AuthentiFace

Working Manual

Contents

1. **Overview** 1

2. **Features** 2

3. **Requirements** 4

4. **Setup** 6

5. **Workflow** 35

6. **Results & Discussion** 43

Overview

AuthentiFace is a sophisticated email system designed to facilitate secure internal communication within organisations. This email platform offers a range of features and security measures to ensure that sensitive information remains confidential while streamlining the exchange of messages and attachments among employees.

AuthentiFace prioritizes the security and confidentiality of communication within your organisation. By employing advanced authentication methods and secure data storage, this email system minimizes the risk of unauthorized access to sensitive emails.

One of Authentiface's primary strengths lies in its versatility when handling attachments. Users can send and receive emails with a wide array of attachment formats, including images, videos, PDFs, and text documents. This flexibility empowers users to share diverse types of information seamlessly.

Features

AuthentiFace offers the following key features:

Secure Email System

- Users can send and receive emails within the organisation, keeping communication organised and efficient.
- Email attachments of various formats (image, video, PDF, text) are supported.
- Email data is stored in a secure MongoDB database.

User Authentication

- Admin registration is required to create user accounts, ensuring the organisation's security.
- Users must provide their credentials (username and password) to log in.

Features

User Profile

- Each user has a dedicated profile page that includes personal information such as photo, name, position, teams, skills, and contact details.
- A logout button is available for users to safely log out of their accounts.

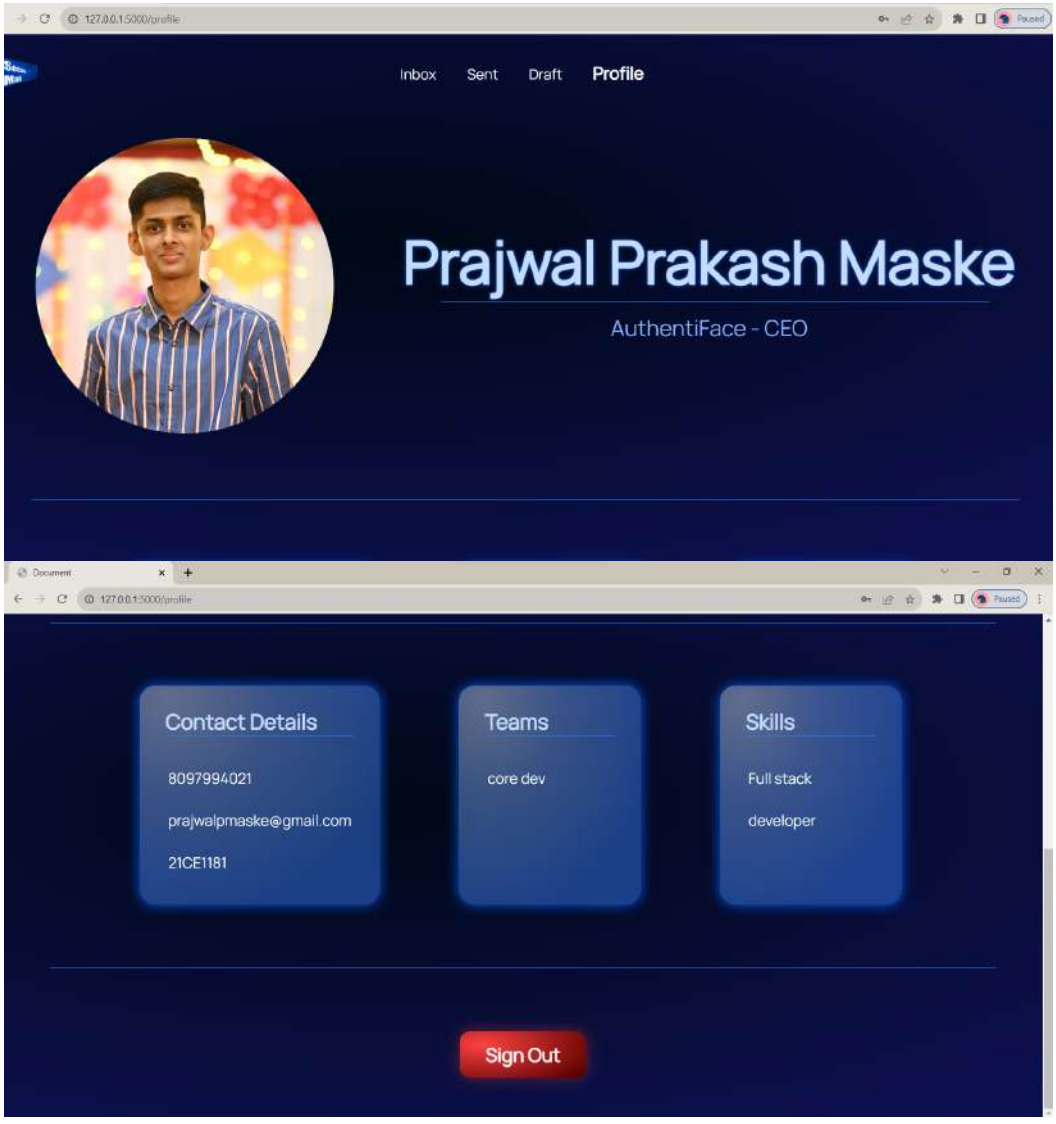

Fig 2.1

Requirements

Software Dependencies

To implement AuthentiFace, ensure that you have the following software dependencies in place:

- Python and Libraries: Python is the core programming language. Install libraries like flask_pymongo, flask, opencv-python (cv2), face_recognition, numpy, base64, datetime, passlib (bcrypt), bson, keras, pandas, sklean, io (BytesIO) and tensorflow to handle various functionalities.

- MongoDB Database: AuthentiFace relies on MongoDB for storing email data securely. Ensure that you have a MongoDB database set up and configured appropriately.

- Code Editor: Download Visual Studio Code (Integrated Development and Learning Environment) https://code.visualstudio.com/

Requirements

Hardware Dependencies

- Processor: i5, AMD Ryzen 5600h or equivalent Quad-Core CPU.
- Memory: 10GB of available space.
- RAM: 8GB DDR4 3200mhz
- Operating System: Windows 10/11

Project Source Code

You can access our project source code here: https://drive.google.com/drive/folders/1-Hi1ksAvYZ188ewYf4w8rAE5U2rKju5R

Download this source code in your device as it is the most important part.

NOTE: Open this link using a high end device like laptop or PC, as it is a zip file you won't be able to download it

Set-Up

Python

- Install latest version of ‘python’ (v3.12.0) and the package manager ‘pip’.
- Once you have installed python version, follow thee below steps:

Suppose, the name of python folder which you have installed is ‘Python312’ , open this folder and look for folder named ‘Scripts’, open it and copy the url of this folder location as shown in fig 1.1

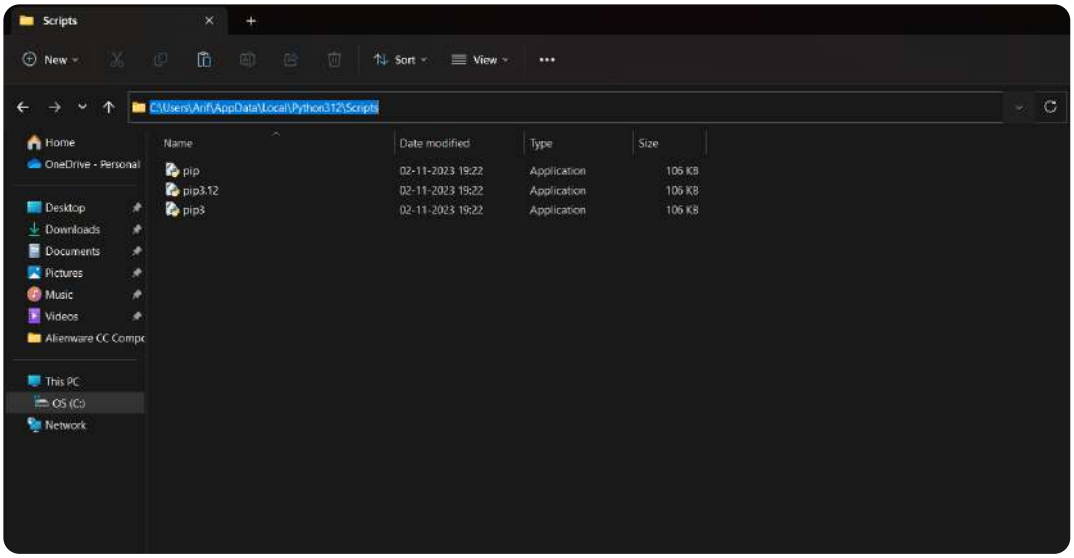

Fig 4.1

Set-Up

Now Search for ‘View Advanced System Settings’ and open it. Click on the ‘Environment Variables..’ as shown in fig 1.2

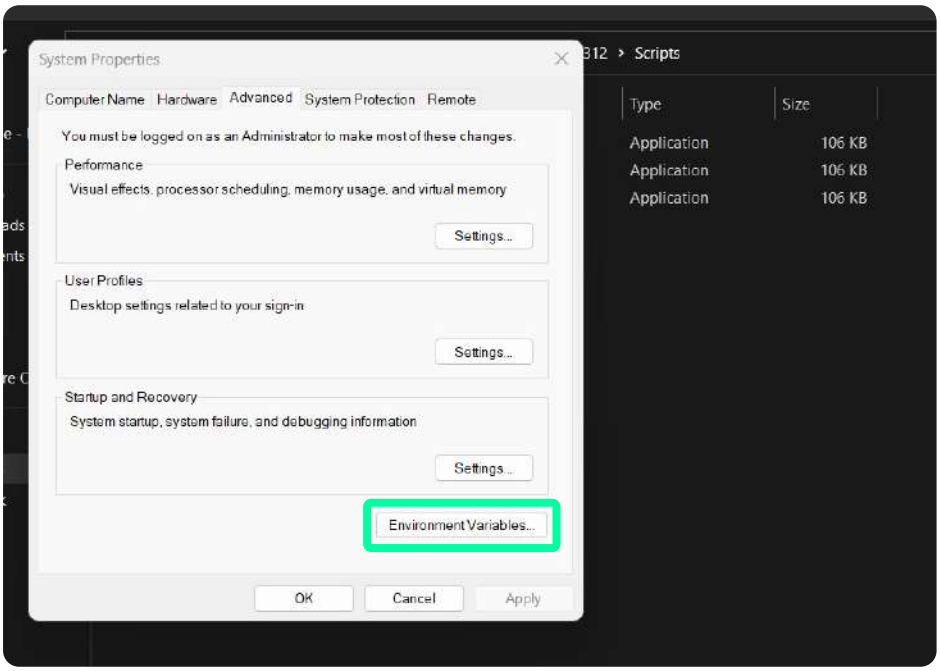

Fig 4.2

Now under the ‘System Variables’ select ‘Path’ and then click on ‘Edit’ as shown in fig 1.3

Set-Up

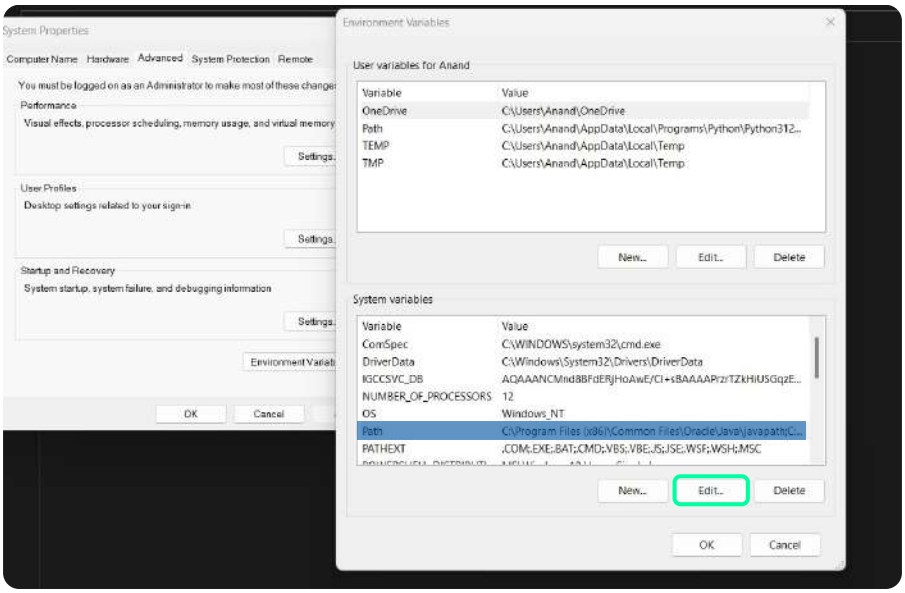

Fig 4.3

Click on ‘New’ and paste the url of the Scripts folder that you copied earlier. Click on OK as shown in fig1.4

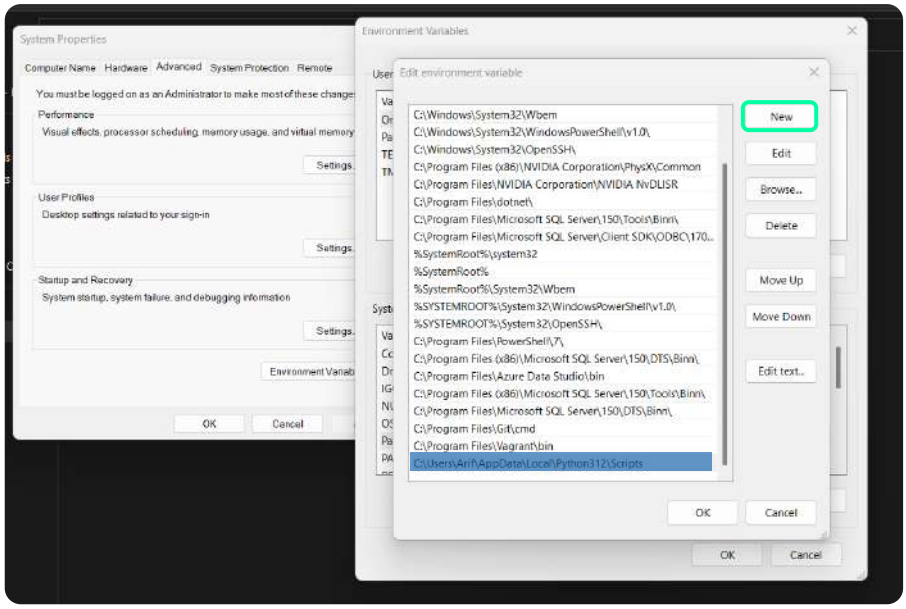

Fig 4.4

Set-Up

Now open the Command Prompt and run the following command:

```
pip install flask_pymongo flask opencv-python face_recognition numpy base64 datetime passlib bson keras pandas
```

Codes

- Download all the codes developed by us and open store them in a proper format as shown in fig 1.5

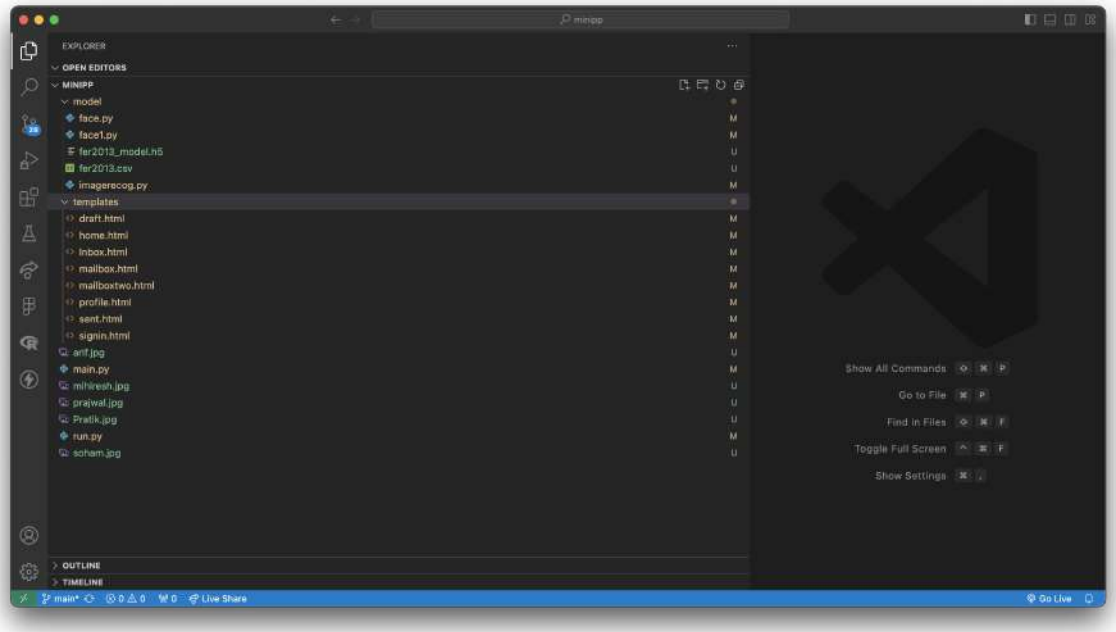

Fig 4.5

Set-Up

Database

You need to create an unstructured database where you can create collections to store the email data and the user data.

For this download MongoDB and MongoDB Compass:

- MongoDB in MacOS:

https://downloads.mongodb.com/compass/mongosh-2.0.2-darwin-arm64.zip

- MongoDB in Windows:

https://downloads.mongodb.com/compass/mongosh-2.0.2-win32-x64.zip

- MongoDB Compass in MacOS:

https://downloads.mongodb.com/compass/mongodb-compass-1.40.4-darwin-x64.dmg

- MongoDB Compass in Windows:

https://downloads.mongodb.com/compass/mongodb-compass-1.40.4-win32-x64.zip

Set-Up

Open MongoDB Compass and click on connect. Create a new database named as “users” and create two collections, name them “emails” which will store data of the content inside mail and a collection named “login” which will store all user info, refer fig 1.6

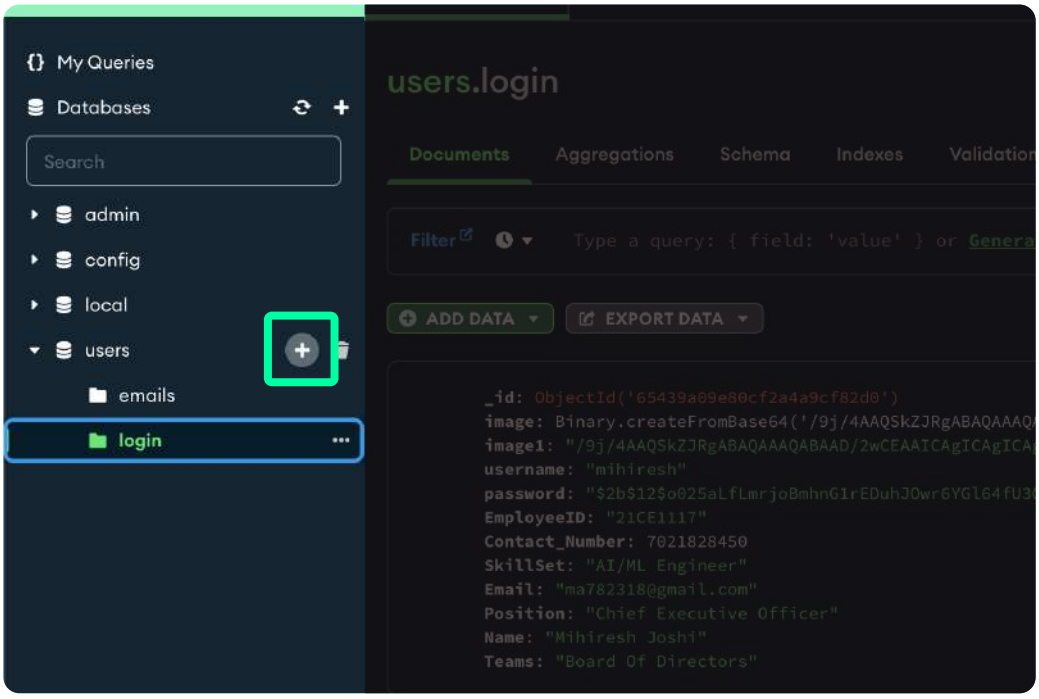

Fig 4.6

Now you can see that there are two collections. The email collection will show all the mails and important details regarding the mail, whereas the login collection will show data regarding user information.

Set-Up

Now using the run.py code enter Employee details in the login collection. In the ‘run.py’ enter your employee credentials and image, also provide the path for that image. This is to add data of your employee in the database. Enter details of your employees in the “image_data” dictionary, refer fig 1.7

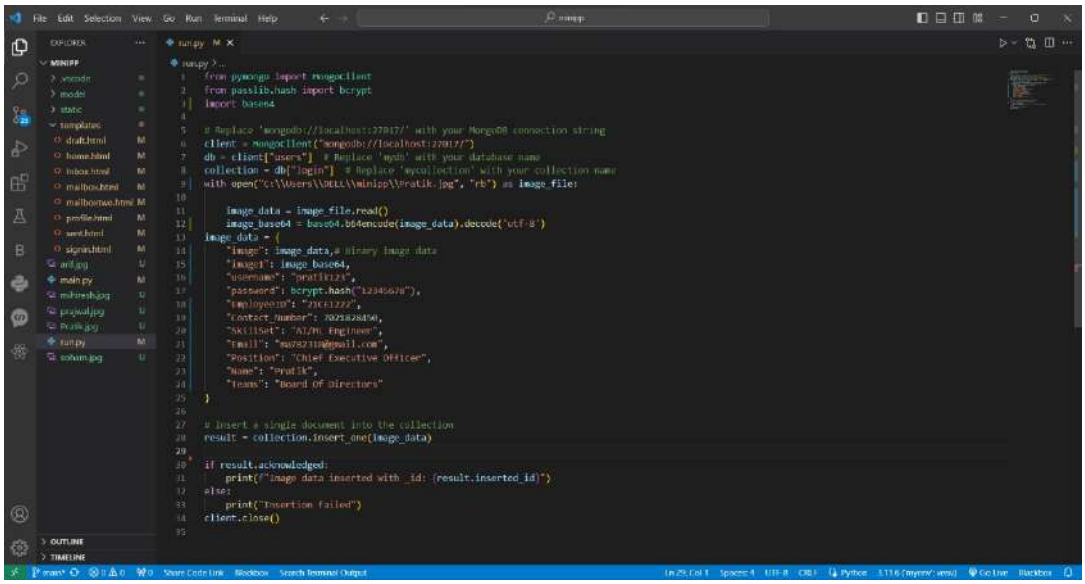

Fig 4.7

After entering details execute the “run.py” using:

```
python run.py
```

Set-Up

Importing and Loading

Following is the code to import libraries, create a URL and load the expression model.

```
from bson import ObjectId
from flask import Flask, make_response, render_template,
request, redirect, url_for, session
from flask_pymongo import PyMongo
from passlib.hash import bcrypt
import face_recognition
import numpy as np
import base64
import datetime
import cv2
from keras.models import load_model
from io import BytesIO
app = Flask(__name__)
app.secret_key = 'arif9353'
app.config['MONGO_URI'] = 'mongodb://localhost:27017/users'
mongo = PyMongo(app)
app.static_folder = 'static'
second_model = load_model('model/fer2013_model.h5')
expressions = ["Angry", "Disgust", "Fear", "Happy", "Sad",
"Surprise", "Neutral"]
```

Set-Up

Route to render HomePage

Following is the code snippet to render login page which asks user to enter his/her ‘username’ and ‘password’. On the basis of which it performs authentication for logging in.

```
@app.route('/')
def home():
return render_template('home.html')

@app.route('/login', methods=['GET', 'POST'])
def login():
if request.method == 'POST':
username = request.form['username']
password = request.form['password']
users_collection = mongo.db.login
user = users_collection.find_one({'username': username})

if user and bcrypt.verify(password, user['password']):
session['username'] = username
print('Login successful.')
url = url_for('profile')
return redirect(url)
else:
print('Login failed. Please check your username and password.')

return render_template('signin.html')
```

Set-Up

Flask Routes

The Flask code includes three routes: /profile retrieves and displays user information and images from a MongoDB collection, /logout clears the session and redirects to the login page, and /draft renders a template for a draft page.

```
app.route('/profile')
def profile():
data = session['username']
items_collection = mongo.db.login
items = list(items_collection.find({'username':data}))
images = []
for item in items:
if 'image' in item:
image_data = item['image']
base64_image = base64.b64encode(image_data).decode('utf-8')
images.append(base64_image)
print("The username from the URL is:",data)
return render_template('profile.html',items=items,
images=images)
@app.route('/logout')
def logout():
session.clear()
print('You have been logged out.')
return redirect(url_for('login'))
@app.route('/draft', methods=['GET', 'POST'])
def draft():
return render_template('draft.html')
```

Set-Up

Flask Email Handling

The Flask route /send_email handles a POST request, retrieves user data, processes an email form, encodes attachments as base64, and stores the email details in the session for further verification, redirecting to a verification route. If the request method is GET, it redirects to the draft page.

```
@app.route('/send_email', methods=['GET', 'POST'])
def send_email():
if request.method == 'POST':
username = session.get('username')
items_collection = mongo.db.emails
arif = None
if request.form.get("confidential") == "on":
arif = "yes"
else:
arif = "no"
session['confi'] = arif
attachment = request.files["attachment"]
if attachment:
# Read the file content and encode it as base64
attachment_data =
base64.b64encode(attachment.read()).decode('utf-8')
else:
attachment_data = None
```

Set-Up

Flask Email Handling (cont.)

```
data = {
"to": request.form.get("to"),
"title": request.form.get("title"),
"message": request.form.get("message"),
"confidential": arif,
"from": username,
"date_time": datetime.datetime.now(),
"attachment": attachment_data
}
xyz = "arif"
result = xyz + username
session[result] = data
return redirect(url_for("first_verification"))
return redirect(url_for("draft"))
```

Set-Up

First Verification Process

The /first_verification route uses facial recognition to compare a live webcam feed with a stored facial image associated with the user. If a match is found, the user is considered verified, and the session is updated accordingly. The route redirects to the next verification step. If verification fails or an error occurs, it redirects to the draft page.

```
@app.route('/first_verification')
def first_verification():
try:
username = session.get('username')
verified = False
items_collection = mongo.db.login
document = items_collection.find_one({"username": username})
base64_data = document["image1"]
image_data = base64.b64decode(base64_data)
known_image =
face_recognition.load_image_file(BytesIO(image_data))
known_face_encoding =
face_recognition.face_encodings(known_image)[0]
cap = cv2.VideoCapture(0)
while True:
ret, frame = cap.read()
face_locations = face_recognition.face_locations(frame)
```

Set-Up

First Verification Process (cont.)

```
for face_location in face_locations:
face_encoding = face_recognition.face_encodings(frame,
[face_location])[0]
match = face_recognition.compare_faces([known_face_encoding],
face_encoding)
name = "Unknown"
if match[0]:
name = "Known Person"
verified = True
top, right, bottom, left = face_location
cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255),
2)
cv2.putText(frame, name, (left + 6, bottom - 6),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
cv2.imshow("Live Facial Recognition", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
if verified:
session['first_verification'] = True
print("First verification successful")
return redirect(url_for('second_verification'))
else:
print("First verification failed.")
except Exception as e:
print(f"An error occurred: {str(e)}")
return redirect(url_for('draft'))
```

Set-Up

Second Verification Process

The /second_verification route captures a live webcam feed, detects facial expressions using a trained model, and requires the user to display at least three distinct facial expressions for verification. If successful, the route redirects to the next step ('insertion'). If not verified, the user is prompted to continue the second verification process.

```
@app.route('/second_verification')
def second_verification():
cap = cv2.VideoCapture(0)
verified = False
unique_emotions = set()
while True:
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5, minSize=(30, 30))
for (x, y, w, h) in faces:
face_roi = gray[y:y + h, x:x + w]
face = cv2.resize(face_roi, (48, 48))
face = np.expand_dims(face, axis=0)
```

Set-Up

Second Verification Process (cont.)

```
face = face / 255.0
emotion = second_model.predict(face)
emotion_label = expressions[np.argmax(emotion)]
unique_emotions.add(emotion_label)
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.putText(frame, emotion_label, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

cv2.imshow('Facial Expression Recognition', frame)

if cv2.waitKey(1) == ord('q'):
break

if len(unique_emotions) >= 3:
verified = True
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
if not verified:
print('Second verification required.')
return redirect(url_for('second_verification'))
else:
return redirect(url_for('insertion'))
```

Set-Up

Email Insertion

The /insertion route retrieves email details from the session, attempts to insert them into a MongoDB collection, and prints a success message if the insertion is successful. If there's an error or the data is not in the correct format, it redirects to the draft page.

```
@app.route('/insertion')
def insertion():
xyz = "arif"
username = session.get('username')
result = xyz + username
data = session.get(result)
try:
if data and isinstance(data, dict):
items_collection = mongo.db.emails
items_collection.insert_one(data)
print('Email details inserted successfully.')
return render_template("draft.html")
else:
print('Data is not in the correct format for insertion.')
return redirect(url_for('draft'))
except Exception as e:
print(f'An error occurred during email insertion: {str(e)}')
return redirect(url_for('draft'))
```

Set-Up

Email Inbox

The /inbox route shows emails for the logged-in user. /mailbox/<email_id> validates and displays an email, redirecting for confidential emails or if not found.

```
@app.route('/inbox')
def inbox():
username = session.get('username')
items_collection = mongo.db.emails
emails = list(items_collection.find({'to': username}))
return render_template('Inbox.html', emails=emails)
@app.route('/mailbox/<email_id>', methods=['GET'])
def mailbox(email_id):
username = session.get('username')
items_collection = mongo.db.emails
email = items_collection.find_one({'_id': ObjectId(email_id),
'to': username})
arif = items_collection.find_one({'_id': ObjectId(email_id),
'confidential': "yes"})
email['_id'] = str(email['_id'])
session['email'] = email
if arif:
return redirect(url_for('firstverify_inbox'))
else:
if email:
return render_template('mailbox.html', email=email)
else:
print('Email not found.')
return redirect(url_for('inbox'))
```

Set-Up

First Verification For Inbox

The /firstverify_inbox route initiates facial recognition for inbox access, comparing a live webcam feed with a stored facial image associated with the user. If verified, it sets a session flag and proceeds to the next verification step (secondverify_inbox). If verification fails or an error occurs, it redirects to the inbox page.

```
@app.route('/firstverify_inbox')
def firstverify_inbox():
try:
username = session.get('username')
verified = False
items_collection = mongo.db.login
document = items_collection.find_one({"username": username})
base64_data = document["image1"]
image_data = base64.b64decode(base64_data)
known_image =
face_recognition.load_image_file(BytesIO(image_data))
known_face_encoding =
face_recognition.face_encodings(known_image)[0]
cap = cv2.VideoCapture(0)
while True:
ret, frame = cap.read()
face_locations = face_recognition.face_locations(frame)
```

Set-Up

First Verification For Inbox (cont.)

```
for face_location in face_locations:
face_encoding = face_recognition.face_encodings(frame,
[face_location])[0]
match = face_recognition.compare_faces([known_face_encoding],
face_encoding)
name = "Unknown"
if match[0]:
name = "Known Person"
verified = True
top, right, bottom, left = face_location
cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255),
2)
cv2.putText(frame, name, (left + 6, bottom - 6),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)
cv2.imshow("Live Facial Recognition", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
if verified:
session['firstverify_inbox'] = True
print("First verification successful")
return redirect(url_for('secondverify_inbox'))
else:
print("First verification failed.")
except Exception as e:
print(f"An error occurred: {str(e)}")
return render_template("inbox.html")
```

Set-Up

Second Verification For Inbox

The /secondverify_inbox route captures live facial expressions using a webcam, requiring the user to display at least three distinct emotions for verification. If successful, it renders the mailbox template with the email content; otherwise, it prompts for further verification.

```
@app.route('/secondverify_inbox')
def secondverify_inbox():
cap = cv2.VideoCapture(0)
verified = False
unique_emotions = set()
while True:
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5, minSize=(30, 30))
for (x, y, w, h) in faces:
face_roi = gray[y:y + h, x:x + w]
face = cv2.resize(face_roi, (48, 48))
face = np.expand_dims(face, axis=0)
face = face / 255.0
```

Set-Up

Second Verification For Inbox (cont.)

```
emotion = second_model.predict(face)
emotion_label = expressions[np.argmax(emotion)]
unique_emotions.add(emotion_label)
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.putText(frame, emotion_label, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

cv2.imshow('Facial Expression Recognition', frame)

if cv2.waitKey(1) == ord('q'):
break

# Assume verification is successful if three different
expressions are detected
if len(unique_emotions) >= 3:
verified = True
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
email = session.get('email')
if not verified:
print('Second verification required.')
return redirect(url_for('secondverify_inbox'))
else:
return render_template('mailbox.html', email=email)
```

Set-Up

Sent Emails

The '/sent' route shows emails sent by the user. '/mailboxtwo/<email_id>' validates and displays sent emails, redirecting for confidential ones or if not found.

```
@app.route('/sent')
def sent():
username = session.get('username')
items_collection = mongo.db.emails
emails = list(items_collection.find({'from':username}))
return render_template('sent.html', emails=emails)
@app.route('/mailboxtwo/<email_id>', methods=['GET'])
def mailboxtwo(email_id):
username = session.get('username')
items_collection = mongo.db.emails
email = items_collection.find_one({'_id': ObjectId(email_id),
'from': username})
arif = items_collection.find_one({'_id': ObjectId(email_id),
'confidential': "yes"})
email['_id'] = str(email['_id'])
session['emailll'] = email
if arif:
return redirect(url_for('firstverify_sent'))
else:
if email:
return render_template('mailboxtwo.html', email=email)
else:
print('Email not found.')
return redirect(url_for('sent'))
```

Set-Up

First Sent Email Verification

The '/firstverify_sent' route conducts facial verification for the logged-in user using live webcam feed, comparing it with a stored facial image. If successful, it sets a verification flag and redirects to 'secondverify_sent.' If not, it prints a failure message and returns to the 'sent' page.

```
@app.route('/firstverify_sent')
def firstverify_sent():
try:
username = session.get('username')
verified = False # Flag to track if verification is successful
items_collection = mongo.db.login
document = items_collection.find_one({"username": username})
base64_data = document["image1"]
image_data = base64.b64decode(base64_data)
known_image =
face_recognition.load_image_file(BytesIO(image_data))
known_face_encoding =
face_recognition.face_encodings(known_image)[0]
cap = cv2.VideoCapture(0)
while True:
ret, frame = cap.read()
face_locations = face_recognition.face_locations(frame)
for face_location in face_locations:
face_encoding = face_recognition.face_encodings(frame,
[face_location])[0]
```

Set-Up

First Sent Email Verification (cont.)

```
match = face_recognition.compare_faces([known_face_encoding],
face_encoding)
name = "Unknown"
if match[0]:
name = "Known Person"
verified = True
top, right, bottom, left = face_location
cv2.rectangle(frame, (left, top), (right, bottom), (255, 0, 0),
2)
cv2.putText(frame, name, (left + 6, bottom - 6),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (255 , 0, 0), 1)

cv2.imshow("Live Facial Recognition", frame)
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
if verified:
session['firstverify_sent'] = True
print("First verification successful")
return redirect(url_for('secondverify_sent'))
else:
print("First verification failed.")
except Exception as e:
print(f"An error occurred: {str(e)}")
return render_template("sent.html")
```

Set-Up

Second Sent Email Verification

The '/secondverify_sent' route captures live facial expressions, requiring the user to display at least three distinct emotions for verification. If successful, it renders the 'mailboxtwo.html' template with the email content; otherwise, it prompts for further verification.

```
@app.route('/secondverify_sent')
def secondverify_sent():
cap = cv2.VideoCapture(0)
verified = False # Flag to track if verification is successful
unique_emotions = set()
while True:
ret, frame = cap.read()
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5, minSize=(30, 30))
for (x, y, w, h) in faces:
face_roi = gray[y:y + h, x:x + w]
face = cv2.resize(face_roi, (48, 48))
face = np.expand_dims(face, axis=0)
face = face / 255.0
emotion = second_model.predict(face)
emotion_label = expressions[np.argmax(emotion)]
unique_emotions.add(emotion_label)
```

Set-Up

Second Sent Email Verification (cont.)

```
cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.putText(frame, emotion_label, (x, y - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

cv2.imshow('Facial Expression Recognition', frame)

if cv2.waitKey(1) == ord('q'):
break

# Assume verification is successful if three different
expressions are detected
if len(unique_emotions) >= 3:
verified = True
if cv2.waitKey(1) & 0xFF == ord("q"):
break
cap.release()
cv2.destroyAllWindows()
email = session.get('emailll')
if not verified:
print('Second verification required.')
return redirect(url_for('secondverify_sent'))
else:
return render_template('mailboxtwo.html', email=email)
```

Set-Up

Attachment Download

The '/download_attachment/<attachment_id>' route retrieves an attachment from MongoDB by its ID, decodes the base64 data, and sends it as an octet-stream for download. If the attachment is not found, it redirects to the inbox page.

```
@app.route('/download_attachment/<attachment_id>',
methods=['GET'])
def download_attachment(attachment_id):
attachment_collection = mongo.db.emails
attachment = attachment_collection.find_one({'_id':
ObjectId(attachment_id)})

if attachment:
attachment_data = attachment['attachment']
decoded_data = base64.b64decode(attachment_data)

response = make_response(decoded_data)
response.headers['Content-Type'] = 'application/octet-stream'
return response
else:
print('Attachment not found.')
return redirect(url_for('inbox'))

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

Set-Up

Start the Software

To start the software in Terminal go in the directory which contains the “main.py” file and then enter the following command:

```
python main.py
```

You will get an ‘http’ link of local host, copy-paste that link on your browser and the software will start running.

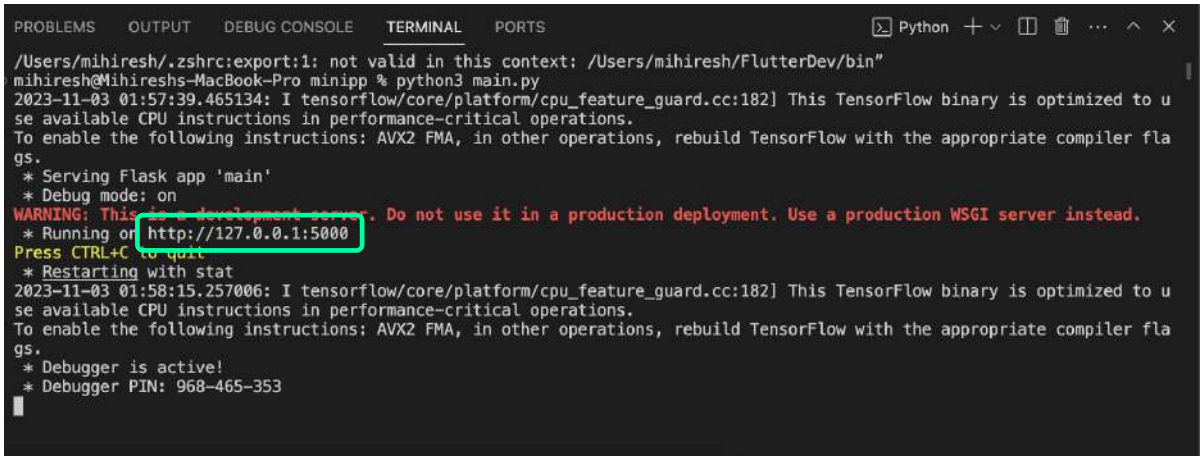

Fig 4.8

Note: Before starting make sure all listed libraries are downloaded and the files are stored in the exact format as mentioned in fig 1.5 and keep the MongoDB Compass active

Workflow

Login

The software will begin with a welcoming page showcasing company logo and key features shown in fig 2.1

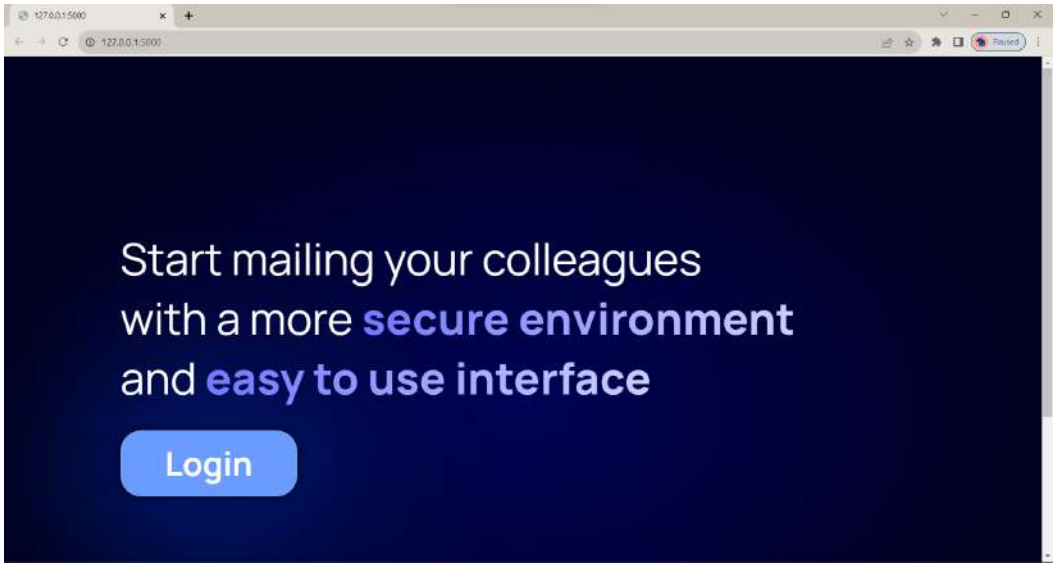

Fig 5.1

Click on the “Login” button to move onto login form.

Workflow

Fill the details which your company admin has entered in the database, refer fig 2.2

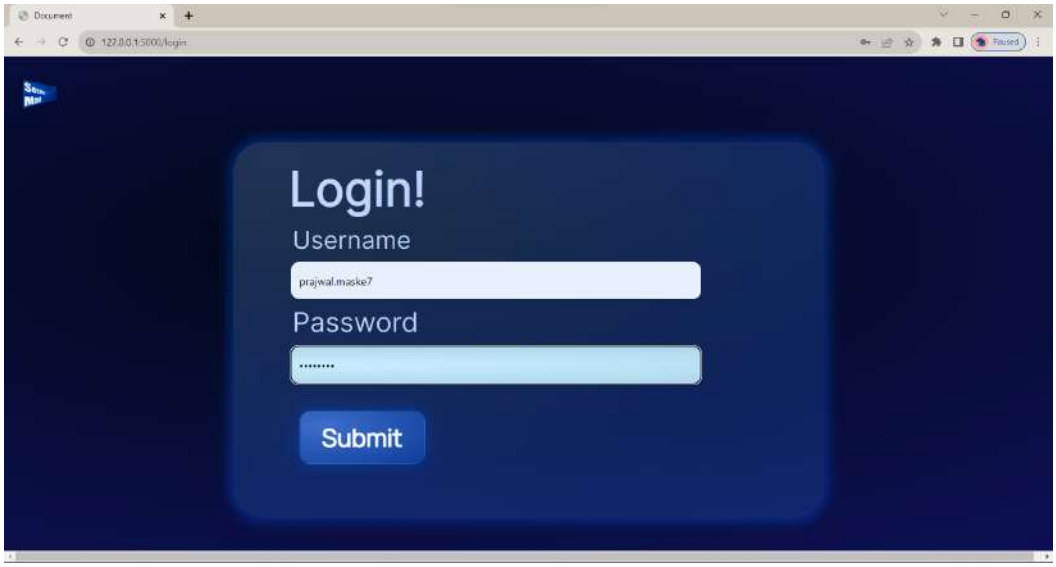

Fig 5.2

After entering all your details click on “Login” to log into the software.

Workflow

The first page which renders after logging in is the profile page in which you get an overview of your profile. Your name, position, photo, contact details, teams and your skill set is shown here.

Fig 5.3

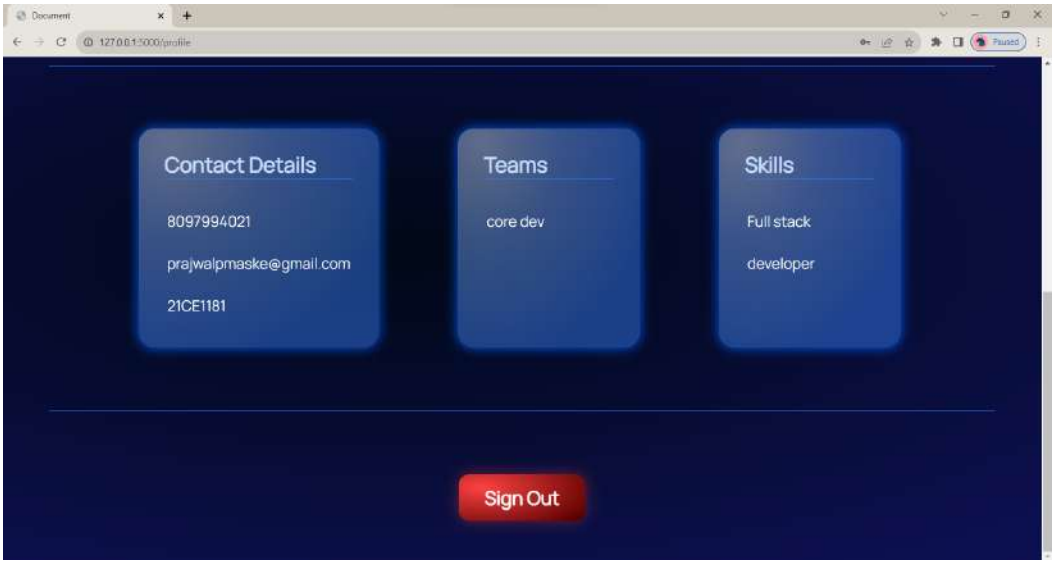

Fig 5.4

Workflow

Draft

To Draft a new mail go in the draft tab from nav bar and fill all the input fields as shown in fig 2.5

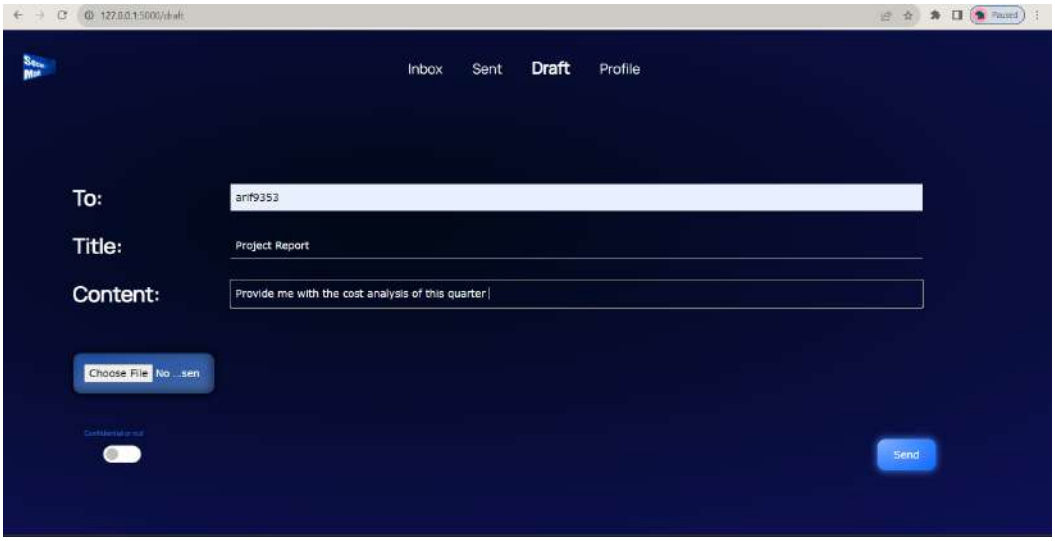

Fig 5.5

The switch button defines whether the receiver needs to verify himself before opening the mail or not.

The “To” consists the username to whom you wish to send the mail, “Title” consists of the title of the mail and the “Content” consists of the main message. You may also add Attachments to your mail using the “Attachments” button.

Workflow

When you click on thee send button the api will start the first verification in which it detects whether its you or someone Unknown.

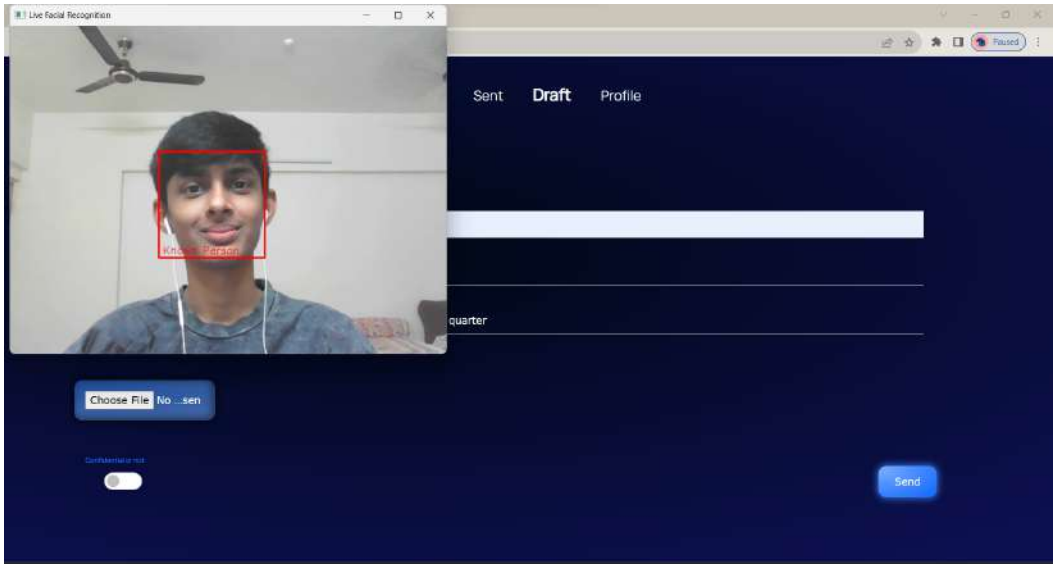

Fig 5.6

The results of the first verification will be displayed as a label to the box which detects your face.

If successful (i.e. “Known”) then press ‘q’ on your keyboard, it will stop the first verification and move onto the second verification. Although if the face is not matched (i.e. Unknown) then pressing ‘q’ will terminate the verification process and the mail won’t be sent.

Workflow

After the first verification is successful, the second verification immediately starts where the user has to perform any three expressions. As shown in fig 2.7

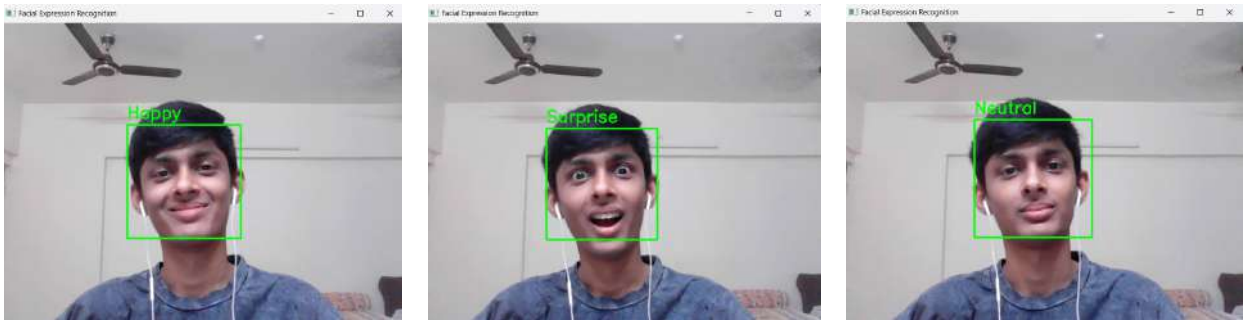

Fig 5.7

After performing three different expression again click ‘q’ on your keyboard. If your first verification is successful then on clicking ‘q’ you will be redirected to the second expression verification, after successful verification of the expression test then clicking ‘q’ will authenticate you . and your mail will be sent.

Workflow

Open Mail

To open any mail that you sent, you need to authenticate yourself in the same manner as mentioned earlier. The authentication process starts as soon as you click on your desired mail from Inbox mail list.

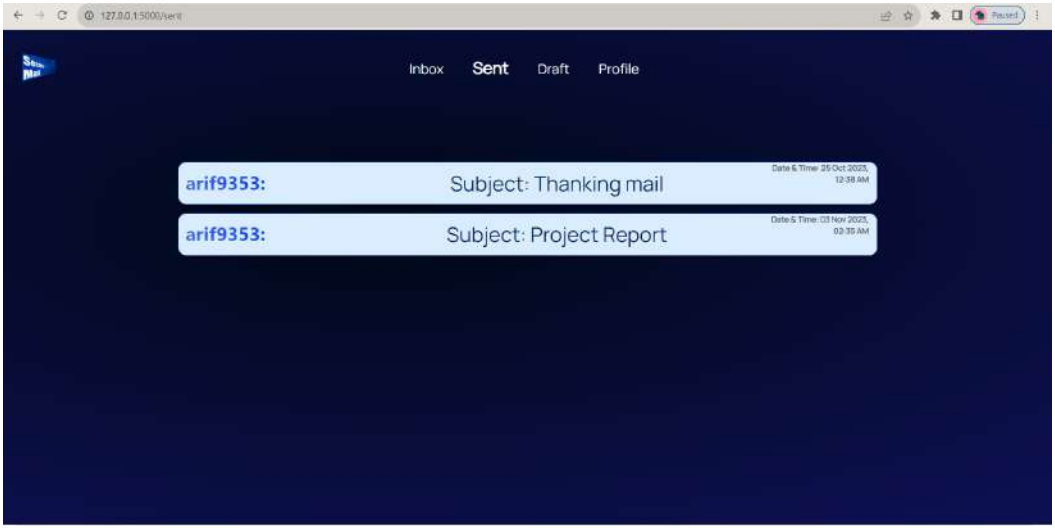

Fig 5.8

Whereas to open mail you received, you might not always require to authenticate yourself. Some mails might directly open which are not secured by the user when sending while some mails would authenticate you first before opening, that follow the same process of authentication as discussed earlier.

Workflow

When the mail opens you see all the details like who sent it, the Title of the mail, the main content and you can download the attachments sent inside the mail.

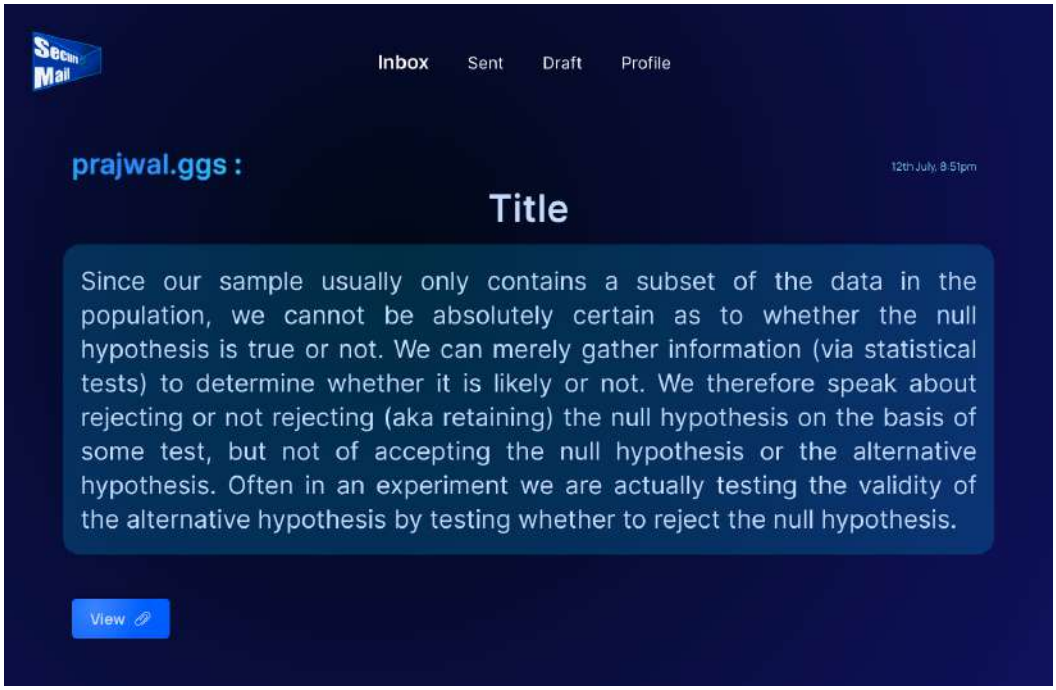

Fig 5.9

In above fig 2.9, you can see the interface you will see when you open the mail.

Results & Discussion

Conclusion

We developed a software where admin could create a detailed profile of yours and you are able to share mail with attachments in a secure environment.

Future Scope

In future, we could add a manage-mail system where you could sort and arrange your mails as you wish. Also we are thinking to add a feature 'Teams' which enables collaborative communication.

NOTE

While running software:

- Make sure Python and all libraries mentioned are properly downloaded
- MongoDB database is created with mentioned connections.
- The data of employees is inserted by admin with a proper photo as that photo will be used for Face Detection.
- MongoDB database is handled properly as it consists of all this sensitive data.

Acknowledgments

We would like to thank our **Honourable Principal, Dr. Mukesh D. Patil** sir, and **H.O.D, Dr. Amarsinh V. Vidhate** sir, of ***"Ramrao Adik Institute of Technology, Navi Mumbai"***, for providing us with this amazing environment and opportunity to implement this project.

Guides:

- **Dr. Puja Padiya**
- **Mrs. Shweta Ashthekar**

Developed by:

- **Mihiresh Joshi**

LinkedIn: https://www.linkedin.com/in/mihiresh-joshi-651423207/
GitHub: https://github.com/mihireshjoshi

- **Mohammed Arif**

LinkedIn: https://www.linkedin.com/in/mohammed-arif-288221225/
GitHub: https://github.com/arif9353

- **Soham Kulkarni**

LinkedIn: https://www.linkedin.com/in/soham-kulkarni-1b5222202/
GitHub: https://github.com/sohamk63

- **Prajwal Maske**

LinkedIn: https://www.linkedin.com/in/prajwal-maske-9166b0223/
GitHub: https://github.com/prajwalpmaske