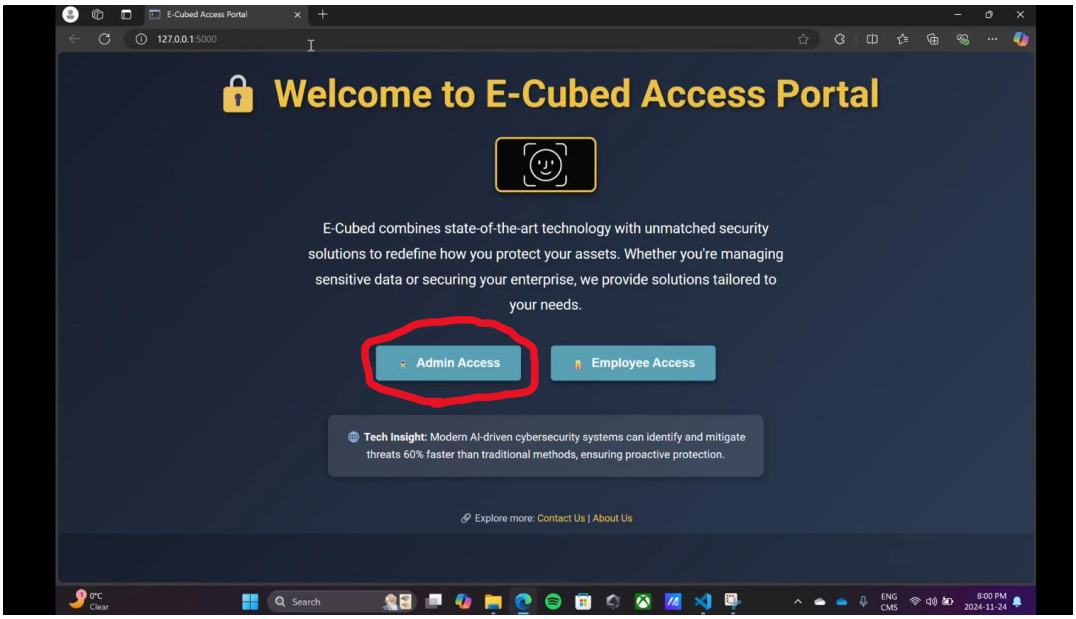


5. Change the information and click 'Update'

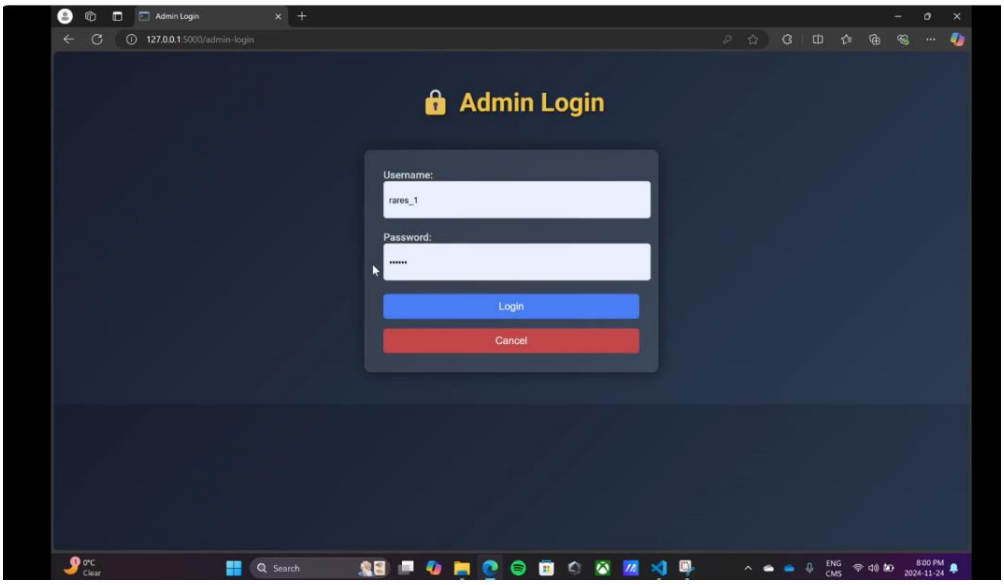


How to check employee location

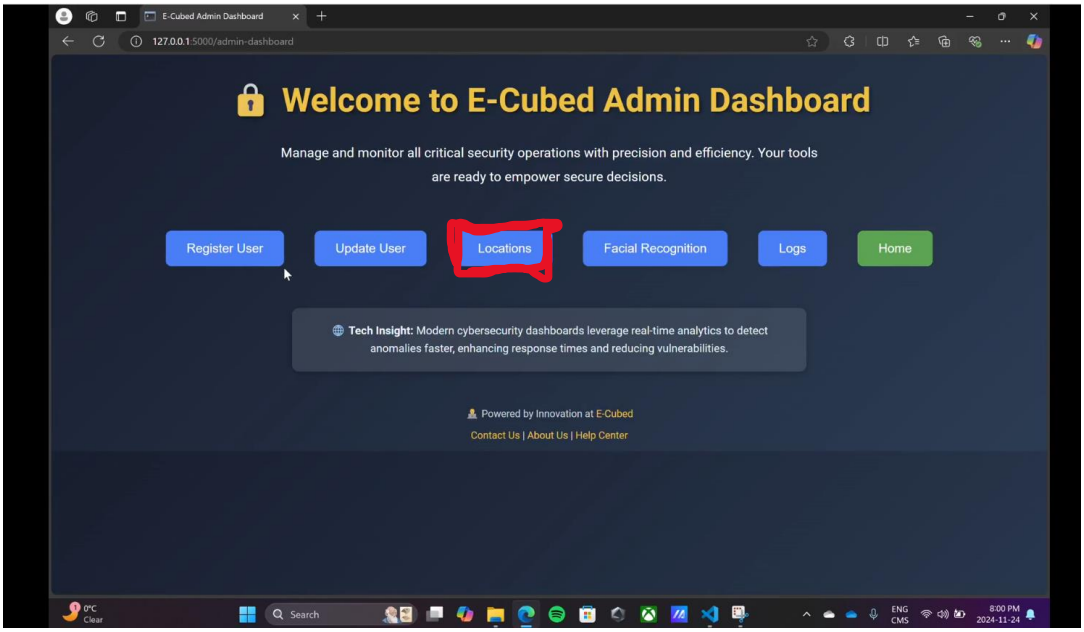
1. Click ‘Admin Access’ to log in



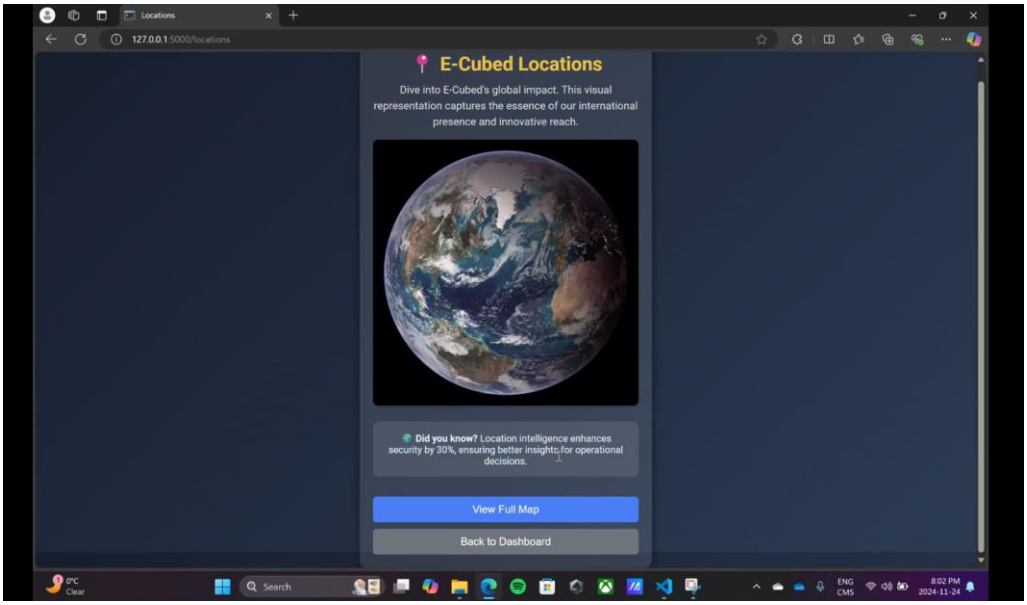
2. **Log into Admin**



3. **Click 'Locations'**

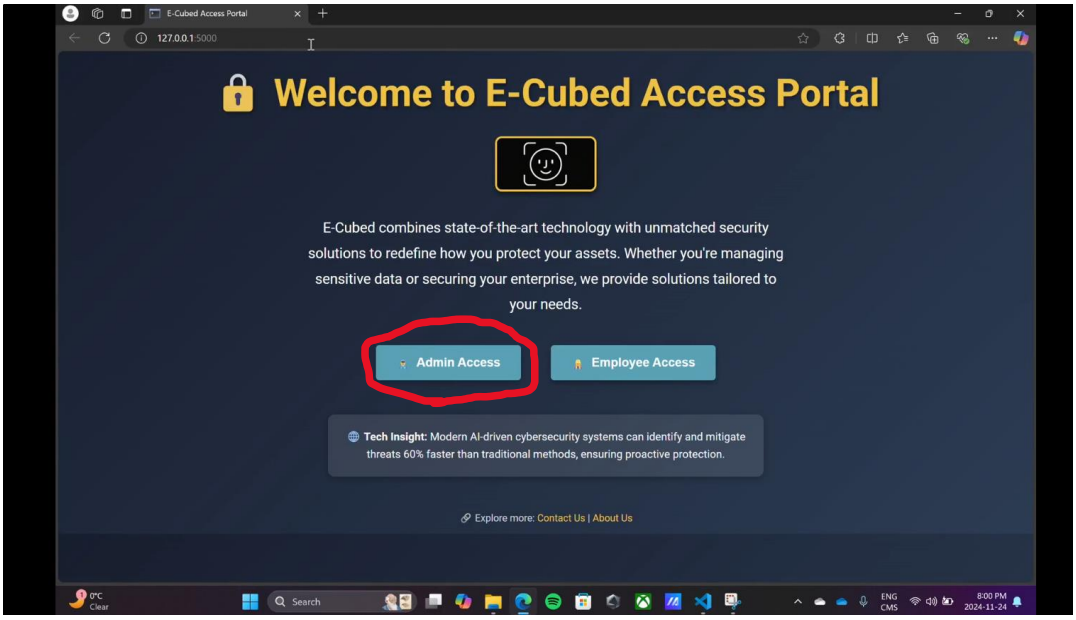


4. Click ‘View Full Map’

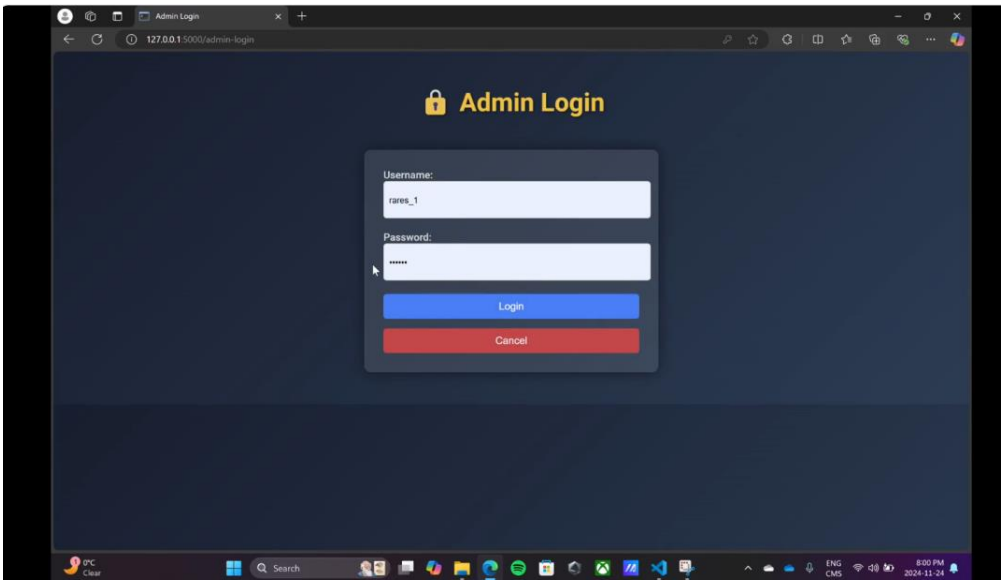


How to Test facial recognition

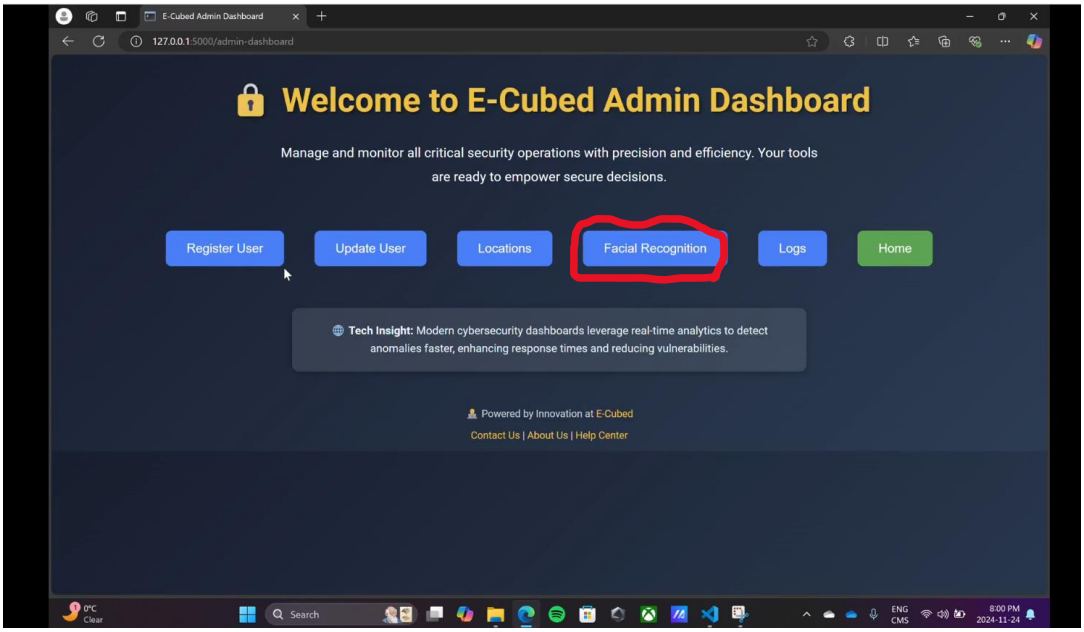
1. Click ‘Admin Access’ to log in



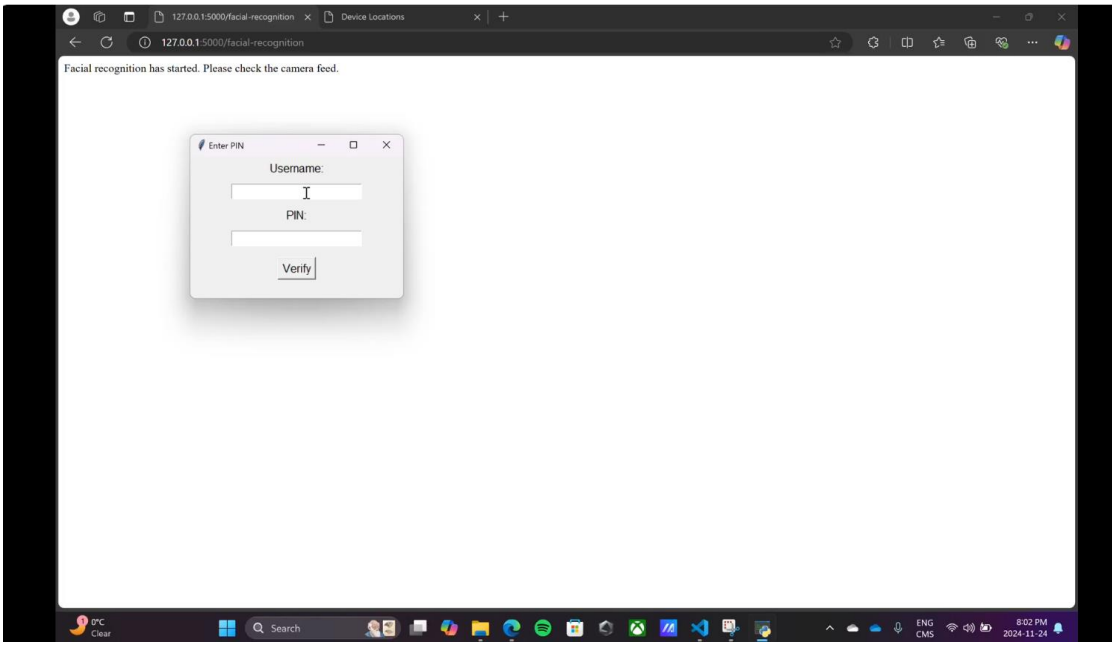
2. **Log into Admin**



3. **Click 'Facial Recognition'**



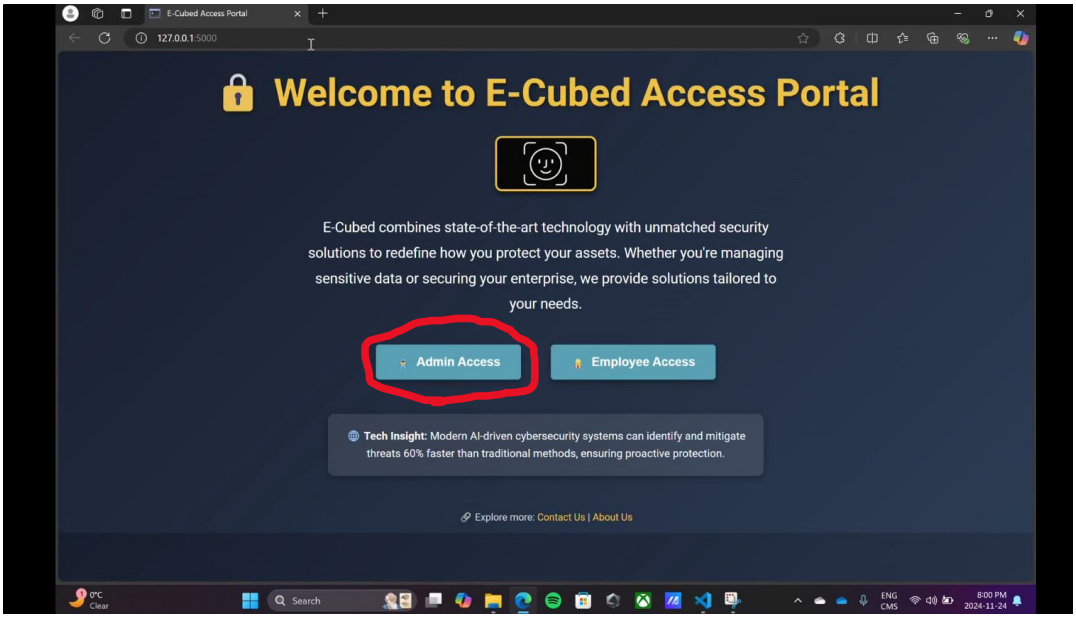
4. Enter credentials



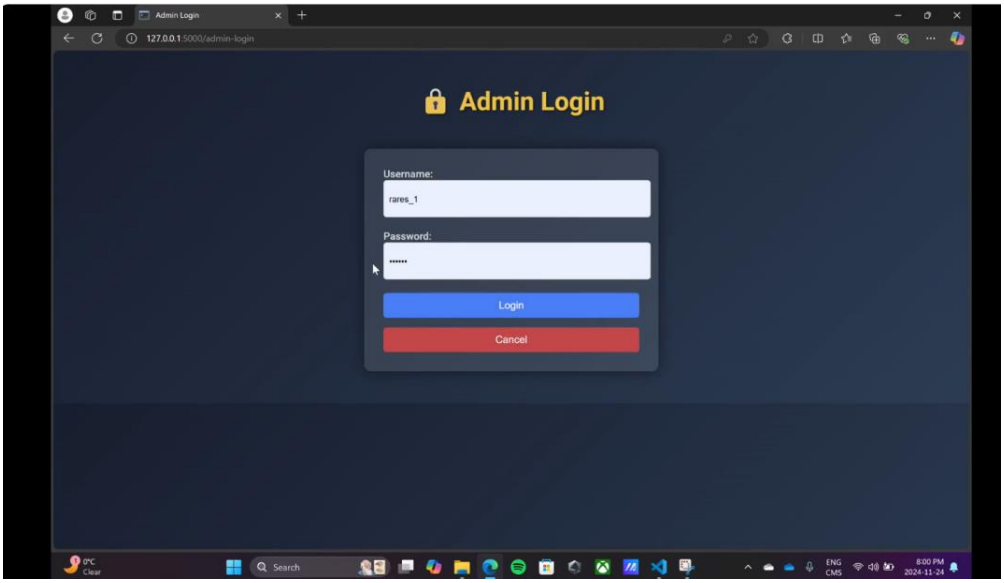
5. If the system recognizes the face and it belongs to the credentials then you will receive access granted pop-up. If the system does not recognize the face, then you will receive a notification with a video of the person who attempted to log in.

How to check employee logs

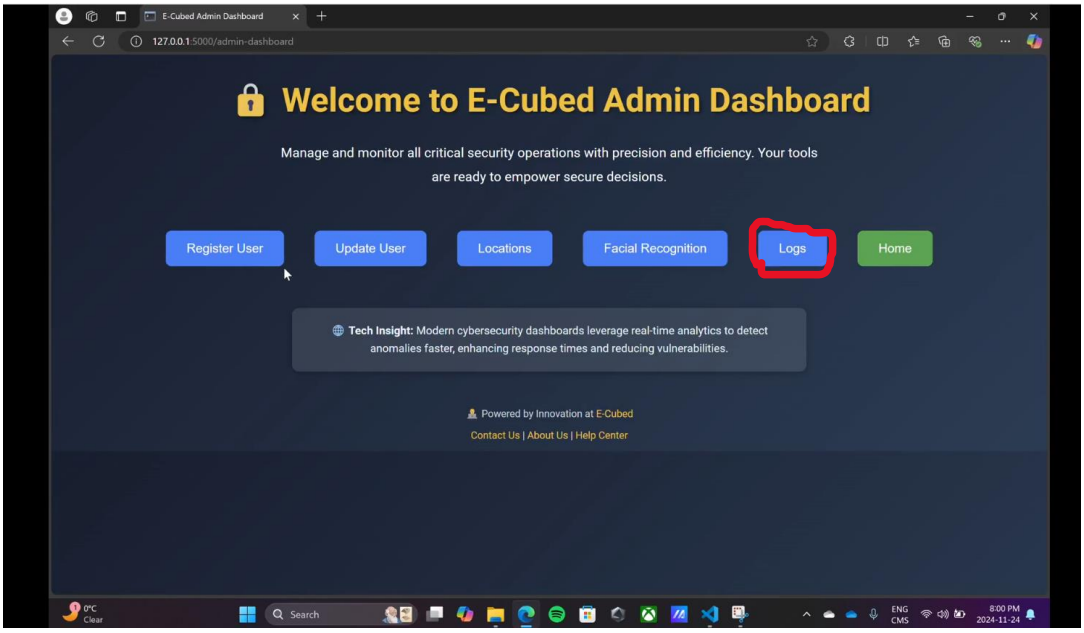
1. Click ‘Admin Access’



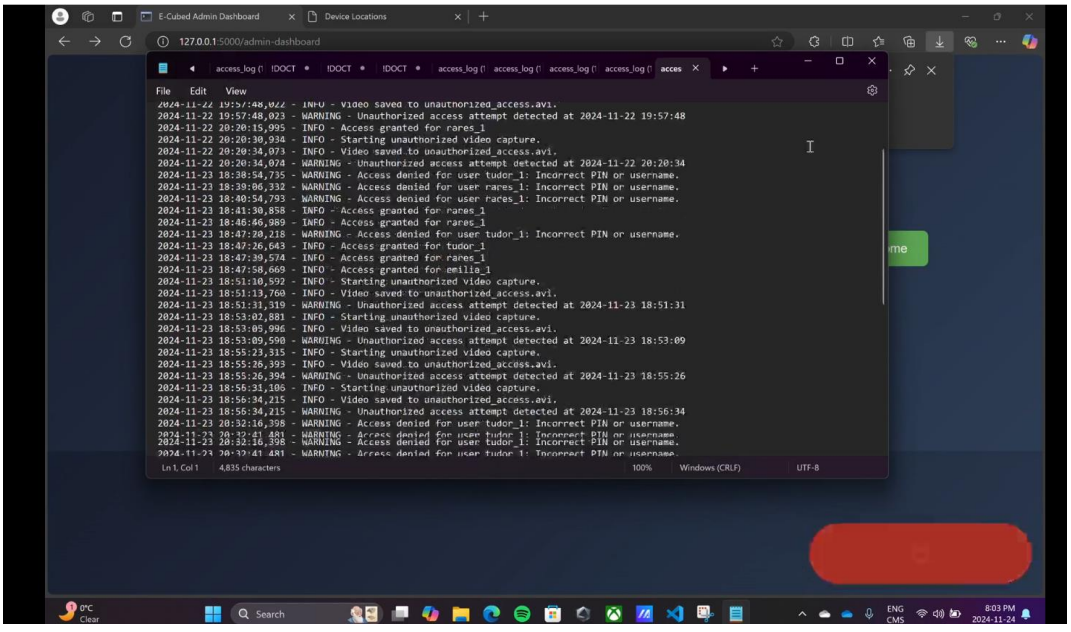
2. Log into Admin



3. Click ‘Logs’

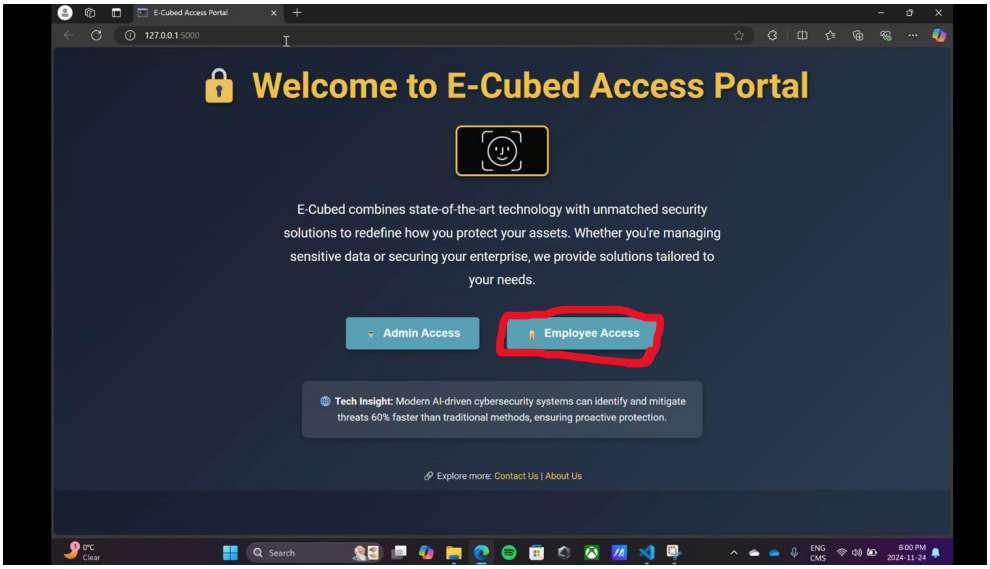


3. Open the Downloaded Document with the Logs.

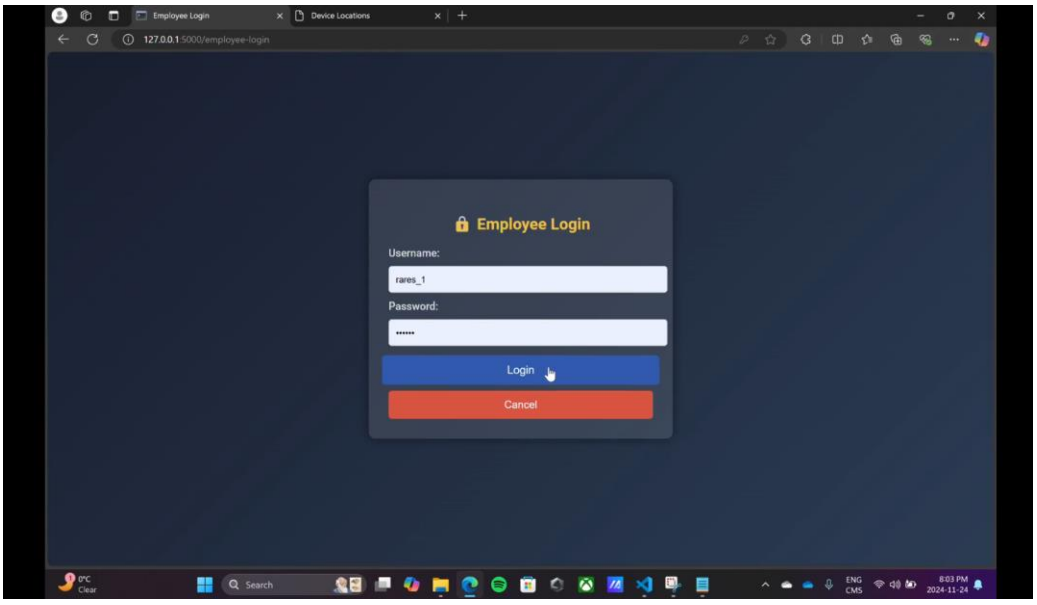


Employee access

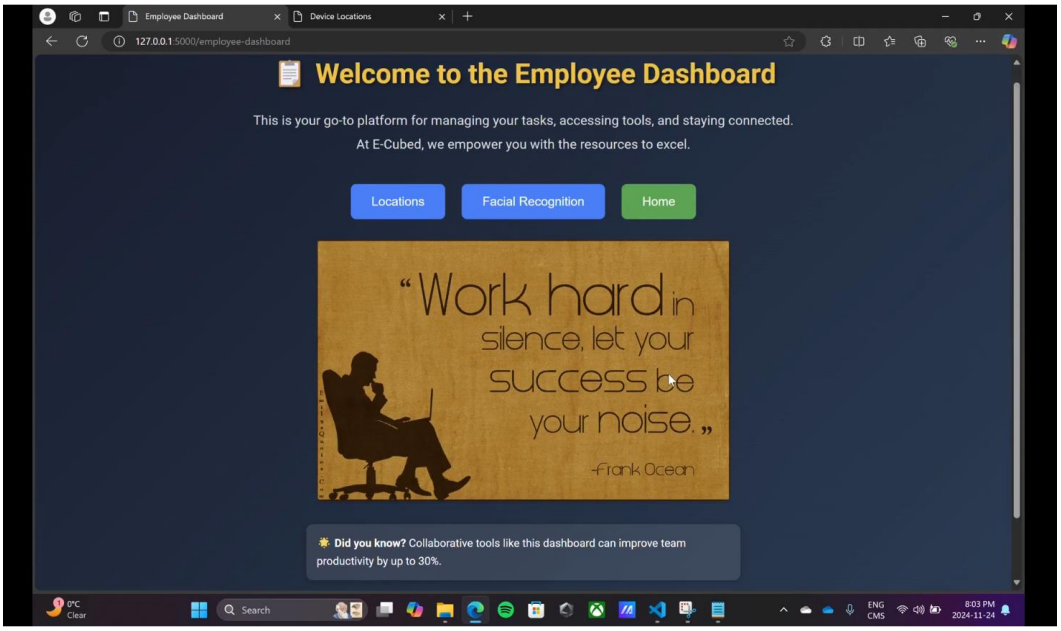
1. Click ‘Employee Access’



2. Log into Employee Login with Credentials



3. You can choose between facial recognition or location which works the same way as administration. If you want further detail look at the administration location and administration facial recognition.



3.7 Exiting the System

To stop the system from operating simply close the html pages by clicking the X button in the top left corner of the screen. Use CTRL+C in the terminals to stop running the python files.

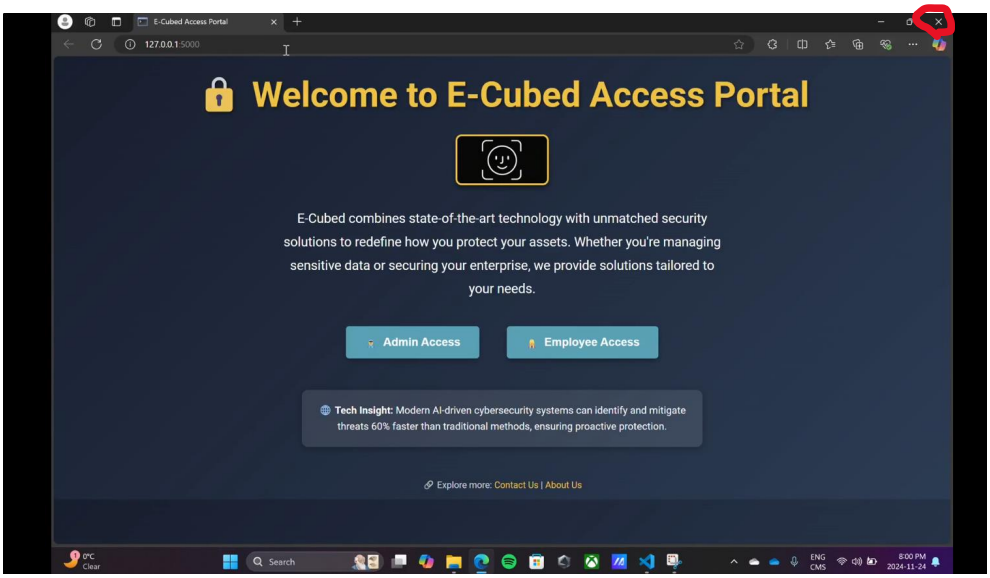


Figure 14. Highlighting the exit button of the system

4 Using the System

We have four main features that set our net application apart. The systems are facial recognition, location, logging, and notifications.

Facial recognition gets activated when the facial recognition button gets pressed and correct login information is entered. It is expected that facial recognition will recognize the user if the face is the same as the logged user. This system uses OpenCV which is great for recognizing one face at a time but may have trouble if there is more than one person. If the program recognizes the face, then there will be a pop up “access granted”. If the program doesn’t recognize the face, then it should take a video and send a notification. The user may encounter problems if they have face coverings preventing the camera from seeing them face which can cause a false denied access.

The Location subsystem is expected to work whenever the location button is pressed. The expected behavior of the location system is the location is updated every time the location is requested, and it is accurate up to 4m.

The logging subsystem is expected to register every time someone attempts to log with the facial recognition subsystem. It is also expected to show every access attempt when the log is opened by administration in a list format. If there is a glitch in the code where a log does not get registered there will be no notification or warning.

The notification subsystem is expected to send an email notification with a video attachment of a person who is trying to log in with wrong credentials. The video should be a couple seconds long with a fair video quality. If there is a failure in sending the email, there is also no warning. In addition, we are using an API called Twilio to send notifications, but it can only send a set amount of notifications for free. Therefore, if there are too many notifications sent the notification system will stop working.

The following sub-sections provide detailed step-by-step instructions on how to use the various functions or features of the E Cubed Security System.

4.1 Location Tracking

Location tracking is established by a wireless connection between our iPhones and the Flask Server. This method allows for high accuracy regardless of any external factors (e.g. cell towers). The following steps show how to set the system up:

1. Open the Shortcuts application on your iPhone.
2. On the top right corner, press the "+" button to create a new Shortcut.
3. In the Search Actions toolbar, type "Get current location" and click on it.
4. Click again on "Search Actions" and look for "Get Details of Locations".
5. From the newly added rectangle box, click on the "Detail" case and under "Variables", select "Latitude".
6. Repeat step 5 but select "Longitude" instead of "Latitude". If the second box contains anything other than "current location", press on it and change it accordingly.
7. Click again on "Search Actions" and find "Get Device Details". Make sure that the action gets the "Device Name" (your device identifier).
8. Click again on "Search Actions" and find "Get Contents of URL".
9. Clear all variables in the box and enter *http://<your IP address>/<port>/update_location.* If you don't know your local IP address, open your command prompt and type "ipconfig". The information on the left of the IPv4 field is your local IP address. Typically, your port is 5000.
10. Next to the entered information, click on the little arrow. Change the Method to "POST". Make sure the Request Body is "JSON".
11. Click on "Add new Field". The first two fields are "Number". Istead of "Key", type "lat" for the first one and "lon" for the second. Instead of the inscribed number, select the blue "Latitude" icon for the first one and "Longitude" for the second. For the third, select "Text", name it "device_id" and select the blue "Device Name" icon.
12. Optional: Under "Search Actions", look for "Show Notification", and write the notification you wish to see when the Shortcut is successfully executed.

Now you must set up your Flask server. N.B. This procedure includes two categories of **users (admin and employee):**

1. Install SQLite database on your desktop.
2. Create two separate Python files. You can name the first "admin_locations.py" and the second one "employee_location.py".
3. On "admin_locations", you can copy the code you find in the annex. With this, you will create an SQLite database with 5 fields: "id", "device_id", "lat", "lon" and "timestamp". After running the code and accessing the IP address given to you in the terminal you should see a page saying, "Waiting for location data".
4. Run the Shortcut on your iPhone and reload your page. You should now see an interpreted map (by Folium) and a pin indicating your current location.

5. If you run multiple Shortcuts all directed to the same IP address, you should see all the locations of all the devices.
6. On “employee_location”, you can copy the second code you find in the annex. This code doesn’t involve the database.
7. Execute step 4 again. You should now see your location on the map.

Additional information:

- To automate the Shortcut process, you can set up an automation action from the automation tab in your application. For example, you can say that when you get a message from “_person_” execute the shortcut. This way, an admin can get the location of all the employees without requesting their permission or any other action from their side.
- To avoid entering the IP address in the shortcut every time you change Network connection, you can create a tunnel for your specific port. This way, you can send the location from anywhere if you have some kind of Network connection. If you want to do that, follow these steps:
    1) Install “ngrok”
    2) Go to you command prompt and type “ngrok http <your port number>”
    3) If you’re on the free trial version, don’t close the command prompt and note that you can only create a tunnel for one port at a time.
    4) Next to the “Forwarding” field, you should see something resembling this: https://a7c5-137-122-64-245.ngrok-free.app
    5) Type this address instead of the IP address you currently have in the Shortcut, while keeping the route (“update_location”). It should look like this: https://a7c5-137-122-64-245.ngrok-free.app/update_location
    6) Now you should be able to send your location from anywhere you currently are, if you’re connected to the internet.
- If you want to run these codes together in parallel with your main application, at the same time, you must run the files on different ports. For example, run the main application on port 5000, employee_location on port 5001 and admin_locations on 5002. Don’t forget to change the information in the Shortcut as well if you need (if you don’t use the tunnel)!

4.2 Facial Recognition

This system does not require set up from the administration end, other than registration of employees. Which has been discussed in the registering a user section.

4.3 Notification

Administration is required to set up a Twilio account. Twilio can be found here: Twilio. Once they have received their authentication token, account SID and their Twilio phone number, they will need to put these in the config file in the labeled sections. Detailed images of the code can be found in later sections of the manual.

4.4 UI/Web Application

1. Download the Application: Clone or download the repository containing the application files.
2. Install Dependencies: Run the following command: “pip install -r requirements.txt”
3. Set Up the Database: Initialize the database by running: “python main.py”
4. Configure Environment Variables:
   - Create a ‘.env’ file to store sensitive credentials (API tokens, email credentials, etc.).
5. Start the Application:
   - Run the backend servers in separate terminals:
     python main.py
     python admin_locations.py
     python employee_location.py

Once downloaded, open a browser and navigate to http://127.0.0.1:5000, so you can begin using the product.

5 Troubleshooting & Support

If the facial recognition system won’t recognize a face and keeps sending notifications.

1. Make sure there are no face coverings, and the video camera has good resolution.

2. Make sure the user has an updated user picture

To update a user picture, look up “how to update a user information” in the user manual.

If the location does not work.

1. Make sure that the devise is connected to a local network

2. Make sure that the program is properly installed on the phone

If you can’t log into administration

1. Make sure that you properly copy the administration log in provided to you by E-Cubed.

2. Make sure to email us if you forget administration login and password.

If you can’t log as an employee

1. Contact your administration, they have access to your username and password, they can confirm your credentials.

2. If an employee still can't log in, please contact us.

5.1 Error Messages or Behaviors

As of our current knowledge there are no errors in this system. The only failure possible is a complete system failure or glitches without any Error Messages.

5.2 Special Considerations

If any issues arise it is necessary to check the directory of information. To ensure that the prototype works correctly the information entered in the directory must be accurate. There cannot be any extra space after a username or password is entered. The phone number must be entered correctly with no extra digits. The photo in the directory must be a clear front facing image of the employee. If these registration steps are not done correctly the system might fail to process the information and unlock the doors.

5.3 Maintenance

To avoid any issues in this prototype it is necessary to ensure that there are no duplicate accounts for one employee in the directories. Additionally, if any employee leaves the company they would need to be manually removed from the directory. If either of these two cases occur there can be entrance to restricted areas and the continued ability of the administration to track these employees.

5.4 Support

When reporting a problem with the system it would be ideal to include a photo of the error message that appears on screen. A short description of the issue would be beneficial so we can provide the best care. If there is a technical issue with our web application, please contact our head of programing Rares at rares@ecubed.com. If there is constructive feedback or have questions about generally navigating the web application, then please contact our head of client relations louis-Philippe by email at Louisphilippe@ecubed.com. If there's any questions about documentation, please contact our Team Leader Emma at Emma@ecubed.com. If there are any security issues, please contact your companies' human recourses and security company for the proper procedures.

6 Product Documentation

In this section we will be covering how our final prototype was built as well as considerations we took when making the final prototype. We will also cover the different options we had at our disposal, the BOM, and the different tests we put our prototype through and the results we received. In each section we have detailed the code we used to develop the product as whole. If the code is copied with the correct libraires you will be able to build the same system.

6.1 main.py

```
import os
import subprocess
import face_recognition
from flask import Flask, request, jsonify, render_template, redirect, send_file,
url_for, session
import sqlite3
import bcrypt
import uuid
from werkzeug.utils import secure_filename

from facial_recognition import hash_pin

app = Flask(__name__)
app.secret_key = 'supersecretkey'

ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}

UPLOAD_FOLDER = 'uploads'
if not os.path.exists(UPLOAD_FOLDER):
    os.makedirs(UPLOAD_FOLDER)

# Database initialization
def initialize_database():
    conn = sqlite3.connect("instance/authorized_users.db")
    c = conn.cursor()
    c.execute('''CREATE TABLE IF NOT EXISTS users
                 (user_id TEXT PRIMARY KEY, username TEXT UNIQUE, access_level
TEXT, password_hash TEXT,
                 name TEXT, pin TEXT, phone TEXT, email TEXT, image BLOB)''')
    conn.commit()
    conn.close()

initialize_database()


def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in
ALLOWED_EXTENSIONS

# Route to serve the HTML file
@app.route('/')
def serve_html():
    return render_template('homepage.html')
```

```
@app.route('/admin-login', methods=['GET', 'POST'])
def admin_login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = sqlite3.connect("instance/authorized_users.db")
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username = ? AND LOWER(access_level)
= 'admin'", (username,))
        user = c.fetchone()
        conn.close()

        if user and bcrypt.checkpw(password.encode('utf-8'), user[3]): # Assuming
the password hash is in the 4th column
            session['username'] = username
            session['access_level'] = 'admin'
            return redirect(url_for('admin_dashboard'))

        return "Invalid credentials", 401

    return render_template('admin_login.html')

@app.route('/employee-login', methods=['GET', 'POST'])
def employee_login():
    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']

        conn = sqlite3.connect("instance/authorized_users.db")
        c = conn.cursor()
        c.execute("SELECT * FROM users WHERE username = ?", (username,))
        user = c.fetchone()
        conn.close()

        if user and bcrypt.checkpw(password.encode('utf-8'), user[3]):  # Assuming
the password hash is in the 4th column
            session['username'] = username
            session['access_level'] = 'employee'
            return redirect(url_for('employee_dashboard'))

        return "Invalid credentials", 401

    return render_template('employee_login.html')

@app.route('/admin-dashboard')
```

```
def admin_dashboard():
    if 'username' not in session or session['access_level'] != 'admin':
        return redirect(url_for('admin_login'))
    return render_template('admin_dashboard.html')

@app.route('/employee-dashboard')
def employee_dashboard():
    if 'username' not in session or session['access_level'] != 'employee':
        return redirect(url_for('employee_login'))
    return render_template('employee_dashboard.html')

@app.route('/register-user', methods=['GET', 'POST'])
def register_user():
    if 'username' not in session or session['access_level'] != 'admin':
        return redirect(url_for('admin_login'))

    if request.method == 'POST':
        try:
            # User registration logic (getting form data and adding to the
database)
            user_id = str(uuid.uuid4())
            username = request.form['username']
            access_level = request.form['access_level']
            password = request.form['password']
            password_hash = bcrypt.hashpw(password.encode('utf-8'),
bcrypt.gensalt())
            name = request.form['name']
            pin = request.form['pin']
            phone = request.form['phone']
            email = request.form['email']
            image = request.files['image']

            if not all([username, name, access_level, password, phone, email, pin,
image]):
                return jsonify({'success': False, 'message': 'Required fields are
missing'}), 400

            # Save the image
            image_path = os.path.join(UPLOAD_FOLDER, image.filename)
            image.save(image_path)

            # Add user to the database
            response = add_user(user_id, username, access_level, password_hash,
name, pin, phone, email, image_path)

            return jsonify(response)  # Return JSON response
```

```
        except Exception as e:
            return jsonify({"success": False, "message": f"Error: {str(e)}"})

    return render_template('register_user.html')

@app.route('/update-user')
def update_user():
    if 'username' not in session or session['access_level'] != 'admin':
        return redirect(url_for('admin_login'))
    return render_template('update_user.html')

@app.route('/update_user/<string:username>', methods=['POST'])
def update_user_post(username):
    data = request.form
    name = data.get('name')
    phone = data.get('phone')
    email = data.get('email')
    access_level = data.get('access_level')
    password = data.get('password')  # Optional
    pin = data.get('pin')            # Optional
    image = request.files.get('image')  # Optional

    if not all([name, phone, email, access_level]):
        return jsonify({'success': False, 'message': 'Required fields are
missing'}), 400

    try:
        conn = sqlite3.connect("instance/authorized_users.db")
        cursor = conn.cursor()

        query = """UPDATE users SET name = ?, phone = ?, email = ?, access_level =
?"""
        params = [name, phone, email, access_level]

        if password:
            hashed_pass = bcrypt.hashpw(password.encode('utf-8'),
bcrypt.gensalt())
            query += ", password_hash = ?"
            params.append(hashed_pass)

        if pin:
            query += ", pin = ?"
            params.append(hash_pin(pin))

        if image and allowed_file(image.filename):
            filename = secure_filename(image.filename)
            image_path = os.path.join(UPLOAD_FOLDER, filename)
```

```
            image.save(image_path)
            # Load and encode the face from the image
            image = face_recognition.load_image_file(image_path)
            face_encodings = face_recognition.face_encodings(image)
            if not face_encodings:
                return {"success": False, "message": "No face found in the image.
Please provide a clear image."}
            face_encoding = face_encodings[0]

            query += ", image = ?"
            params.append(face_encoding.tobytes())

        query += " WHERE username = ?"
        params.append(username)

        cursor.execute(query, tuple(params))
        conn.commit()

        if cursor.rowcount == 0:
            return jsonify({'success': False, 'message': 'User not found'}), 404

        return jsonify({'success': True, 'message': 'User updated successfully'})

    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

    finally:
        conn.close()

@app.route('/retrieve_user/<string:username>', methods=['GET'])
def retrieve_user(username):
    if not username:
        return jsonify({"success": False, "message": "Username is required"}), 400

    try:
        conn = sqlite3.connect("instance/authorized_users.db")
        c = conn.cursor()

        c.execute("""SELECT username, access_level, name, phone, email FROM users
WHERE username = ?""", (username,))
        user = c.fetchone()
        conn.close()

        if user:
            return jsonify({
                'success': True,
                'user': {
```

```
                    'username': user[0],
                    'access_level': user[1],
                    'name': user[2],
                    'phone': user[3],
                    'email': user[4],
                }
            })
        else:
            return jsonify({'success': False, 'message': 'User not found'}), 404

    except Exception as e:
        return jsonify({'success': False, 'message': str(e)}), 500

@app.route('/facial-recognition', methods=['GET'])
def facial_recognition():
    if 'username' not in session or session['access_level'] not in ['admin',
'employee']:
        return redirect(url_for('admin_login'))
    try:
        subprocess.Popen(['python', 'facial_recognition.py'])
        return "Facial recognition has started. Please check the camera feed.",
200
    except Exception as e:
        return str(e), 500

@app.route('/logs', methods=['GET'])
def view_logs():
    if 'username' not in session or session['access_level'] != 'admin':
        return redirect(url_for('admin_login'))
    log_file_path = os.path.join('logs', 'access_log.log')
    if os.path.exists(log_file_path):
        return send_file(log_file_path, as_attachment=False)
    else:
        return "Log file not found.", 404

@app.route('/locations', methods=['GET'])
def locations():
    if 'username' not in session:
        return redirect(url_for('admin_login'))

    # Determine the back_url and map_url based on user access level
    if session.get('access_level') == 'admin':
        back_url = url_for('admin_dashboard')
        map_url = 'http://127.0.0.1:5002/'  # Admin users point to port 5002
    elif session.get('access_level') == 'employee':
        back_url = url_for('employee_dashboard')
        map_url = 'http://127.0.0.1:5001/'  # Employee users point to port 5001
```

```
    else:
        return redirect(url_for('admin_login'))

    # Render the locations.html template and pass the URLs
    return render_template('locations.html', back_url=back_url, map_url=map_url)

    # Add user to the database
def add_user(user_id, username, access_level, password_hash, name, pin, phone,
email, image_path):
    try:
        conn = sqlite3.connect("instance/authorized_users.db")
        c = conn.cursor()

        # Load and encode the face from the image
        image = face_recognition.load_image_file(image_path)
        face_encodings = face_recognition.face_encodings(image)
        if not face_encodings:
            return {"success": False, "message": "No face found in the image.
Please provide a clear image."}
        face_encoding = face_encodings[0]

        # Hash the PIN
        hashed_pin = hash_pin(pin)

        # Insert user data into the database
        c.execute("INSERT INTO users (user_id, username, access_level,
password_hash, name, pin, phone, email, image) VALUES (?, ?, ?, ?, ?, ?, ?, ?,
?)",
                  (user_id, username, access_level, password_hash, name,
hashed_pin, phone, email, face_encoding.tobytes()))
        conn.commit()
        conn.close()

        return {"success": True, "message": f"User {name} registered
successfully."}

    except Exception as e:
        return {"success": False, "message": str(e)}

if __name__ == '__main__':
    app.run(debug=True)
```

6.2 admin_locations.py

```
from flask import Flask, render_template, request
from flask_sqlalchemy import SQLAlchemy
import folium
```

```
from datetime import datetime
import bandwidth

app = Flask(__name__)

# Configure SQLite database
app.config["SQLALCHEMY_DATABASE_URI"] = "sqlite:///locations.db"
app.config["SQLALCHEMY_TRACK_MODIFICATIONS"] = False
db = SQLAlchemy(app)

# Define the DeviceLocation model for storing device data
class DeviceLocation(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    device_id = db.Column(db.String(50), unique=True, nullable=False)  # Unique
device identifier
    lat = db.Column(db.Float, nullable=False)  # Latitude
    lon = db.Column(db.Float, nullable=False)  # Longitude
    timestamp = db.Column(db.DateTime, default=datetime.utcnow)  # Last updated
timestamp

# Initialize the database
with app.app_context():
    db.create_all()

@app.route("/update_location", methods=["POST"])
def update_location():
    data = request.json
    print("Raw request data:", request.data)
    print("Parsed JSON data:", data)

    if not data:
        print("No JSON data received.")
        return {"status": "error", "message": "No JSON data received."}, 400

    device_id = int(data.get("device_id"))

    try:
        api_token = bandwidth.get_access_token()
    except Exception as e:
        api_token = None
        print(f"Error fetching API token: {e}")

    try:
        if api_token:
            bandwidth.invocation(api_token, device_id)
        else:
            print("Skipping bandwidth invocation due to missing API token.")
```

```
    except Exception as e:
        print(f"Error invoking bandwidth: {e}")

    lat = data.get("lat")
    lon = data.get("lon")

    if not all([device_id, lat, lon]):
        missing = [k for k in ["device_id", "lat", "lon"] if not data.get(k)]
        print(f"Missing fields: {missing}")
        return {"status": "error", "message": f"Missing fields: {',
'.join(missing)}"}, 400

    try:
        # Check if the device already exists
        existing_device =
DeviceLocation.query.filter_by(device_id=device_id).first()
        if existing_device:
            # Update the existing device's location
            print(f"Updating location for device_id: {device_id}")
            existing_device.lat = float(lat)
            existing_device.lon = float(lon)
            existing_device.timestamp = datetime.utcnow()
        else:
            # Add a new device location
            print(f"Inserting new device_id: {device_id}")
            new_device = DeviceLocation(
                device_id=device_id, lat=float(lat), lon=float(lon)
            )
            db.session.add(new_device)
        db.session.commit()
    except Exception as e:
        print(f"Error inserting/updating data: {e}")
        return {"status": "error", "message": "Failed to insert or update data."},
500

    return {"status": "success", "message": "Location updated."}, 200

@app.route("/")
def index():
    """
    Display a map with all device locations.
    """
    devices = DeviceLocation.query.all()
    if not devices:
        return "No device locations available."

    # Calculate the map's center based on all devices' locations
```

```
    avg_lat = sum([device.lat for device in devices]) / len(devices)
    avg_lon = sum([device.lon for device in devices]) / len(devices)
    map_object = folium.Map(location=[avg_lat, avg_lon], zoom_start=10)

    # Add markers for all devices
    for device in devices:
        tooltip = f"Device: {device.device_id}\nLast Updated:
{device.timestamp.strftime('%Y-%m-%d %H:%M:%S')}"
        folium.Marker(
            [device.lat, device.lon],
            tooltip=tooltip,
            icon=folium.Icon(color="blue", icon="info-sign")
        ).add_to(map_object)

    # Render the map in an HTML template
    html_map = map_object._repr_html_()
    return render_template('device_locations.html', html_map=html_map)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5002, debug=True)
```

6.3 employee_location.py

```
import folium
from flask import Flask, render_template, request

import bandwidth

app = Flask(__name__)

# Store the latest location (default: None)
latest_location = {"lat": None, "lon": None}

@app.route("/update_location", methods=["POST"])
def update_location():
    """
    Receive real-time GPS data (latitude and longitude) from the iPhone.
    """
    global latest_location
    data = request.json

    device_id = int(data.get("device_id"))

    try:
        api_token = bandwidth.get_access_token()
    except Exception as e:
```

```
        api_token = None
        print(f"Error fetching API token: {e}")

    try:
        if api_token:
            bandwidth.invocation(api_token, device_id)
        else:
            print("Skipping bandwidth invocation due to missing API token.")
    except Exception as e:
        print(f"Error invoking bandwidth: {e}")

    if "lat" in data and "lon" in data:
        latest_location["lat"] = data["lat"]
        latest_location["lon"] = data["lon"]
        return {"status": "success"}, 200
    return {"status": "error", "message": "Invalid data"}, 400

@app.route("/")
def index():
    """
    Display the latest location on an interactive map.
    """
    if not latest_location["lat"] or not latest_location["lon"]:
        return "Waiting for location data..."

    # Create a map centered around the latest location
    map_object = folium.Map(location=[latest_location["lat"],
latest_location["lon"]], zoom_start=15)
    folium.Marker(
        [latest_location["lat"], latest_location["lon"]],
        tooltip="Current Location",
        icon=folium.Icon(color="green", icon="info-sign")
    ).add_to(map_object)

    # Render the map in an HTML template
    html_map = map_object._repr_html_()
    return render_template('device_locations.html', html_map=html_map)

if __name__ == "__main__":
    app.run(host="0.0.0.0", port=5001, debug=True)
```

6.4 facial_recognition.py

```
import cv2
import face_recognition
import sqlite3
```

```
from flask import json
import numpy as np
import hashlib
from twilio.rest import Client
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
import datetime
import time
import os
import smtplib
from tkinter import Tk, Label, Entry, Button, messagebox
import logging
from logging.handlers import TimedRotatingFileHandler

# Setup logging
LOG_DIR = 'logs'
LOG_FILE = 'access_log.log'

# Load the config file
with open("config.json", "r") as config_file:
    config = json.load(config_file)

if not os.path.exists(LOG_DIR):
    os.makedirs(LOG_DIR)

# Create and configure the logger
logger = logging.getLogger("FaceIDLogger")
logger.setLevel(logging.DEBUG)

# File Handler (rotates weekly, keeps 4 backups)
file_handler = TimedRotatingFileHandler(
    os.path.join(LOG_DIR, LOG_FILE),
    when="W0",  # Rotate weekly on Monday
    interval=1,
    backupCount=4
)

file_handler.setLevel(logging.INFO)  # File logs less critical data
file_handler.setFormatter(logging.Formatter('%(asctime)s - %(levelname)s -
%(message)s'))

# Console Handler (for debugging)
console_handler = logging.StreamHandler()
console_handler.setLevel(logging.DEBUG)  # Console logs all debugging info
console_handler.setFormatter(logging.Formatter('%(levelname)s: %(message)s'))
```

```
# Add handlers to logger
logger.addHandler(file_handler)
logger.addHandler(console_handler)

# Utility function to log with immediate flush
def log_with_flush(level, message):
    """
    Log a message and flush all handlers immediately.
    """
    try:
        logger.log(level, message)
        for handler in logger.handlers:
            handler.flush()
    except Exception as e:
        print(f"Logging failed: {e}")

# Load authorized users from the database
def load_authorized_users():
    try:
        conn = sqlite3.connect("instance/authorized_users.db")
        c = conn.cursor()
        c.execute("SELECT username, name, pin, image, phone, email FROM users")
        users = [(username, name, pin, np.frombuffer(image, dtype=np.float64),
phone, email)
                 for username, name, pin, image, phone, email in c.fetchall()]
        conn.close()
        return users
    except Exception as e:
        return []

# Notify user of successful access
def notify_access(username, email, phone):
    message = f"Access granted for {username}"
    log_with_flush(logging.INFO, message)
    try:
        send_sms(phone, message)
        send_email(email, "Access Granted", message)
    except Exception as e:
        logger.error(f"Failed to notify user {username}: {e}")

# Notify admin of unauthorized access
def notify_admin_unauthorized_access(video_path=None):
    message = f"Unauthorized access attempt detected at
{datetime.datetime.now().strftime('%Y-%m-%d %H:%M:%S')}"
    log_with_flush(logging.WARNING, message)
    try:
```

```
        send_sms(config["ADMIN_PHONE_NUMBER"], message)
        send_email(config["ADMIN_EMAIL"], "Unauthorized Access Alert", message,
video_path)
    except Exception as e:
        logger.error(f"Failed to notify admin: {e}")

# Send SMS
def send_sms(phone_number, message):
    client = Client(config["ACCOUNT_SID"], config["AUTH_TOKEN"])
    client.messages.create(to=phone_number, from_=config["TWILIO_PHONE_NUMBER"],
body=message)

# Send email with optional video attachment
def send_email(to_address, subject, message, attachment_path=None):
    msg = MIMEMultipart()
    msg['From'] = config["EMAIL_ADDRESS"]
    msg['To'] = to_address
    msg['Subject'] = subject
    msg.attach(MIMEText(message, 'plain'))
    if attachment_path and os.path.exists(attachment_path):
        with open(attachment_path, 'rb') as attachment:
            part = MIMEBase('application', 'octet-stream')
            part.set_payload(attachment.read())
            encoders.encode_base64(part)
            part.add_header('Content-Disposition', f'attachment;
filename="{os.path.basename(attachment_path)}"')
            msg.attach(part)
    try:
        with smtplib.SMTP(config["SMTP_SERVER"], config["SMTP_PORT"]) as server:
            server.starttls()
            server.login(config["EMAIL_ADDRESS"], config["EMAIL_PASSWORD"])
            server.send_message(msg)
    except Exception as e:
        logger.error(f"Failed to send email: {e}")

# Capture unauthorized 3-second video
def capture_unauthorized_video():
    video_path = "unauthorized_access.avi"
    log_with_flush(logging.INFO, "Starting unauthorized video capture.")
    try:
        video_capture = cv2.VideoCapture(0)
        fourcc = cv2.VideoWriter_fourcc(*'XVID')
        out = cv2.VideoWriter(video_path, fourcc, 20.0, (640, 480))
        start_time = time.time()

        while time.time() - start_time < 3:
            ret, frame = video_capture.read()
```

```
            if not ret:
                logger.error("Failed to capture video frame.")
                break
            out.write(frame)

        video_capture.release()
        out.release()
        logger.info(f"Video saved to {video_path}.")
        return video_path
    except Exception as e:
        logger.error(f"Failed to capture unauthorized video: {e}")
        return None

# Recognize faces and grant access based on username match
def recognize_faces(user):
    # Load authorized users from the database
    authorized_users = load_authorized_users()
    user_face_encoding = None
    user_phone = None
    user_email = None

    # Find the user's face encoding based on the provided username
    for username, _, _, image, phone, email in authorized_users:
        if username == user:
            user_face_encoding = image
            user_phone = phone
            user_email = email
            break

    if user_face_encoding is None:
        print("User not found in database.")
        notify_admin_unauthorized_access(None)  # Log and notify admin for
unrecognized access
        return False

    # Start the webcam
    video_capture = cv2.VideoCapture(0)
    if not video_capture.isOpened():
        print("Error: Could not open webcam.")
        return False

    start_time = time.time()
    recognized = False

    # Recognize face within 5 seconds
    while True:
        ret, frame = video_capture.read()
```

```
        if not ret:
            print("Failed to grab frame.")
            break

        rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        face_locations = face_recognition.face_locations(rgb_frame)

        if face_locations:
            face_encodings = face_recognition.face_encodings(rgb_frame,
face_locations)
            for face_encoding in face_encodings:
                match = face_recognition.compare_faces([user_face_encoding],
face_encoding)
                if True in match:
                    recognized = True
                    print(f"Access granted for authorized user: {username}")
                    notify_access(username, user_email, user_phone)  # Logging is
handled here
                    break

        if recognized:
            break

        # Check for unauthorized access if no face is recognized within 5 seconds
        if time.time() - start_time > 5:
            print("Unauthorized access: No face recognized within 5 seconds.")
            video_path = capture_unauthorized_video()
            notify_admin_unauthorized_access(video_path)
            break

        cv2.imshow("Face Recognition", frame)
        if cv2.waitKey(1) & 0xFF == ord("q"):
            break

    video_capture.release()
    cv2.destroyAllWindows()
    return recognized

# Hash the PIN for secure storage
def hash_pin(pin):
    return hashlib.sha256(pin.encode()).hexdigest()

# PIN verification and GUI
def verify_pin():
    entered_username = pin_username_entry.get()
    entered_pin = pin_entry.get()
    authorized_users = load_authorized_users()
```

```
    for username, _, pin, _, _, _ in authorized_users:
        if username == entered_username and hash_pin(entered_pin) == pin:
            if recognize_faces(username):
                messagebox.showinfo("Access Granted", "Welcome! Access Granted.")
            else:
                messagebox.showwarning("Access Denied", "Face recognition
failed.")
            return
    logger.warning(f"Access denied for user {entered_username}: Incorrect PIN or
username.")
    messagebox.showwarning("Access Denied", "Incorrect PIN or Username.")

# GUI for PIN input
def pin_input_gui():
    global pin_username_entry, pin_entry
    pin_root = Tk()
    pin_root.title("Enter PIN")
    pin_root.geometry("300x200")
    Label(pin_root, text="Username:", font=("Arial", 12)).pack(pady=5)
    pin_username_entry = Entry(pin_root, font=("Arial", 12))
    pin_username_entry.pack(pady=5)
    Label(pin_root, text="PIN:", font=("Arial", 12)).pack(pady=5)
    pin_entry = Entry(pin_root, show='*', font=("Arial", 12))
    pin_entry.pack(pady=5)
    Button(pin_root, text="Verify", command=verify_pin, font=("Arial",
12)).pack(pady=10)
    pin_root.mainloop()

if __name__ == "__main__":
    pin_input_gui()
```

6.5 bandwidth.py

```
import http.client
import json

def get_access_token():
    conn = http.client.HTTPConnection("192.168.3.18", 31002)
    payload = json.dumps({
        "client_id": "0b9972c1-a76f-4c44-9ee4-9f6990811434",
        "client_secret": "g1InNk1-kQRueYHj3DmvNALQK8pGw_U_dlpL38A6Rqc"
    })
    headers = {
        'Content-Type': 'application/json'
    }
```

```
    conn.request("POST", "/security/v1/token", payload, headers)
    res = conn.getresponse()
    data = res.read()
    token_data = json.loads(data.decode("utf-8"))
    access_token = token_data.get("access_token")
    conn.close()
    return access_token

def invocation(access_token, device_id):
    conn = http.client.HTTPConnection("192.168.3.18", 7999)
    payload = json.dumps({
        "device": {
            "deviceId": device_id
        },
        "maxBitRate": 400,
        "direction": "uplink",
        "duration": 10000
    })
    headers = {
        'Content-type': 'application/json',
        'Authorization': f'Bearer {access_token}'
    }
    conn.request("POST", "/qos/v1/bandwidth", payload, headers)
    res = conn.getresponse()
    data = res.read()
    print(data.decode("utf-8"))
    conn.close()

# Main execution
if __name__ == '__main__':
    access_token = get_access_token()
    if access_token:
        invocation(access_token)
    else:
        print("Failed to retrieve access token.")
```

6.6 homepage.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Cubed Access Portal</title>
    <style>
```

```
        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            text-align: center;
            overflow-x: hidden;
        }
        h1 {
            font-size: 3.5em;
            margin: 20px 0;
            color: #FFC107;
            text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.5);
        }
        p.intro {
            font-size: 1.3em;
            margin: 20px auto;
            max-width: 700px;
            line-height: 1.8;
        }
        .button {
            display: inline-block;
            margin: 20px;
            padding: 15px 30px;
            font-size: 18px;
            font-weight: bold;
            color: #fff;
            background: #17A2B8;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background 0.3s, transform 0.2s;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.3);
        }
        .button:hover {
            background: #138496;
            transform: translateY(-3px);
        }
        .icon {
            margin-right: 10px;
            vertical-align: middle;
            font-size: 0.8em; /* Adjust the size of the icon */
        }
        .fun-fact {
            background: rgba(255, 255, 255, 0.1);
            margin: 30px auto;
```

```
            max-width: 600px;
            padding: 20px;
            border-radius: 10px;
            line-height: 1.6;
            font-size: 1em;
            color: #F8F9FA;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
        }
        .footer {
            margin-top: 50px;
            font-size: 0.9em;
            color: #CED4DA;
        }
        .footer a {
            color: #FFC107;
            text-decoration: none;
        }
        .footer a:hover {
            text-decoration: underline;
        }
        .image-banner {
            width: 10%;
            max-height: 75px;
            margin: 10px auto;
            border-radius: 10px;
            border: 3px solid #FFC107;
            box-shadow: 0 4px 8px rgba(0, 0, 0, 0.5);
        }
    </style>
</head>
<body>
    <h1>🔒 Welcome to E-Cubed Access Portal</h1>
    <img
src="https://media.wired.com/photos/5a08a2d5e16e3b181d3dac41/191:100/pass/FaceID-
MainArt.jpg" alt="Security Banner" class="image-banner">
    <p class="intro">
        E-Cubed combines state-of-the-art technology with unmatched security
solutions to redefine how you protect your assets.
        Whether you're managing sensitive data or securing your enterprise, we
provide solutions tailored to your needs.
    </p>
    <div>
        <button class="button" onclick="window.location.href='/admin-login'">
            <span class="icon">👮</span>Admin Access
        </button>
        <button class="button" onclick="window.location.href='/employee-login'">
            <span class="icon">👷</span>Employee Access
```

```
        </button>
    </div>
    <div class="fun-fact">
        🌐 <strong>Tech Insight:</strong> Modern AI-driven cybersecurity systems
can identify and mitigate threats 60% faster than traditional methods, ensuring
proactive protection.
    </div>
    <div class="footer">
        <p>🔗 Explore more: <a href="/contact">Contact Us</a> | <a
href="/about">About Us</a></p>
    </div>
</body>
</html>
```

6.7 admin_login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Login</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap
');

        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            text-align: center;
        }
        h1 {
            margin-top: 50px;
            font-size: 2.5em;
            color: #ffc107;
            text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.5);
        }
        .login-container {
            margin: 50px auto;
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0px 0px 20px rgba(0, 0, 0, 0.5);
```

```
            max-width: 400px;
            text-align: left;
        }
        label {
            font-size: 1em;
            font-weight: 500;
            color: #ddd;
        }
        .form-group {
            margin-bottom: 20px;
        }
        .form-group input {
            width: 100%;
            height: 40px;
            font-size: 1rem;
            padding: 8px;
            border-radius: 5px;
            border: 1px solid #ccc;
            background: #f4f4f4;
        }
        .button-group {
            display: flex;
            flex-direction: column; /* Stack buttons vertically */
            gap: 15px; /* Add spacing between buttons */
            margin-top: 20px;
        }
        .button {
            padding: 10px;
            font-size: 1rem;
            font-weight: 500;
            text-align: center;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: transform 0.2s, background-color 0.3s;
            width: 100%; /* Full width button */
        }
        .button:hover {
            transform: scale(1.05);
        }
        .blue-button {
            background-color: #007bff;
            color: white;
        }
        .blue-button:hover {
            background-color: #0056b3;
        }
```

```
        .cancel-button {
            background-color: #dc3545;
            color: white;
        }
        .cancel-button:hover {
            background-color: #c82333;
        }
    </style>
</head>
<body>
    <h1>🔓 Admin Login</h1>
    <div class="login-container">
        <form id="loginForm" action="/admin-login" method="POST">
            <div class="form-group">
                <label for="username">Username:</label><br>
                <input type="text" id="username" name="username"
placeholder="Enter your username" required>
            </div>
            <div class="form-group">
                <label for="password">Password:</label><br>
                <input type="password" id="password" name="password"
placeholder="Enter your password" required>
            </div>
            <div class="button-group">
                <button type="submit" class="button blue-button"
id="loginButton">Login</button>
                <button type="button" class="button cancel-button"
id="cancelButton">Cancel</button>
            </div>
        </form>
    </div>

    <script>
        document.getElementById("cancelButton").addEventListener("click", () => {
            const loginForm = document.getElementById("loginForm");
            loginForm.reset();
            window.location.href = "/";
        });
    </script>
</body>
</html>
```

6.8 admin_dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>E-Cubed Admin Dashboard</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap
');

        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            text-align: center;
        }
        h1 {
            font-size: 3em;
            margin: 30px 0;
            color: #ffc107;
            text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.5);
        }
        p.intro {
            font-size: 1.2em;
            margin: 20px auto;
            max-width: 800px;
            line-height: 1.8;
        }
        .dashboard {
            margin-top: 40px;
        }
        .button {
            padding: 15px 30px;
            margin: 20px;
            font-size: 18px;
            font-weight: 500;
            border: none;
            border-radius: 8px;
            cursor: pointer;
            transition: transform 0.2s, box-shadow 0.3s;
            text-transform: none; /* Ensure text is displayed in normal case */
        }
        .button:hover {
            transform: scale(1.05);
            box-shadow: 0 0 15px rgba(0, 123, 255, 0.6);
        }
        .blue-button {
```

```
            background-color: #007bff;
            color: white;
        }
        .blue-button:hover {
            background-color: #0056b3;
        }
        .green-button {
            background-color: #28a745;
            color: white;
        }
        .green-button:hover {
            background-color: #218838;
        }
        .fun-fact {
            margin: 40px auto;
            max-width: 700px;
            padding: 20px;
            font-size: 1em;
            background: rgba(255, 255, 255, 0.1);
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.3);
            line-height: 1.6;
            color: #e0f7fa;
        }
        .footer {
            margin-top: 50px;
            font-size: 0.9em;
            color: #CED4DA;
        }
        .footer a {
            color: #ffc107;
            text-decoration: none;
        }
        .footer a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <h1>🔓 Welcome to E-Cubed Admin Dashboard</h1>
    <p class="intro">
        Manage and monitor all critical security operations with precision and
efficiency. Your tools are ready to empower secure decisions.
    </p>
    <div class="dashboard">
        <button class="button blue-button"
onclick="window.location.href='/register-user'">Register User</button>
```

```
        <button class="button blue-button" onclick="window.location.href='/update-user'">Update User</button>
        <button class="button blue-button" onclick="window.location.href='/locations'">Locations</button>
        <button class="button blue-button" onclick="window.location.href='/facial-recognition'">Facial Recognition</button>
        <button class="button blue-button" onclick="window.location.href='/logs'">Logs</button>
        <button class="button green-button" onclick="window.location.href='/'">Home</button>
    </div>
    <div class="fun-fact">
        🌐 <strong>Tech Insight:</strong> Modern cybersecurity dashboards leverage real-time analytics to detect anomalies faster, enhancing response times and reducing vulnerabilities.
    </div>
    <div class="footer">
        <p>🥷 Powered by Innovation at <a href="#">E-Cubed</a></p>
        <p><a href="/contact">Contact Us</a> | <a href="/about">About Us</a> | <a href="/help">Help Center</a></p>
    </div>
</body>
</html>
```

6.9 register_user.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Register User</title>
    <style>
        @import url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap');

        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            color: #fff;
```

```
        }
        .register-form {
            background: rgba(255, 255, 255, 0.1);
            padding: 20px 30px; /* Increased padding for balanced spacing on left
and right */
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.5);
            width: 400px;
            text-align: left;
            margin: auto;
        }
        h2 {
            text-align: center;
            color: #ffc107;
            margin-bottom: 20px;
        }
        label {
            display: block;
            margin-top: 10px;
            color: #ddd;
            font-weight: 500;
        }
        input[type="text"], input[type="email"], input[type="password"], select,
input[type="file"] {
            width: 100%;
            padding: 10px;
            margin-top: 5px;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 1rem;
            box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        }
        button {
            margin-top: 15px;
            padding: 10px;
            background: #007bff;
            border: none;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
            font-size: 1rem;
            transition: background 0.3s, transform 0.2s;
        }
        button:hover {
            background: #0056b3;
            transform: scale(1.05);
```

```
        }
        .buttons-container {
            margin-top: 20px;
            display: flex;
            justify-content: space-between;
            gap: 20px; /* Add gap for spacing between the buttons */
        }

        .home-button, .back-button {
            padding: 10px 20px;
            background: #28a745;
            border: none;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
            font-size: 1rem;
            transition: background 0.3s, transform 0.2s;
        }

        .home-button:hover {
            background: #218838;
        }

        .back-button {
            background: #6c757d;
        }

        .back-button:hover {
            background: #5a6268;
        }

    </style>
</head>
<body>
    <div class="register-form">
        <h2>🔒 Register User</h2>

        <form id="registrationForm" enctype="multipart/form-data">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" placeholder="Enter
username" required>

            <label for="access_level">Access Level:</label>
            <select id="access_level" name="access_level" required>
                <option value="employee">Employee</option>
                <option value="admin">Admin</option>
            </select>
```

```
            <label for="password">Password:</label>
            <input type="password" id="password" name="password"
placeholder="Enter password" required>

            <label for="name">Name:</label>
            <input type="text" id="name" name="name" placeholder="Enter full name"
required>

            <label for="pin">PIN:</label>
            <input type="password" id="pin" name="pin" placeholder="Enter PIN"
required>

            <label for="phone">Phone:</label>
            <input type="text" id="phone" name="phone" placeholder="Enter phone
number" required>

            <label for="email">Email:</label>
            <input type="email" id="email" name="email" placeholder="Enter email
address" required>

            <label for="image">Face Image:</label>
            <input type="file" id="image" name="image" accept="image/*" required>

            <button type="button" onclick="submitForm()">Register</button>
        </form>

        <div class="buttons-container">
            <button class="back-button" onclick="window.location.href='/admin-
dashboard'">Back</button>
            <button class="home-button"
onclick="window.location.href='/'">Home</button>
        </div>
    </div>

    <script>
        function submitForm() {
            const formData = new
FormData(document.getElementById("registrationForm"));
            const registerButton =
document.querySelector("button[type='button']");

            registerButton.disabled = true;
            registerButton.textContent = "Registering...";

            fetch("/register-user", {
                method: "POST",
```

```
            body: formData,
        })
        .then(response => response.json())
        .then(data => {
            if (data.success) {
                alert("Registration successful: " + data.message);
                document.getElementById("registrationForm").reset();
            } else {
                alert("Registration failed: " + data.message);
            }
        })
        .catch(error => {
            console.error("Error:", error);
            alert("An unexpected error occurred. Please try again.");
        })
        .finally(() => {
            registerButton.disabled = false;
            registerButton.textContent = "Register";
        });
    }
</script>
</body>
</html>
```

6.10 update_user.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Update User Information</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap
');

        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            text-align: center;
        }
        .container {
            max-width: 800px;
```

```
            margin: 50px auto;
            background: rgba(255, 255, 255, 0.1);
            padding: 20px 30px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.5);
            text-align: left; /* Align text inside the container */
        }

        h2 {
            text-align: center;
            color: #ffc107;
            font-size: 2rem; /* Adjust font size if needed */
            margin-bottom: 20px; /* Add spacing below the heading */
        }

        .image-container {
            text-align: center; /* Center the image horizontally */
            margin-bottom: 20px; /* Add some spacing below the image */
        }

        .image-container img {
            width: 100%;
            max-width: 400px;
            height: auto;
            border-radius: 8px;
            box-shadow: 0 0 10px rgba(0, 0, 0, 0.3);
        }
        .form-group {
            margin-top: 20px;
            margin-bottom: 15px;
        }
        label {
            display: block;
            font-weight: bold;
            color: #ddd;
            margin-bottom: 5px;
        }
        input[type="text"],
        input[type="email"],
        input[type="file"],
        select {
            width: 100%;
            padding: 10px;
            margin-top: 5px;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 16px;
```

```
            box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        }
        button {
            padding: 10px 15px;
            font-size: 16px;
            margin-right: 10px;
            cursor: pointer;
            border: none;
            border-radius: 5px;
            transition: transform 0.2s, background 0.3s;
        }
        button:hover {
            transform: scale(1.05);
        }
        .cancel-button {
            background-color: #f44336;
            color: white;
        }
        #retrieveButton, #updateButton {
            background-color: #007bff;
            color: white;
        }
        .home-button {
            background-color: #28a745;
            color: white;
        }
        .button-group {
            text-align: center;
            margin-top: 20px;
        }
    </style>
</head>
<body>

<div class="container">
    <h2>🔄 Update User Information</h2>

    <div class="image-container">
        <img
src="https://d1nhio0ox7pgb.cloudfront.net/_img/v_collection_png/512x512/shadow/bus
inessman_edit.png" alt="Teamwork">
    </div>

    <!-- User ID to identify the user to update -->
    <div class="form-group">
        <label for="username">User Name:</label>
```

```
        <input type="text" id="username" name="username" placeholder="Enter User
Name" required>
    </div>

    <div class="button-group">
        <!-- Retrieve Button -->
        <button type="button" id="retrieveButton">Retrieve</button>
        <button type="button" class="cancel-button"
id="topCancelButton">Cancel</button>
    </div>

    <form id="updateForm" style="display:none;" enctype="multipart/form-data">
        <div class="form-group">
            <label for="email">Email:</label>
            <input type="email" id="email" name="email" placeholder="Enter email
address">
        </div>
        <div class="form-group">
            <label for="name">Name:</label>
            <input type="text" id="name" name="name" placeholder="Enter full
name">
        </div>
        <div class="form-group">
            <label for="phone">Phone:</label>
            <input type="text" id="phone" name="phone" placeholder="Enter phone
number">
        </div>
        <div class="form-group">
            <label for="access_level">Access Level:</label>
            <select id="access_level" name="access_level">
                <option value="Employee">Employee</option>
                <option value="Admin">Admin</option>
            </select>
        </div>
        <div class="form-group">
            <label for="password">New Password (Optional):</label>
            <input type="text" id="password" name="password" placeholder="Enter a
strong password">
        </div>
        <div class="form-group">
            <label for="pin">New PIN (Optional):</label>
            <input type="text" id="pin" name="pin" placeholder="Enter a secure
PIN">
        </div>
        <div class="form-group">
            <label for="image">New Image (Optional):</label>
            <input type="file" id="image" name="image" accept="image/*">
```

```
        </div>
        <div class="button-group">
            <button type="button" id="updateButton">Update</button>
            <button type="button" class="cancel-button"
id="cancelButton">Cancel</button>
            <button type="button" class="home-button"
id="homeButton">Home</button>
        </div>
    </form>
</div>

<script>
document.getElementById("retrieveButton").addEventListener("click", function () {
    const username = document.getElementById("username").value;
    if (username) {
        fetch(`/retrieve_user/${username}`)
            .then(response => response.json())
            .then(data => {
                if (data.success) {
                    document.getElementById("email").value = data.user.email;
                    document.getElementById("name").value = data.user.name;
                    document.getElementById("phone").value = data.user.phone;

                    const accessLevelDropdown =
document.getElementById("access_level");
                    for (let i = 0; i < accessLevelDropdown.options.length; i++) {
                        if (accessLevelDropdown.options[i].value.toLowerCase() ===
data.user.access_level.toLowerCase()) {
                            accessLevelDropdown.options[i].selected = true;
                            break;
                        }
                    }

                    document.getElementById("password").value = "";
                    document.getElementById("pin").value = "";
                    document.getElementById("image").value = ""; // Keep empty for
new upload
                    document.getElementById("updateForm").style.display = "block";
                } else {
                    alert("User not found.");
                }
            })
            .catch(error => {
                console.error("Error fetching user data:", error);
                alert("An error occurred while retrieving user data.");
            });
    } else {
```

```
        alert("Please enter a User Name.");
    }
});

document.getElementById("updateButton").addEventListener("click", function () {
    const username = document.getElementById("username").value;
    const name = document.getElementById("name").value;
    const email = document.getElementById("email").value;
    const phone = document.getElementById("phone").value;
    const access_level = document.getElementById("access_level").value;
    const password = document.getElementById("password").value;
    const pin = document.getElementById("pin").value;
    const image = document.getElementById("image").files[0];

    const formData = new FormData();
    formData.append("username", username);
    formData.append("name", name);
    formData.append("email", email);
    formData.append("phone", phone);
    formData.append("access_level", access_level);

    if (password) formData.append("password", password);
    if (pin) formData.append("pin", pin);
    if (image) formData.append("image", image);

    fetch(`/update_user/${username}`, {
        method: 'POST',
        body: formData,
    })
    .then(response => response.json())
    .then(data => {
        if (data.success) {
            alert("User updated successfully.");
            document.getElementById("updateForm").style.display = "none";
        } else {
            alert("Failed to update user: " + data.message);
        }
    })
    .catch(error => {
        console.error("Error updating user:", error);
        alert("An error occurred while updating the user.");
    });
});

document.getElementById("cancelButton").addEventListener("click", function () {
    window.location.href = "/admin-dashboard";
});
```

```
document.getElementById("topCancelButton").addEventListener("click", function () {
    window.location.href = "/admin-dashboard";
});

document.getElementById("homeButton").addEventListener("click", function () {
    window.location.href = "/";
});
</script>

</body>
</html>
```

6.11 locations.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Locations</title>
    <style>
        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
        }
        .locations-container {
            background: rgba(255, 255, 255, 0.1);
            padding: 20px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.5);
            width: 400px;
            text-align: center;
        }
        h2 {
            font-size: 1.8em;
            color: #ffc107;
            margin-bottom: 10px;
        }
        p.description {
```

```
            font-size: 1em;
            margin: 10px 0;
            line-height: 1.5;
            color: #ddd;
        }
        .map-container {
            margin: 15px 0;
            border-radius: 8px;
            overflow: hidden;
        }
        img {
            width: 100%;
            height: auto;
            border-radius: 8px;
            box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.3);
        }
        .fun-fact {
            margin: 20px 0;
            padding: 15px;
            background-color: rgba(255, 255, 255, 0.1);
            border-radius: 8px;
            font-size: 0.9em;
            color: #e0f7fa;
        }
        button {
            margin-top: 10px;
            padding: 10px;
            background: #007bff;
            border: none;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
            font-size: 1rem;
            transition: transform 0.2s, background 0.3s;
        }
        button:hover {
            background: #0056b3;
            transform: scale(1.05);
        }
        .back-button {
            margin-top: 10px;
            padding: 10px;
            background: #6c757d;
            border: none;
            color: #fff;
            border-radius: 5px;
```

```
            cursor: pointer;
            width: 100%;
            font-size: 1rem;
            transition: transform 0.2s, background 0.3s;
        }
        .back-button:hover {
            background: #5a6268;
            transform: scale(1.05);
        }
    </style>
</head>
<body>
    <div class="locations-container">
        <h2>📍 E-Cubed Locations</h2>
        <p class="description">Dive into E-Cubed's global impact. This visual
representation captures the essence of our international presence and innovative
reach.</p>

        <!-- Updated Map Section -->
        <div class="map-container">
            <img src="https://jooinn.com/images/earth-from-space-10.jpg"
alt="Map">
        </div>

        <!-- Fun Fact Section -->
        <div class="fun-fact">
            🌍 <strong>Did you know?</strong> Location intelligence enhances
security by 30%, ensuring better insights for operational decisions.
        </div>

        <!-- Navigation Buttons -->
        <a href="{{ map_url }}" target="_blank">
            <button type="button">View Full Map</button>
        </a>
        <button class="back-button" onclick="window.location.href='{{ back_url
}}'">Back to Dashboard</button>
    </div>
</body>
</html>
```

6.12 employee_login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

```
    <title>Employee Login</title>
    <style>
        @import
url('https://fonts.googleapis.com/css2?family=Roboto:wght@400;500;700&display=swap
');

        body {
            font-family: 'Roboto', sans-serif;
            margin: 0;
            padding: 0;
            background: linear-gradient(135deg, #141E30, #243B55);
            display: flex;
            justify-content: center;
            align-items: center;
            height: 100vh;
            color: #fff;
        }
        .login-container {
            background: rgba(255, 255, 255, 0.1);
            padding: 30px;
            border-radius: 10px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.5);
            width: 400px; /* Sets the container width */
            text-align: left;
        }
        h2 {
            text-align: center;
            color: #ffc107;
            margin-bottom: 20px;
        }
        label {
            display: block;
            margin-top: 10px;
            color: #ddd;
            font-weight: 500;
        }
        input[type="text"],
        input[type="password"] {
            width: 100%; /* Matches the container width */
            padding: 10px;
            margin-top: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            font-size: 1rem;
            box-shadow: 0 0 5px rgba(0, 0, 0, 0.1);
        }
        button {
```

```
            margin-top: 15px;
            padding: 12px;
            background: #007bff;
            border: none;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
            font-size: 1rem;
            transition: background 0.3s, transform 0.2s;
        }
        button:hover {
            background: #0056b3;
            transform: scale(1.05);
        }
        .cancel-button {
            margin-top: 10px;
            padding: 12px;
            background: #f44336; /* Red cancel button */
            border: none;
            color: #fff;
            border-radius: 5px;
            cursor: pointer;
            width: 100%;
            font-size: 1rem;
            transition: background 0.3s, transform 0.2s;
        }
        .cancel-button:hover {
            background: #d32f2f; /* Darker red on hover */
            transform: scale(1.05);
        }
    </style>
</head>
<body>
    <div class="login-container">
        <h2>🔒 Employee Login</h2>
        <form id="employeeLoginForm" action="/employee-login" method="POST">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" placeholder="Enter
your username" required>

            <label for="password">Password:</label>
            <input type="password" id="password" name="password"
placeholder="Enter your password" required>

            <button type="submit">Login</button>
        </form>
```

```
        <!-- Red Cancel Button -->
        <button class="cancel-button"
onclick="window.location.href='/'">Cancel</button>
    </div>
</body>
</html>

    <script>
        // Clear the form inputs for each rendering
        document.addEventListener("DOMContentLoaded", () => {
            const loginForm = document.getElementById("loginForm");
            loginForm.reset(); // Clear all inputs when the page loads
        });

        // Cancel button to clear the form and redirect to Home page
        document.getElementById("cancelButton").addEventListener("click", () => {
            const loginForm = document.getElementById("loginForm");
            loginForm.reset(); // Clear the form inputs
            window.location.href = "/"; // Redirect to the Home page
        });
    </script>
</body>
</html>
```

6.13 employee_dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Employee Dashboard</title>
    <style>
        body {
            font-family: 'Roboto', sans-serif;
            background: linear-gradient(135deg, #141E30, #243B55);
            color: #fff;
            text-align: center;
            margin: 0;
            padding: 0;
        }
        h1 {
            font-size: 2.5em;
            margin-top: 20px;
            color: #ffc107;
```

```
            text-shadow: 2px 2px 5px rgba(0, 0, 0, 0.5);
        }
        p.intro {
            font-size: 1.2em;
            margin: 20px auto;
            max-width: 800px;
            line-height: 1.8;
            color: #ddd;
        }
        .dashboard {
            margin-top: 30px;
        }
        .button {
            padding: 15px 30px;
            margin: 10px;
            font-size: 18px;
            border: none;
            border-radius: 8px;
            cursor: pointer;
            transition: transform 0.2s, box-shadow 0.2s;
        }
        .button:hover {
            transform: scale(1.1);
            box-shadow: 0 0 15px rgba(255, 255, 255, 0.2);
        }
        .blue-button {
            background-color: #007bff;
            color: white;
        }
        .blue-button:hover {
            background-color: #0056b3;
        }
        .green-button {
            background-color: #28a745;
            color: white;
        }
        .green-button:hover {
            background-color: #218838;
        }
        .image-container {
            margin: 20px auto;
            width: 80%;
            max-width: 600px;
            border-radius: 8px;
            overflow: hidden;
        }
        .image-container img {
```

```
            width: 100%;
            height: auto;
            border-radius: 8px;
            box-shadow: 0px 0px 15px rgba(0, 0, 0, 0.3);
        }
        .fun-fact {
            margin: 30px auto;
            max-width: 600px;
            padding: 15px;
            font-size: 1em;
            background-color: rgba(255, 255, 255, 0.1);
            border-radius: 10px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.2);
            line-height: 1.6;
            text-align: left;
        }
        .footer {
            margin-top: 50px;
            font-size: 0.9em;
            color: #ddd;
        }
        .footer a {
            color: #ffc107;
            text-decoration: none;
        }
        .footer a:hover {
            text-decoration: underline;
        }
    </style>
</head>
<body>
    <h1>📋 Welcome to the Employee Dashboard</h1>
    <p class="intro">
        This is your go-to platform for managing your tasks, accessing tools, and
staying connected. At E-Cubed, we empower you with the resources to excel.
    </p>
    <div class="dashboard">
        <button class="button blue-button"
onclick="window.location.href='/locations'">Locations</button>
        <button class="button blue-button" onclick="window.location.href='/facial-
recognition'">Facial Recognition</button>
        <button class="button green-button"
onclick="window.location.href='/'">Home</button>
    </div>
    <div class="image-container">
        <img
src="https://th.bing.com/th/id/R.4fc2a5ad08eae53099a21c0b75bf48ad?rik=SRQmTm%2fYu0
```

```
rh4g&riu=http%3a%2f%2femilysquotes.com%2fwp-
content%2fuploads%2f2015%2f04%2fEmilysQuotes.Com-work-hard-work-silence-success-
noise-advice-attitude-inspirational-motivational-Frank-
Ocean.jpg&ehk=uN%2f1P2sSB2uOCv0lcRShHWvf3zXCQSHuhTCoJvL7RBo%3d&risl=&pid=ImgRaw&r=
0" alt="Team Collaboration">
    </div>
    <div class="fun-fact">
        🌟 <strong>Did you know?</strong> Collaborative tools like this dashboard
can improve team productivity by up to 30%.
    </div>
    <div class="footer">
        <p>🔗 Powered by Innovation at <a href="#">E-Cubed</a></p>
        <p><a href="/contact">Contact Us</a> | <a href="/about">About Us</a> | <a
href="/help">Help Center</a></p>
    </div>
</body>
</html>
```

6.14 config.json

This file link to facial_recognition.py. It contains some of your personal credentials as well as other information from your Twilio account.

```
{
    "ACCOUNT_SID": "your actual account SID",
    "AUTH_TOKEN": "your actual auth token",
    "TWILIO_PHONE_NUMBER": "+1 your given Twilio phone number",
    "SMTP_SERVER": "smtp.gmail.com", # if you have a gmail address. If not, change
“gmail” to your mail platform.
    "SMTP_PORT": "587",
    "EMAIL_ADDRESS": "your email address",
    "EMAIL_PASSWORD": "your app password generated in your mail account",
    "ADMIN_EMAIL": "email of admin",
    "ADMIN_PHONE_NUMBER": "+1 phone number of admin"
}
```

*Note: All the .html files need to be placed in a folder intitled “templates”

6.15 Bill of Materials (BOM)

6.15.1 BOM

PART NUMBER	PART NAME	DESCRIPTION	LINK	QUANTITY	UNITS	SUPPLIER / MANUFACTURER	UNIT COST	TOTAL PART COST
1	OpenCV	Face Recognition API	OpenCV	1	NA	OpenCV	$0	$0
2	PIP libraries	Library for python	Pip	1	NA	PIP	$0	$0
3	Shabodi	Sandbox	Shabodi	1	NA	Shabodi	$0	$0
4	Twilio	Notification API	Twilio	1	milliseconds	Twilio	$0	$0
5	Shabodi Bandwidth	API	Shabodi	1	bits per second	Shabodi	$0	$0
6	Flask	API	Flask	1	N/A	Flask	$0	$0
			TOTAL PARTS	7		TOTAL		$0.00

6.15.2 Equipment list

The equipment needed to build this system you will need to use all libraries, APIs, and software listed in the BOM. In addition to these materials, you will also need to have a door with a pin system, a cellular phone, and a camera (web camera or external). These materials were all simulated in our prototype by using a laptop and an iPhone.

The software equipment required is a sever with Phython 3.8+ installed. The following libraries: Flask, SQLAlchemy, OpenCV, face_recognition, bcrypt, Twilio SDK, Folium. A web browser such as Chrome or Frefox, but any modern browser will work.

6.16 Testing & Validation

This prototype went through a series of test. The facial recognition, notification and log subsystems were tested individually. The remaining subsystems were tested in a comprehensive

medium fidelity model. Majority of the testing was done by physical trials and returning Boolean values.

Starting with facial recognition this system was tested by having only one person in the directory and then running the system to see if it was able to correctly match the individual to the directory. Once this test was successful more than 10 times, we began testing to see if it cannot recognize faces. During this testing period we exposed the system repeatedly to a variety of faces either the original person in the directory or unregistered people. These results were very successful and worked 100% of the time. For the final testing before integrating the whole system we added additional people to the directory. Like the test previously the system was able to correctly assign people and not permit access unnecessarily.

Figure 15. Example of Correct Access Assignment

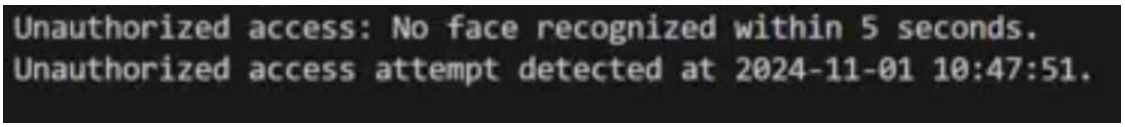


Figure 16. Example of Correct UAAE Assignment

The notification system was implemented next. We tested this system by running the facial recognition system cases of true or false. In these test trials we were recording two types of data. The first set of data was if the notification being sent was accurate (true/false) and the second set was the speed of notification. Initial the notification being sent was accurate and fast (under 2 seconds) but it did not contain an image or video of the UAAE. Adding a photo of the UAAE to the notification the results only changed in terms of speed. The length of time between the event and receiving the notification is about 4 seconds. Finally, adding a 3 second video of the event the notification delay time is about 5 seconds. We got these values from physically timing the notification and then graphing values to extrapolate the estimated value.

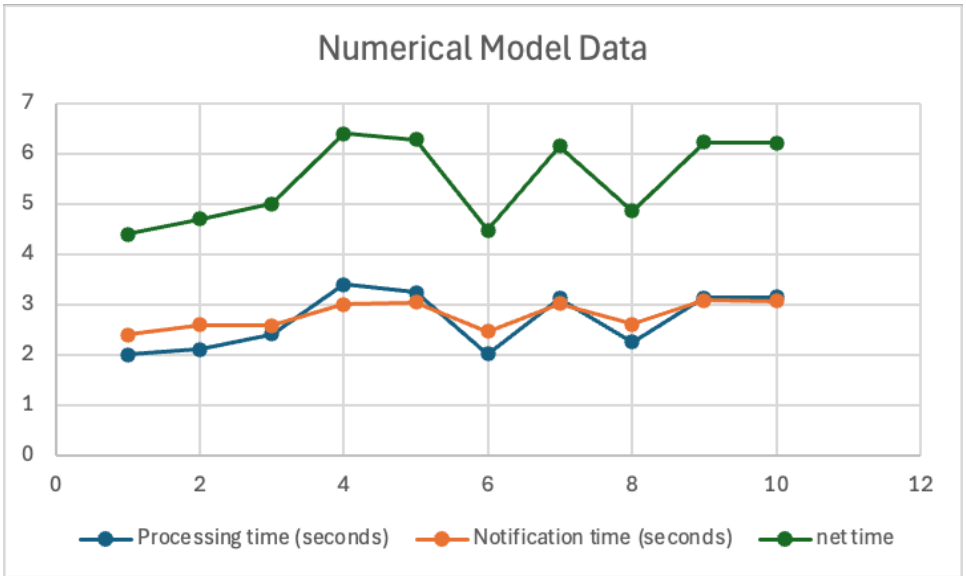


Figure 17. Response Time to Unauthorized and Authorized Access Attempts

The log system was tested like the notification system, where all trials were run in conjunction with the facial recognition system. We ran the log system multiple times and counted the number of unnecessary notifications in the system. Initially the log would add everything the python file would do. After making modifications to the code the log system was only adding access attempts. We continued these trials to ensure that this system would log everything correctly more than 95% of the time.

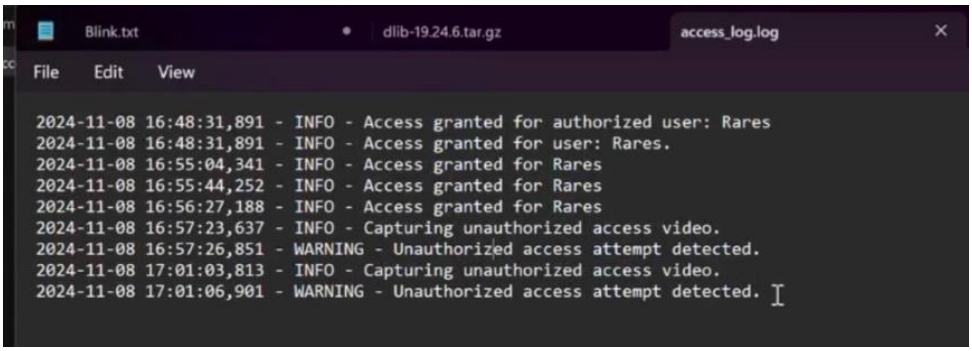


Figure 18. Image of Access Log after Consecutive Log in Attempts

For the final set of testing, we had built a comprehensive model. Testing here tried to model a real world. We added and updated different individual into the systems. We had the ‘employees’ and unregistered people present themselves to the camera to ensure that the facial recognition system, notification and log system worked as expected. When testing here the issue of not adding a password to an employee appeared. If no password was assigned to an employee, it would make the password a space. Once corrected the test results came back as working 100% of the time. For location tracking we tested using only one iPhone. We had this user register their phone number and use the shortcut functions on the phone. We had them appear in various locations and then run the tracking subsystem. We measured out the coordinates of the ‘pin’ from the generated map to where they were. We did these tests over a course of a week, and in different locations. The average distance result was with in 5 meters.

Table 3. Results from Medium Fidelity Comprehensive Prototype

Subsystem	Accuracy	Number of attempts
Facial recognition	95%	20
Location tracking	Accurate to 20 $m^2 \approx$ 4-5 m radius	20
Notification	100%	20
Log-In System	100%	20
Registration - New	100%	2
Registration - Update	100%	5

7 Conclusions and Recommendations for Future Work

To conclude, we have created a net-aware application which consists of an administrator interphase and an employee interphase. Our application utilizes 4 main subsystems which include facial recognition, notification, log, and location. Our recommendation for future work is to stay open to new ideas and be open to criticism from others and team members. Through our projects our team members had to navigate feedback from the client, teacher, project manager, other students, and between team members. It's hard to navigate through all this feedback but it is easier if the team members are close and help each other get through it together.

If another group of students would like to continue this project, they could work on implementing a system that is able to process multiple faces at once. This can help prevent people using others pins while they are present to gain access to other areas. We were not able to implement this due to time constraints. Another idea other students could add if they had more time could add a way to restrict access to specific people at the will of the administration.

8 Bibliography

There were no external sources used in this user manual.

APPENDIX

9 APPENDIX I: Design Files

This user manual is a guide to the restricted zone alerting system we had developed over a course of the semester. If there are further questions about the development of the idea, the client's needs, and detailed descriptions of all prototypes they can be answered by referring to our deliverables and presentations. All those documents are found on our MarkerRepo page. The link to the page is: https://makerepo.com/ebela099/2171.gng1103-shabodi-restricted-zone-alerting-group2-e-cubed.

Table 4. Referenced Documents

Document Name	Document Location and/or URL	Issuance Date
Deliverable C	MakerRepo	October 6th, 2024
Deliverable D	MakerRepo	October 13th, 2024
Deliverable D - Slides	MakerRepo	October 13th, 2024
Deliverable E	MakerRepo	October 27th, 2024
Deliverable F	MakerRepo	November 3rd, 2024
Deliverable F - slides	MakerRepo	November 3rd, 2024
Deliverable G	MakerRepo	November 10th, 2024
Deliverable H	MakerRepo	November 24th, 2024