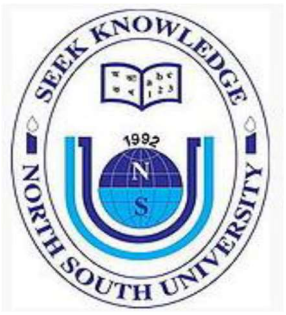
CSE 299: JUNIOR DESIGN

# CAMERA SURVEILLANCE SYSTEM

# JUBA


## Final Report




# North South University

## CSE 299: Junior Design Project

### Section: 09

## Final report

**Project Title:** Camera Surveillance System

**Submitted By:**

- Tahiat Hakim Himel [2013075042]
- Zobaer Ahammod Zamil [2021796042]

**Submitted To:**

Muhammad Shafayat Oshman

Lecturer, Department of Electrical & Computer Engineering

North South University

# Table of Contents

## Introduction

The Juba Camera Surveillance is a comprehensive endeavor to develop an advanced and efficient system for monitoring and managing surveillance cameras. In today's increasingly complex security landscape, robust surveillance solutions have become paramount. This project focuses on designing and implementing a sophisticated software application that leverages cutting-edge technologies to enable seamless integration, real-time monitoring, and centralized control of a network of surveillance cameras.

Our primary target was to prepare a system for North South University through which any unauthorized person entering the university premises was identified. This university campus is open to students and university staff only. It is only possible for someone directly affiliated with the university to enter the university premises. But still, it is seen that sometimes people enter the university without permission. Unauthorized persons entering the University may cause security disturbances. In addition, it may be necessary to identify a student or staff of the university without informing them. The primary objective of our project is to detect people through face detection and display their information.

This report provides a detailed overview of the project's objectives, methodology, system architecture, implementation details, and evaluation results, highlighting the significant contributions made toward achieving a reliable and scalable camera surveillance software solution.

## Problem Statement

One of the problems of North South University is the entry of non-students into the campus. Many outsiders enter the university grounds, especially around a special event. As a result of which, it can be seen that the crowd on the campus is excessive. University students cannot enjoy the program properly due to the large group. Moreover, the lecture is disturbed because of the noise from extra people. Sometimes outsiders misbehave with university students—especially misbehaving with girls. Since North South University is private, it must undergo stringent security measures. Again, one university student is often seen teasing another student. But the victim could not complain due to not knowing the name or ID number of the harasser.

Our project will solve these problems.

- No outsiders shall enter the University premises without permission on any occasion or at any time.
- If a university student commits a crime, he should be immediately concerned.

## Project description

***Frontend:***

We used HTML, CSS, and JavaScript for the front end of our project. HTML provides the structure and content of web pages. Key components of the HTML development phase include: Defining the website's overall design using appropriate HTML tags. Incorporating multimedia elements such as images, videos, and audio enriches the user experience. We are establishing links between web pages to enable navigation within the site and setting up forms to collect user input or allow user interactions.

CSS is responsible for the visual presentation of the website. It allows developers to customize the layout, typography, colors, and other visual aspects. The CSS development phase involves: Creating a consistent and visually appealing design by defining styles for various HTML elements. Applying responsive design techniques ensures the website adapts to different screen sizes and devices. They optimize the website for faster loading times by minimizing file sizes and utilizing CSS preprocessors if desired.

JavaScript adds interactivity and dynamic behavior to the website, allowing users to engage with the content. JavaScript development involves: Manipulating HTML elements and their properties to respond to user actions or trigger events, validating user input in forms, providing real-time feedback, and implementing animations and transitions to enhance the user experience.

The navigation bar is created within the wrapper class. We added hover using CSS and HTML to create admin login and sign-up pages. Both CSS and JavaScript have been used to design these two pages.

In the case of registration, first, a form is created, and all the information is taken in it. The form is inside the container class. In the case of forms, the action tag is used to send the form inputs to the database. Photo registration is done after filling out the form. We have used JavaScript on the registration page so that the form cannot be submitted until every input box is filled. A notification will be given if a user forgets or intentionally does not fill an input field. The Surveillance page displays test results. For information update, information is first searched by NSU ID, then information is updated from that result. The data is updated in the database as soon as the information is updated.

***Backend:***

We use ***Python, Python Flask, and MongoDB*** (an online server) in the backend.

```
# ----------------------- Import Libraries and Packages -----------------------
import os
import pickle
import time
import cv2
import face_recognition
import tkmessagebox
from flask import Flask, render_template, request, Response
from imutils import paths
from pymongo import MongoClient

```

These are the libraries we used in our code.

```
# ----------------------- Connect to MongoDB Server & Create FLASK app -----------------------
# Flask App Create
app = Flask(__name__, template_folder='templates', static_folder='static')

# MongoDB Database Connection
mongodb_uri = "mongodb+srv://zamilza51:zamilza51@cluster0.msilet9.mongodb.net/?retryWrites=true&w=majority"
client = MongoClient(mongodb_uri)
# Database Create
db = client["Security_Surveillance"]
# Collections Create
log_sign = db["Login_Signup"]
register = db["Face_Info_Registration"]

cfp = os.path.dirname(cv2.__file__) + "/data/haarcascade_frontalface_alt2.xml"
face_cascade = cv2.CascadeClassifier(cfp)

```

Then we create a ***Flask App*** and connection with our online database ***"Security_Surveillance."*** This database has multiple collections: ***Login_Signup***, and ***Face_Info_Registration***.

```
# Home Page After LogIn
@app.route('/Home')
def home_after_login_page():
    video.release()
    return render_template('Home.html')

# LogIn Page
@app.route('/Login')
def login_page():
    return render_template('Login.html')
```

After setting up the connections, we connect all our HTML webpages to our backend following similar lines where ***@app.route()*** determines the route of the page, and the functions will return the corresponding pages, such as *Home.html*, *Login.html*, etc.

```
@app.route('/login_form', methods=['POST', 'GET'])
```

After connecting the pages, we build the functionality codes for each page. In *@app.route()*, we use *methods=[‘POST,’ ‘GET’]*, which helps interact with a backend frontend and transfer the forms information to the backend.

**Signup:**

```
97  # User SignUp Form
98  @app.route('/signup_form', methods=['POST', 'GET'])
99  def signup_form():
100     if request.method == 'POST':
101         ins_name = request.form['Institute_Name']
102         email = request.form['Email']
103         username = request.form['Username']
104         pwd = request.form['Password']
105         cn_pwd = request.form['Confirm_Password']
```

*POST method* is used to send all data of frontend forms to the backend.

```
106
107         if log_sign.find_one({"Username": username}) is None:
108             if pwd == cn_pwd:
109                 # valid and Insert
110                 UserDocument = {
111                     "Institute_Name": ins_name,
112                     "Email": email,
113                     "Username": username,
114                     "Password": pwd
115                 }
116                 log_sign.insert_one(UserDocument)
117                 return render_template('login.html')
```

Firstly, the program checks that the username does not exist in the database, then it checks that the *“password”* and the *“confirm password”* is equal. After satisfying these conditions, it enters new signup information into the designed database.

```
118             else:
119                 return render_template('SignUp.html',
120                                        info="Enter Same Password Twice !")  # Not Same Password and enter same password
121         else:
122             return render_template('SignUp.html',
123                                    info="User Name Exist ! Try another.")  # Invalid User and try new username
124
```

If an error occurs, then it will send a corresponding error message to the *signup* page.

**Login:**

```python
# User LogIn Form
@app.route('/login_form', methods=['POST', 'GET'])
def login_form():
    if request.method == 'POST':
        username = request.form['Username']
        pwd = request.form['Password']

        if log_sign.find_one({"Username": username}) is None:
            return render_template('login.html', info='Invalid User')   # Invalid User
        else:
            if log_sign.find_one({"Password": pwd}) is None:
                return render_template('login.html', info='Invalid Password')   # Invalid Password
            else:
                return render_template('Home.html', info=username)
```

Here we check whether the input for the username and corresponding password matches our database. If it satisfies, the program permits a user to log in; otherwise, it will send an error message to the *login* page.

**Registration Form:**

```python
        if register.find_one({"NSU_ID": nsu_id}) is None:
            if register.find_one({"Email": email}) is None:
                if os.path.isdir(f'faces/{nsu_id}') is False:
                    # valid and Insert
                    UserDocument = {
                        "Full_Name": full_name,
                        "NSU_ID": nsu_id,
                        "Department": department,
                        "Email": email,
                        "Contact_Number": contact_num,
                        "Designation": designation
                    }
                    # Insert Data In MongoDB
                    register.insert_one(UserDocument)
                    # Create a directory for the user
                    os.makedirs(f'faces/{nsu_id}')
```

After collecting data using the *POST method*, the program checks if the *NSU_ID, Email*, exists in the database and if there is any directory in the *faces directory* with the given *NSU_ID* in the device.

```python
                    # Register faces
                    total_time = 10
                    start_time = time.time()
                    count = 0
                    while time.time() - start_time < total_time:
                        if count < 20:
                            ret, img = cap.read()
                            cv2.imshow('Capturing image', img)
                            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                            faces = face_cascade.detectMultiScale(gray, 1.2, 5)
                            for (x, y, w, h) in faces:
                                cv2.rectangle(img, (x, y), (x + w, y + h), (0, 0, 255), 2)
                                roi_gray = gray[y:y + h, x:x + w]
                                # roi_color = img[y:y + h, x:x + w]
                                cv2.imwrite(f'faces/{nsu_id}/{count}.jpg', roi_gray)
                                count += 1
                            cv2.waitKey(50)
                    # Release webcam
                    cap.release()
                    cv2.destroyAllWindows()
                    return render_template('Home.html', info=" Successfully Registered. ")
```

Then, the program will capture face images of up to 20 images. After capturing the photos, the window will automatically destroy and send a successfully registered message to the home page, *redirecting to the home page*.

```
196                 else:
197                     return render_template('Registration.html',
198                                            info=" This Person is Already Registered in faces !!")  # person registered
199                 else:
200                     return render_template('Registration.html',
201                                            info=" This Person's Email is Already Registered !!")  # person is registered
202             else:
203                 return render_template('Registration.html',
204                                        info=" This Person's ID is Already Registered in DB !!")  # person is registered
205
206
```

If any error occurs, a corresponding error message will be shown on the *registration* page.

**Encode:**

The program will encode the images with their names from the faces directory and make a ***"face_name_encode"*** file. After successfully encoding, the program will show a popup alert message *"Images Encode Successfully."* It will help reduce the encoding time for each registration and arrange the face mapping and their resembling name.

**Surveillance:**

```
video = cv2.VideoCapture(0)

face_cascade.load(cv2.samples.findFile("static/haarcascade_frontalface_alt2.xml"))
```

Firstly, the program takes the camera access and loads the *'haarcascade_frontalface_alt2.xml'* file. It will help to detect faces.

```
250 data = None
251
252 try:
253     with open('face_name_encode', "rb") as f:
254         data = pickle.load(f)
255 except FileNotFoundError:
256     tkmessagebox.showinfo("Camera Surveillance System", "face_name_encode not found!")
257
```

Then the program checks and opens the *'face_name_encode'* file as f, showing an error message if it doesn't exist.

```
289     while True:
290         ret, frame = video.read()
```

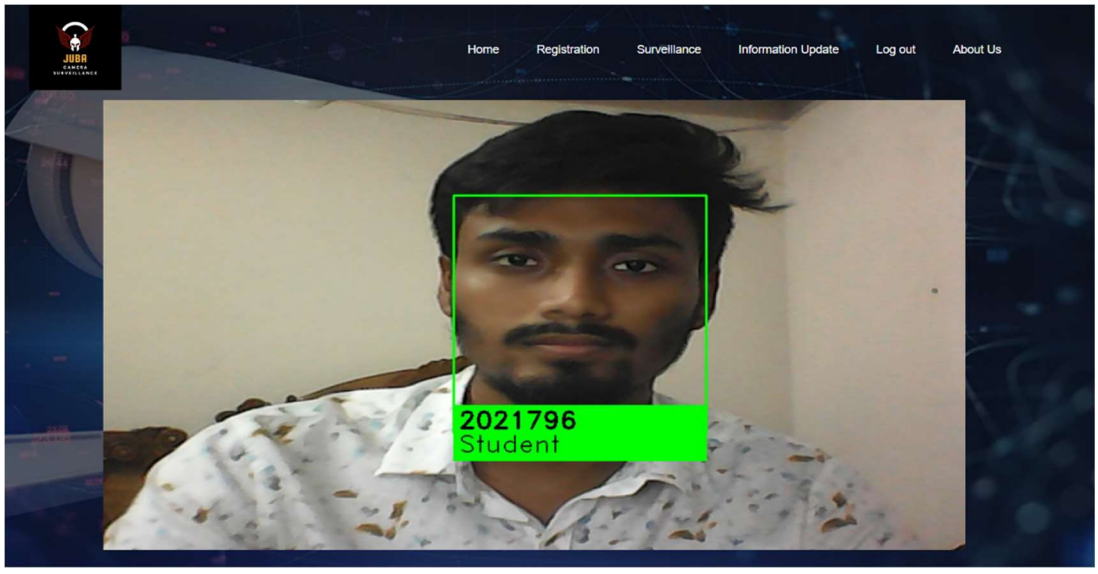
```python
# Recognizing faces
1 usage
def process_frame(frame):
    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
    small_frame = cv2.resize(rgb_frame, (0, 0), fx=0.25, fy=0.25)
    boxes = face_recognition.face_locations(small_frame, model='hog')
    encodings = face_recognition.face_encodings(small_frame, boxes)

    names = []
    for encoding in encodings:
        matches = face_recognition.compare_faces(data["encodings"], encoding)
        name = "Unknown"

        if True in matches:
            matchedIdxs = [i for (i, b) in enumerate(matches) if b]
            count = {}

            for i in matchedIdxs:
                name = data["names"][i]
                count[name] = count.get(name, 0) + 1
```

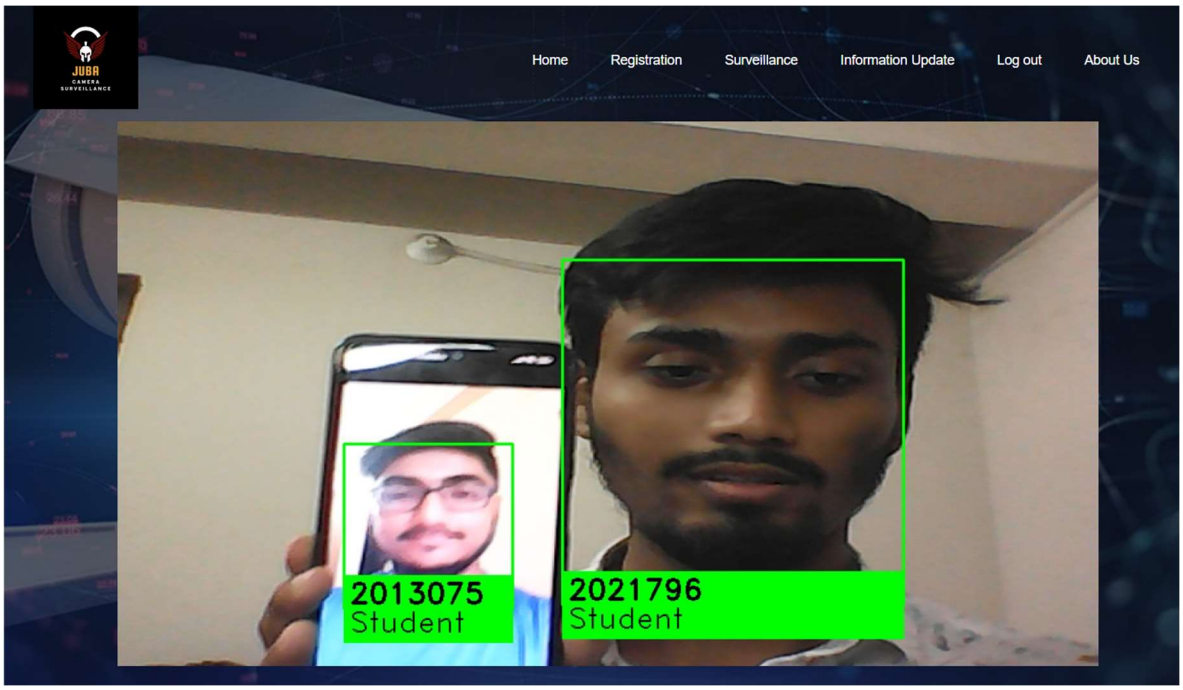
Now, it takes frames using the webcam and processes them. First, it resizes the frame. Then, using the *hog model,* it detects the face location. After that, it encodes the frame, and from that encoding, the program starts matching with the registered faces as *'face_name_encode.'*

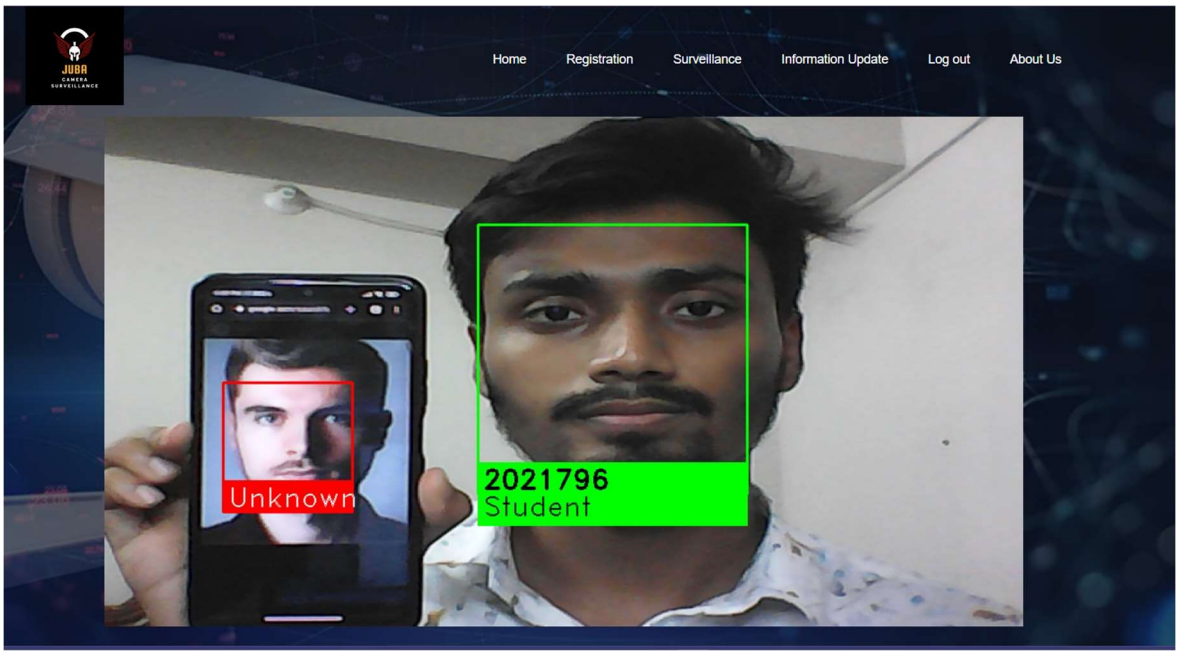
```python
            if register.find_one({"NSU_ID": name}):
                person = register.find_one({"NSU_ID": name})
                cv2.rectangle(resized_frame, (left, top), (right, bottom), (0, 255, 0), 2)
                cv2.rectangle(resized_frame, (left, bottom + 35), (right, bottom - 35), (0, 255, 0), cv2.FILLED)
                font = cv2.FONT_HERSHEY_DUPLEX
                cv2.putText(resized_frame, name, (left + 6, bottom - 6), font, 1.0, (0, 0, 0), 2)
                cv2.putText(resized_frame, person["Designation"], (left + 6, bottom + 24), font, 1.0, (0, 0, 0), 1)
            else:
                output = name
                cv2.rectangle(resized_frame, (left, top), (right, bottom), (0, 0, 255), 2)
                cv2.rectangle(resized_frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
                font = cv2.FONT_HERSHEY_DUPLEX
                cv2.putText(resized_frame, output, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

        ret, jpeg = cv2.imencode('.jpg', resized_frame)
        frame_data = jpeg.tobytes()

        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame_data + b'\r\n\r\n')
```

The matched faces and names display with their corresponding *NSU_ID* and *Designation* on the screen. And if it doesn't find any match, it shows an *unknown* on the screen.

***Results:***






## Update Info:

```
348         # Update Persons Details
349         UpdateUserDocument = {"$set": {
350             "Full_Name": full_name,
351             "NSU_ID": nsu_id,
352             "Department": department,
353             "Email": email,
354             "Contact_Number": contact_num,
355             "Designation": designation
356         }}
```

After collecting the data from the update form, the program sets those values as a document.

```
357         if register.find_one({"NSU_ID": s_nsu_id}) is None:
358             return render_template('Update_Information.html', info=" ID NOT FOUND")  # Not Found
359         else:
360             register.find_one_and_update({"NSU_ID": s_nsu_id}, UpdateUserDocument)
361             return render_template('Update_Information.html',
362                                    info=" Update Personal Details Successful !!")  # Successful Update
363
```

Then, it searches in the database collections, and if it finds a match, it replaces the existing data with new data and sends a success message on the update information page. The update information page shows an error message if no data is in the database.

## Main function:

```
365 # Main Function To Run Flask App
366 if __name__ == '__main__':
367     app.run(debug=True)
368
369 # Close MongoDB Database Connection
370 client.close()
```

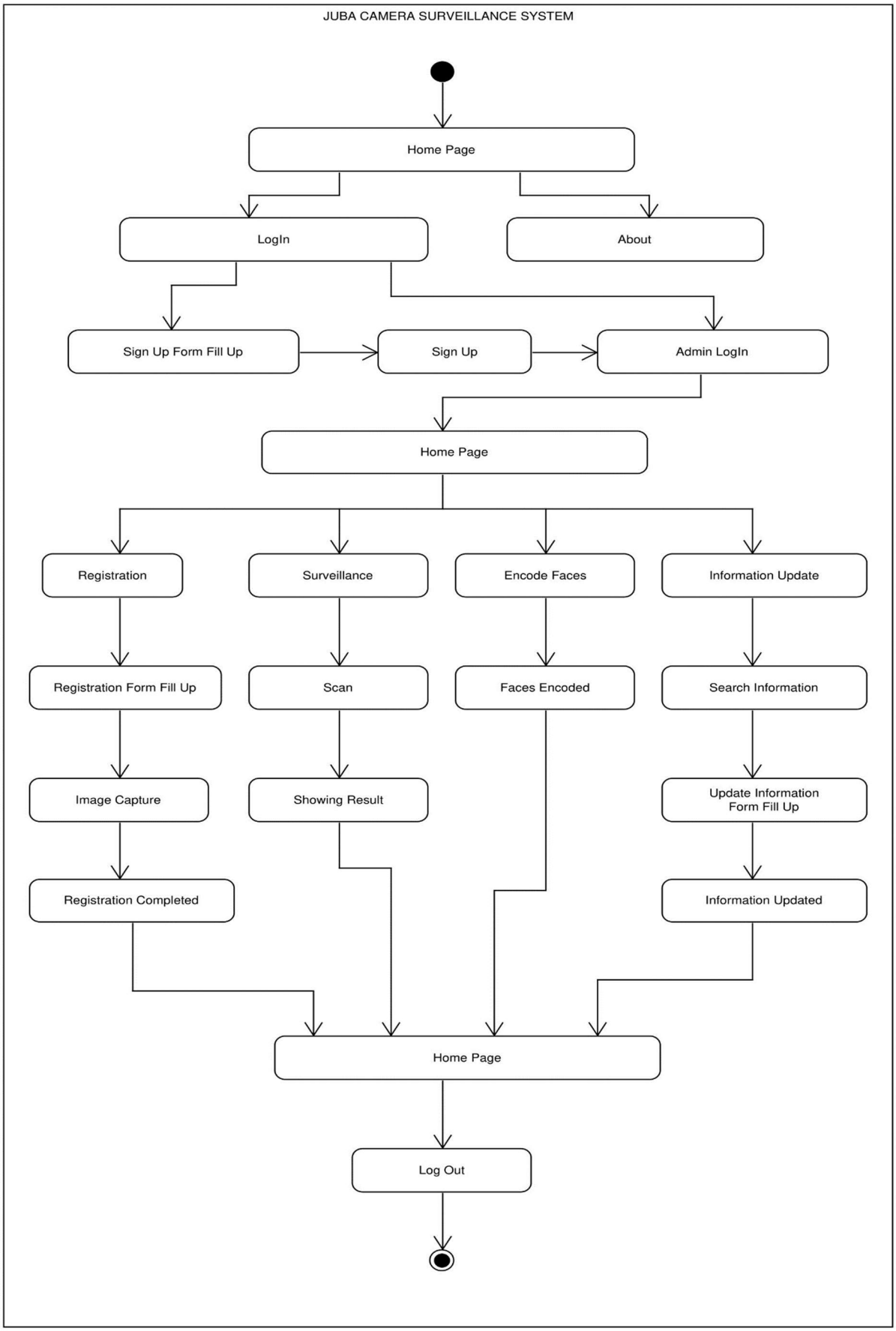
The program file is named *main.py* and will start executing the *Flask App* while enabling debug mode.

The ***__name__*** is a built-in variable in Python that represents the current module's name. When a Python script is executed directly, i.e., it is the main module being run, the value of *__name__* is set to ***'__main__'***. If the script is imported as a module by another hand, the value of *__name__* is set to the module's name.

***App.run()*** is typically used in web frameworks like *Flask* or *Django* to start the web application. It tells the application to start running and listening for incoming *HTTP requests*. In this case, you're using *Flask*, as the *app.run()* is commonly used in *Flask applications*.

Lastly, we close the MongoDB database connection.

# Project Flowchart



## Technologies used

1. PyCharm IDE
2. Python
3. Python - Flask
4. HTML
5. CSS
6. MongoDB (Online storage Database)
7. OpenCV
8. JavaScript

## Cost Analysis

***Total time needed to complete the project:***

Requirement Analysis = 1 month
Development Time = 2 months
Total Time = ***3 months***

***Requirement Analysis Budget:***

Time needed = 1 month
= 22 Working days = **176 hours**

Salary Per Working Hour for required analyst = **BDT 350**

Total expenses for requirement analysis = $BDT\ 350 * 176$
= ***BDT 61,600***

***Development Budget:***

Salary Per Working Hour for developer = BDT 580
Working days in a month = 22 days
Total working hours = $2*22*8\ hours$
= $352\ hours$
Total Salary of Developer = $BDT\ (352 * 580 * 2)$
= ***BDT 4,08,320***

***Rental Budget:***

| | |
|---|---|
| Office space rent per month | = BDT 20,000 |
| Time | = 2 months |
| Rent Cost | = BDT 20,000 * 2 |
| | = BDT 40,000 |
| Utilities Cost | = BDT 10,000 |
| Total Rental cost | = ***BDT 50,000*** |

***Hardware Cost:***

| | |
|---|---|
| Camera | = BDT 1,20,000 |
| Monitor and System | = BDT 80,000 |
| Total Hardware Cost | =***BDT 2,00,000*** |

***Maintenance Budget for One year:***

| | |
|---|---|
| The time needed in a week | = 3 hours |
| Cost per hour | = BDT 1,000 |
| Total time for maintenance | = 3*52 hours |
| | = 156 hours |
| Total Maintenance Cost | = ***BDT 1,56,000*** |

Total Expenses = BDT (61,600+4,08,320+50,000+2,00,000+1,56,000)
= ***BDT 8,75,920***

## Conclusion

To sum up, this program will help stop trespassing on the premises and identify anyone's identity. This cutting-edge technology can help us to deploy this program on a broader scale. Initially, it will stop all extra hassle of trespassing or any unwanted occurrence within the campus. Enough security measure is ensured, as it gathers some personal information. So, this program also provides the security of personal data. The user has to be an authorized user to use this program. No one can look into the online database server directly. Only a few selected people will maintain it initially. After deployment, no one can access it without the creator/administrative permission. So, this program secures personal information as well as reduce unwanted occurrence and solves them if there is any.

## GitHub Repository Link

1. Repository Link:
   https://github.com/z-a-zamil/Camera-Surveillance-System

****** THE END *******