Real Time Movement Tracking System Using Artificial Intelligence
(RTMTSUAI)

By

Muhammad Bilal

Najaf Abbas

Aftab Ahmed khan

Faisal Najeeb

Zaid Yousaf

Supervised by:

Lt Col Khawir Mehmood

Submitted to the faculty of Department of Computer Software Engineering,
Military College of Signals, National University of Sciences and Technology, Islamabad,
in partial fulfillment for the requirements of B.E Degree in Software Engineering.

June 2022

In the name of ALLAH, the Most benevolent, the Most Courteous

CERTIFICATE OF CORRECTNESS AND APPROVAL

This is to officially state that the thesis work contained in this report

"Real Time Movement tracking System using Artificial Intelligence"

is carried out by

Muhammad Bilal

Najaf Abbas

Aftab Ahmed khan

Faisal Najeeb

Zaid Yousaf

under my supervision and that in my judgement, it is fully ample, in scope and excellence, for the degree of Bachelor of Software Engineering in Military College of Signals, National University of Sciences and Technology (NUST), Islamabad.

Approved by

Supervisor

Lt Col Khawir Mehmood

Date: 25 May,2022

DECLARATION OF ORIGINALITY

We hereby declare that no portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

ACKNOWLEDGEMENTS

Allah Subhan'Wa'Tala is the sole guidance in all domains.

Our parents, colleagues and most of all supervisor, Lt Col Khawir Mehmood without your guidance.

The group members, who through all adversities worked steadfastly.

Plagiarism Certificate (Turnitin Report)

This thesis has ____ similarity index. Turnitin report endorsed by Supervisor is attached.

Muhammad Bilal

NUST Serial no

00000278770

Najaf Abbas

NUST Serial no

00000278772

Aftab Ahmed Khan

NUST Serial no

00000278787

Faisal Najeeb

NUST Serial no

00000240967

Zaid Yousaf

NUST Serial no

: 00000209037

_____

Signature of Supervisor

ABSTRACT

In recent years, automated human tracking over camera networks is getting essential for video surveillance. The tasks of tracking human over camera networks are not only inherently challenging due to changing human appearance, but also have enormous potentials for a wide range of practical applications, ranging from security surveillance to retail and health care. This review paper surveys the most widely used techniques and recent advances for human tracking over camera networks.

Table of Contents

Chapter 1: Introduction

Tracking is one of the areas of computer vision that is maturing very rapidly. It allows us to identify and locate objects in an image or video. With this kind of identification and localization, object detection and tracking can be used to count objects in a particular scene and determine and track their precise locations, all while accurately labeling them.

In this project, we have made use of two of the most popular Python libraries for object detection, OpenCV and ImageAI.

Every premises either office or educational institute nowadays has at least one CCTV camera installed. And the data is stored in a centralized repository with a timestamp. Our end goal was to identify the people coming in and going out of the premises and track their movement inside the secure area covered by the camera network,

1.1 Overview

Our project aims for Developing a prototype using artificial intelligence that can automatically track the movement of a person inside a secure area.Using camera that will scan faces and will detect the person face A software that will automatically match the provided face with the captured face and make the movement graph
Scope of the Project:

- Project can be used in schools / colleges / universities. .
- Can be utilized at security sensitive areas/premises.
- Can be utilized in any other government or private institution where movement of a person must be restricted.

1.2 Problem Statement

Everthing can not be done manually lot of work and effort is neeeded to control things manually. So For Manual real time tracking systems

- Manual movement tracking through camera is tedious and time-consuming
- Manual movement tracking is inherently vulnerable to Human errors and erroneous results.
- A lot of time of is needed in Manual movement tracking even in small area.

1.3Proposed Solution

The proposed solution to avoid some tedious and tuime consumig work is to develop an automatic system
We are going to develop an automatic system which will mininmize the manual work

- Developing a prototype using artificial intelligence that can automatically track the movement of a person inside a secure area.
- Using camera that will scan faces and will detect the person face
- A software that will automatically match the provided face with the captured face and make the movement graph

1.4 Working Principle

The project mainly works on the principles of image processing amalgamated with machine learning algorithms. The project is divided into different modulus and every module is inter-woven with the next module. The list of modules is as under:

- Datasets containing images.
- Dataset training and processing
- Output extraction
- Decision based upon Output
- Web based application
- Integration
- GUI based presentation of output

1.4.1 Datasets and annotations:

The integral part of the project is the preparation of datasets. The dataset comprises of images of different persons entering into the campuses,premises. .

1.4.2 Dataset training and processing:

The prepared dataset is used as input to detection models using machine learning.Training is done automatically We are using already trained model.

1.4.2.1 simple face recognition algorithm:

Our Project uses simple face recognition algorithm to detect face inside camera or video. This algorithm finds all the faces that appear in a video Get the locations and outlines of each person's eyes, nose, mouth and chin Recognizes who appears in video by matching the features of the face detected inside the video and already present inside the dataset provided earlier.

1.4.3 Output Extraction:

The outputs are extracted on the basis of a person detected inside a camera , person/persons is/are detected and tracked to keep a record.

1.4.4 Decision based upon Outputs:

The extracted outputs, the detection and tracking of a person in the camera , is used in decision making.

1.4.4.1 Feature matched based decision:

The primary decision is based upon the number of the features matched with any image in the dataset.

.

1.4.5 Integration:

The different modules is then integrated in to one web based application.

1.4.6 GUI presentation:

The visual demonstration of the project is done through the aid of GUI (graphical user interface).

1.4.7 Raspberry-PI:

Raspberry-Pi is a small computer that is helpful in implementing the project physically. This project has a live camera attached with raspberry-pi which process the live feed and produces the desired outputs. [4]

1.5 Objectives

1.5.1 General Objectives:

"To build an innovative state of the art software integrated hardware prototype powered by Machine Learning (ML) and Internet Protocol (IP) techniques, providing a smart administrative tool to automatically track and detect a person's mpvemnet inside a scure area covered by the CCTV cameras."

1.5.2 Academic Objectives:

- Development of a real time movement tracking System.
- To implement Machine Learning and artificiall intelligence techniques to simulate the results
- To increase productivity by working in a team
- To design a project that contributes to the welfare of society

1.6 Scope

- Project can be used in schools / colleges / universities. .
- Can be utilized at security sensitive areas/premises.

- Can be utilized in any other government or private institution where movement of a person must be restricted.

1.7 Deliverables

1.7.1 Web Application:

A web basedc application running on any browser through which a user can login to the account and control the system using different options available on the dashboard like upload image upload video live tracking etc.

1.7.2 Object of interest:

It can detect the object of interest by using the same combination of image processing and machine learning techniques. By detecting the object of interest, we mean detecting the suspected person inside the camera.

1.8 Relevant Sustainable Development Goals

What is the Locally Relevant Socio-Economic Issue that the Project Addresses?
Ususstainable development
Our system focuses on sustainable development and it is developed in the way that features can be added easily anytime it uses already working camera network and also it can be used for long time without any adverse effect on environment

Justify how particular SDG is related to your FYP?
Industrial innovation and infrastructure
As our project tries to innovate previous project already in use in the industry Our basic aim is to improve accuracy of the previous real time movement tracking system and further incorporate some new features

1.9 Structure of Thesis

Chapter 2 contains the literature review and the background and analysis study this thesis is based upon.

Chapter 3 contains the design and development of the project.

Chapter 4 introduces detailed evaluation and analysis of the code.

Chapter 5 contains the conclusion of the project.

Chapter 6 highlights the future work needed to be done for the commercialization of this project.

Chapter 2: Literature Review

A new product is launched by modifying and enhancing the features of previously launched similar products. Literature review is an important step for development of an idea to a new product. Likewise, for the development of a product, and for its replacement, related to real time movement tracking systems , a detailed study regarding all similar projects is compulsory. Our research is divided into the following points.

- Industrial Background
- Existing solutions and their drawbacks
- Research Papers

2.1 Industrial background

In today's era, one of the major issues faced across the world is manually running different systems as there a lot of system working around in any organisation to manual run every system it is very difficult and also much effort and more staff is neede to be hired to run those systems . Manual tracking has alsoled to many further problems, as discussed in Problem Statement, which increases need for a an automated system . Ultimately, results in a big marketplace for industrial development.

Initially, many tracking systems were developed .Some were very successful at that time but some were having accuracy issues .Instituions in Pakistan either educational or anyother used these system to keep track of the terrorists or any suspicious person inside the premises Then, increase in industrial growth due to the rapid expansion of domestic demand need for more such system that are having competitive edge over

existing systems increased. And now industries are inclining towards smart , automatic systems based on new technologies (Internet of Things (IOT), Machine Learning (ML) techniques, Artificial Intelligence (AI)). Hence, Real Time Movement Tracking system provides good market growth and impacts economy directly as it is a fully automated system.

2.2 Existing solutions and their drawbacks

Different solutions are previously being provided for real time face detection ,but every product has some pros and cons. Following are some solutions which are already being prepared and being implemented.

2.2.1:**Automated Video Surveillence System For Human Motion Detection:**

Automated video surveillance systems are most useful and important in the security field of today's world. Video is a base for higher level intelligence applications. Video surveillance is an important security asset to control theft, trespassing or traffic monitoring, banks, department stores, highways, crowded public places and borders. In this thesis, our objective is to design a complete framework able to automatically detect and recognize moving objects in video sequences acquired with a static camera. The aimed practical application for this framework is its use as an automatic intelligent video surveillance system. In video surveillance, detection of moving objects from a video is necessary for object classification, target tracking, activity recognition, as well as behavior understanding. Results are not accurate using this system.

2.2.2**Human Motion Detection and Tracking for Video Surveillance**

The system aims at tracking an object in motion and classifying it as a Human or Non-Human entity, which would help in subsequent human activity analysis. The system employs a novel combination of an Adaptive Background Modeling Algorithm (based on the Gaussian Mixture Model) and a Human Detection for Surveillance (HDS) System. The HDS system incorporates a Histogram of Oriented Gradients based human detector which is well known for its performance in detecting humans in still images. Detailed analysis is

carried out on the performance of the system on various test videos. Cannot detect a person from multiple video streams

2.2.3Detection And Tracking Of Moving Object In Visual Surveillance System

Moving object detection and tracking of these moving objects. Running average method has been used to detect the moving objects in the video, which is taken from a static camera. Tracking of foreground objects has been realized by using a Kalman filter. After background subtraction, morphological operators are used to remove noises detected as foreground. Active contour models (snakes) are the segmentation tools for the extracted foregrounds. Snakes have been also used as an extra tool for object tracking. Location of the detected object was not accurate founded by this system.

Chapter 3: Interfacing and Detection

3.1user interfaces:

Front-end software:simple react js form python opencv
Back-end software: django DB

Login :

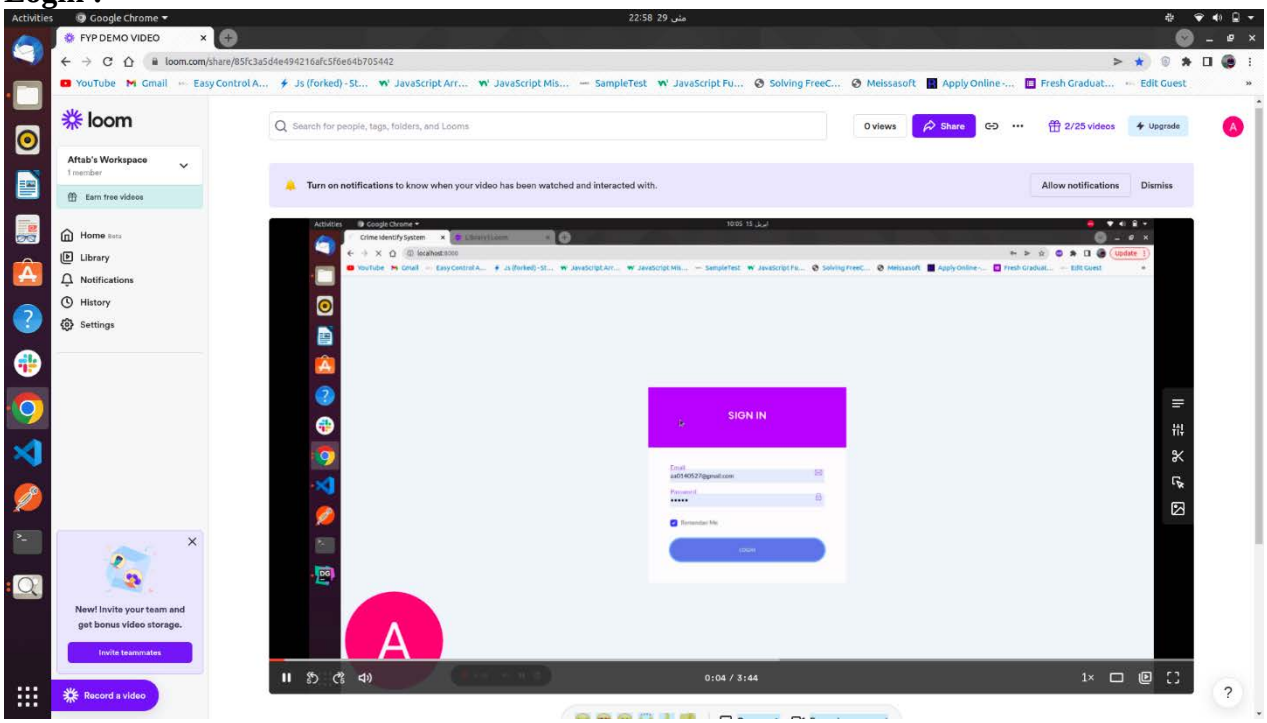

FIGURE 3.1

Dashboard view

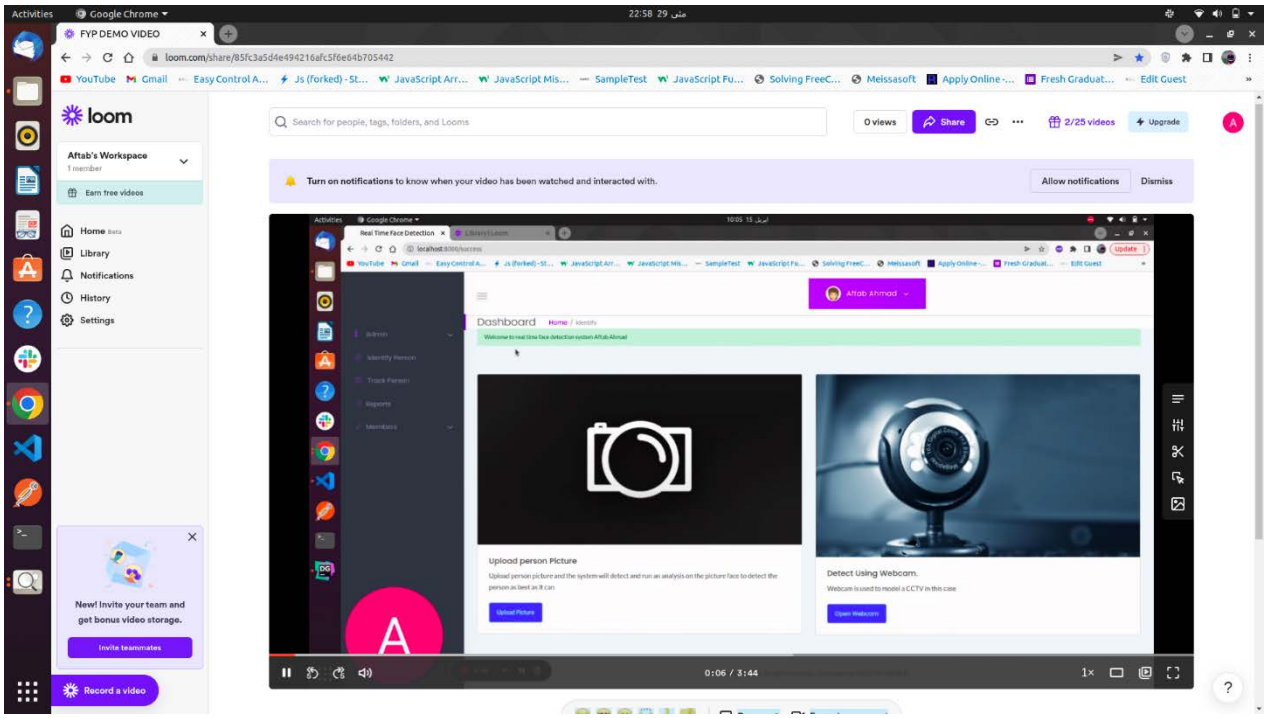

FIGURE 3.2

Upload Picture:

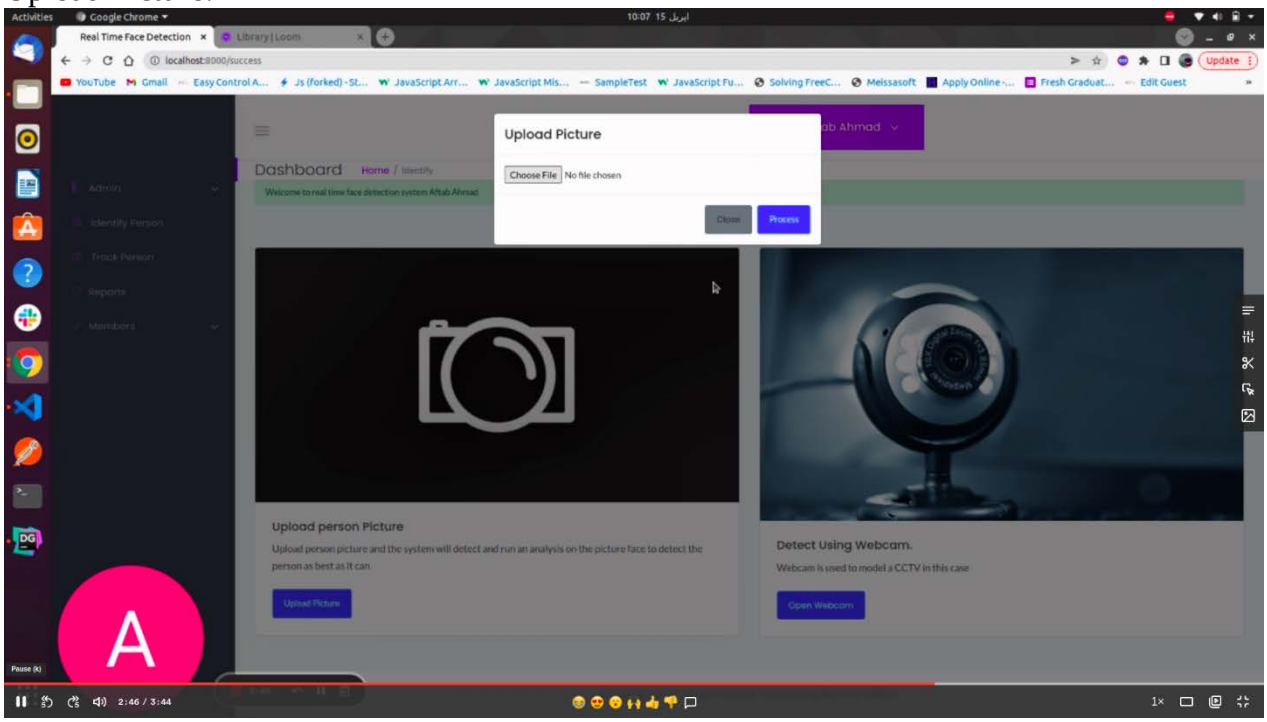

FIGURE 3.3

Multiple Person Detection:

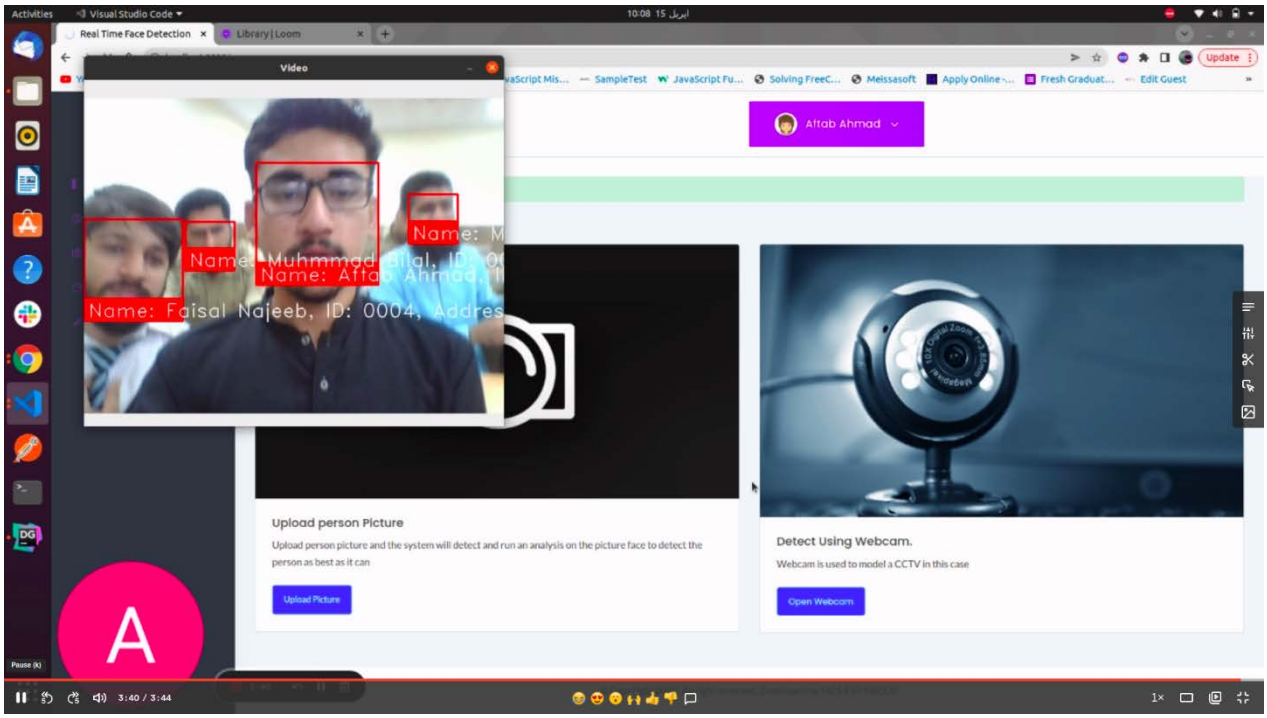

FIGURE 3.4

Single Person Detection:

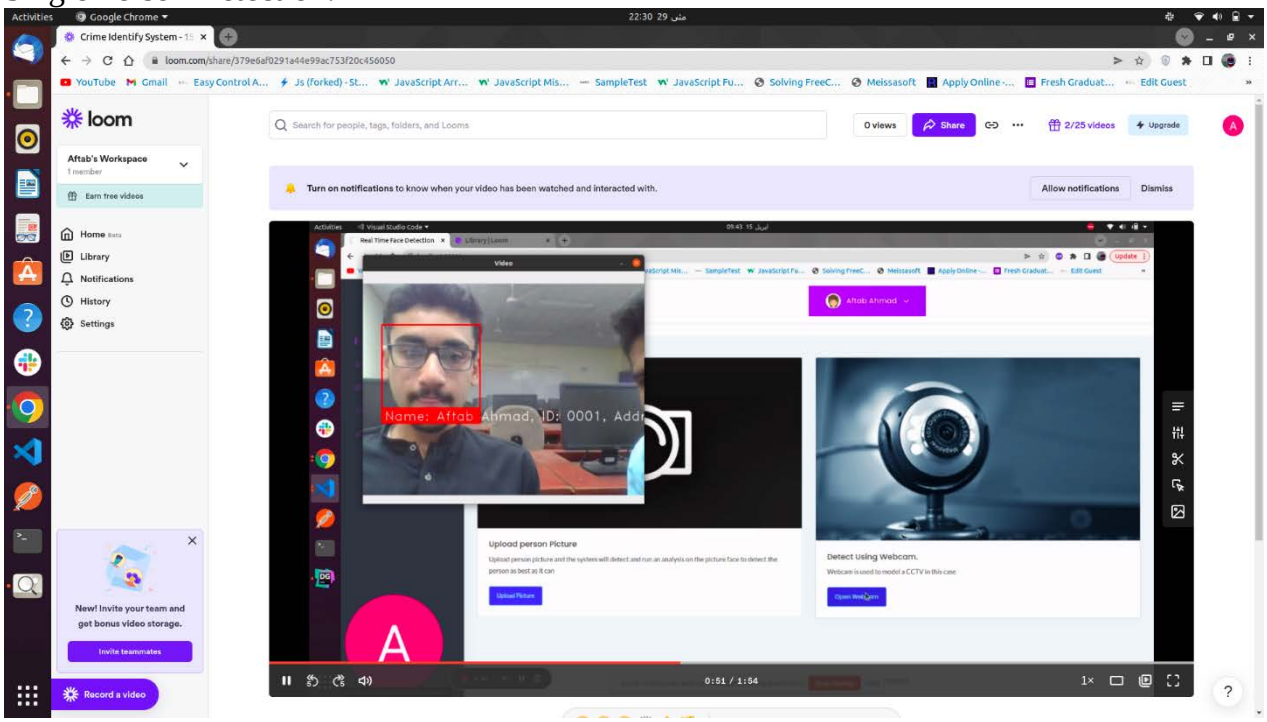

FIGURE 3.5

Tracking Records:

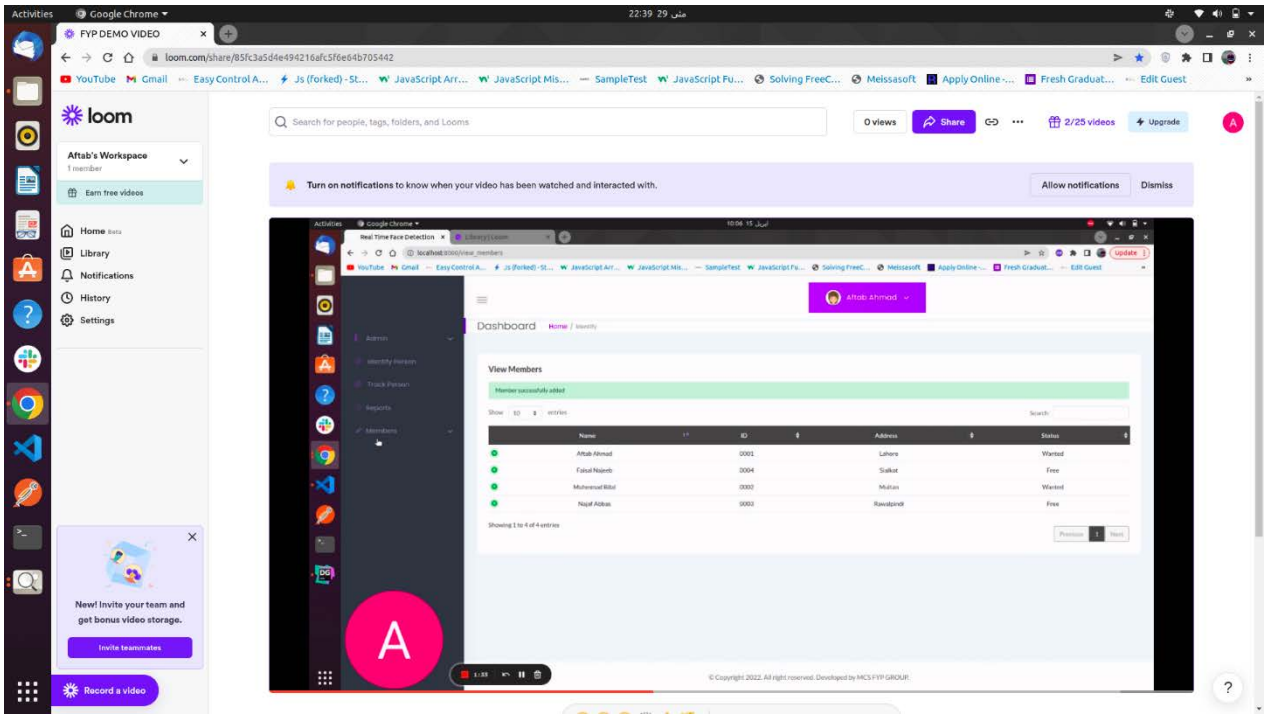

FIGURE 3.6

Registered User:

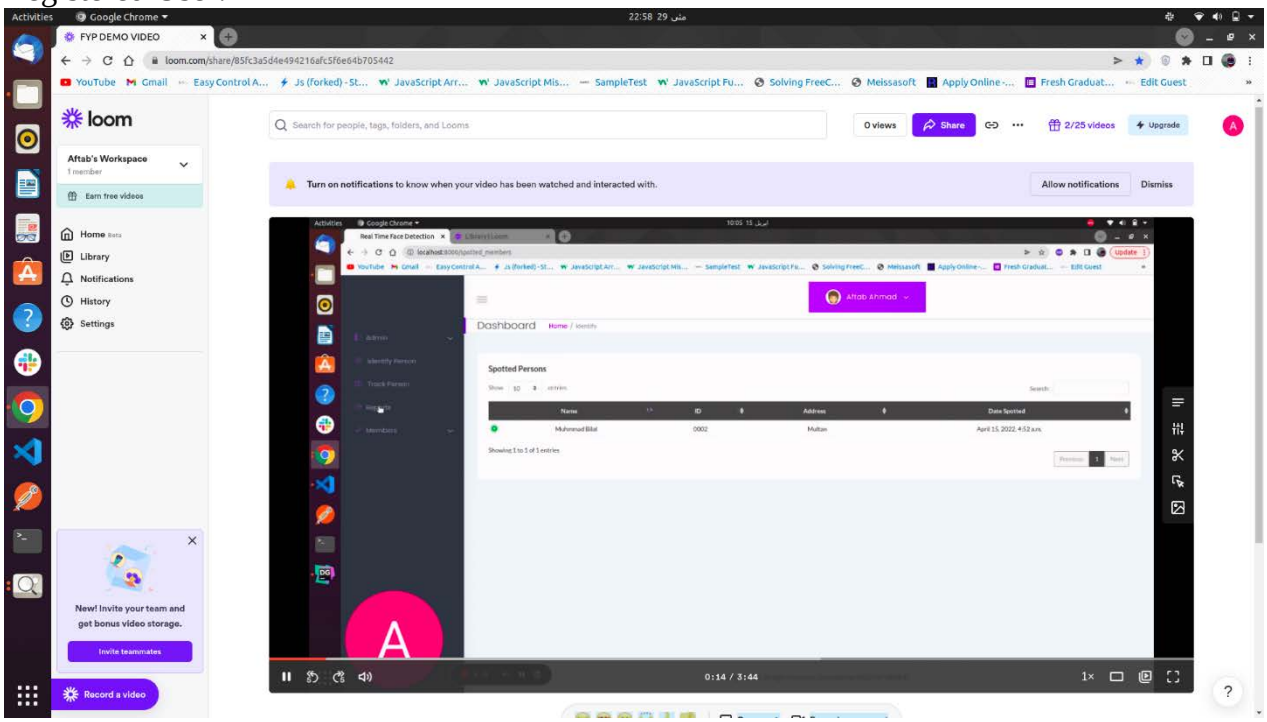

FIGURE 3.7

3.2hardware interfaces:

- Windows.
- A browser which supports jupyternotebook(anaconda) distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment.

3.3software interfaces :

Following are the software used for the real time movement tracking system.
Windows7/8/10 operating system will be used during development process
Pycharm
Anaconda
Matlab

3.4communication interfaces:

TCP/IP will provide the communictaion between RTMTSUAI and Google API services

3.5 Person Detection

There can be multiple persons inside an area covered by a single camera The task is to correctly detect the person we are concerened and its images are included in our dataset and the person that is not concerned must be notified as unknown .From multiple faces to correctly identify the person that we are concerned at least a large number of images of the person must be provided to the dataset. Detection of person is an essential part of our project. When it is detected, the user is notified by displaying name of the person on the screen.

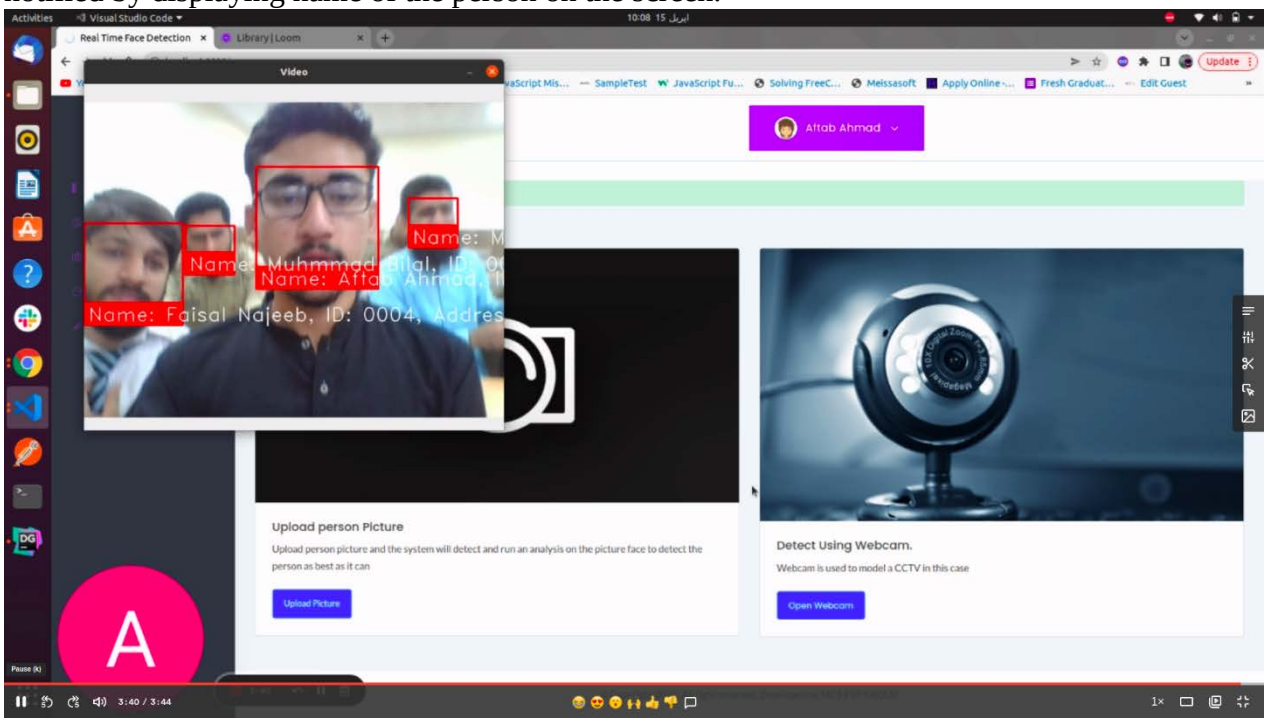

FIGURE 3.7

3.5.1 Preparing Dataset

Dataset refers to a compilation of instances that share a mutual attribute. Dataset is used to sculpt a machine learning algorithm going forward. The more data is provided, the better is the efficiency. In this model we used about 30+ images of a person which includes different face angles.

Images:

FIGURE 3.8

To create bounding boxes, the software **LabelImg version Windows_v1.8.0** is used

3.6System overview

This software product is AI and machine learning based applied to surveilllance of secured area. The RTMS is used for human identification, detection, movement tracking.It is a client-server based web application.

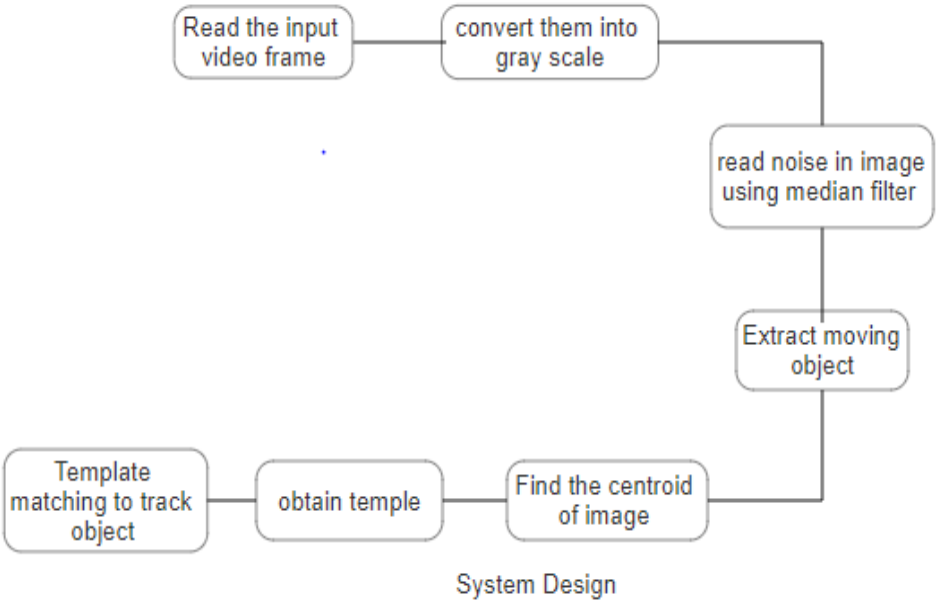

FIGURE 3.9

3.7System Architecture

3.7.1Architectural Design

This software has many architecture design in it but the pipe and filter architecture design is being used for the core functionality. The RTMS is continuously filtering the data coming the camera and analyzing it to give the required output in form of human surveillance.

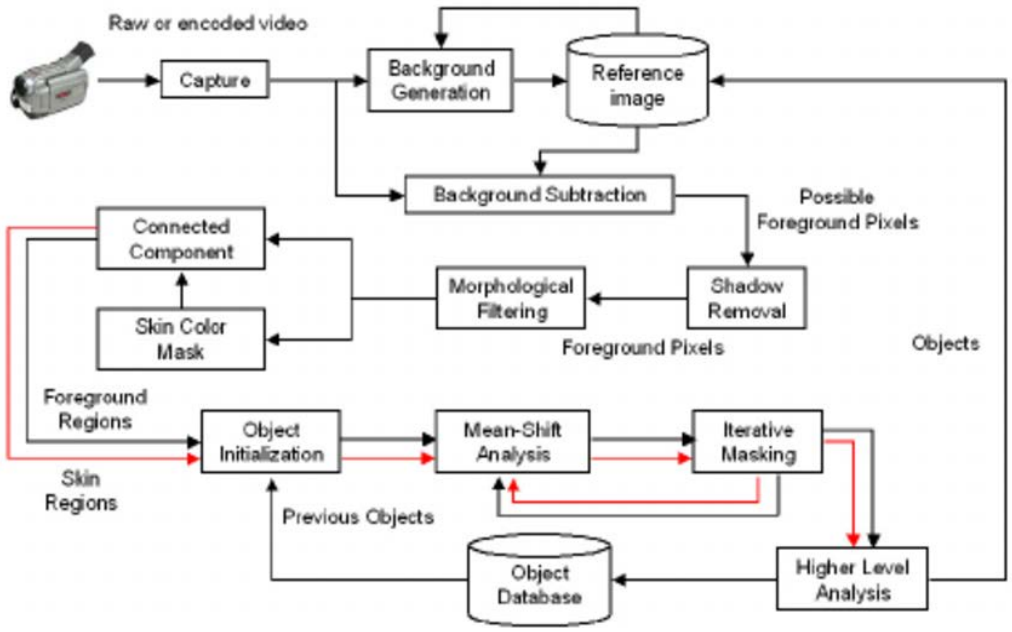

FIGURE 3.10

3.7.2Decomposition Architecture:

Class diagram:

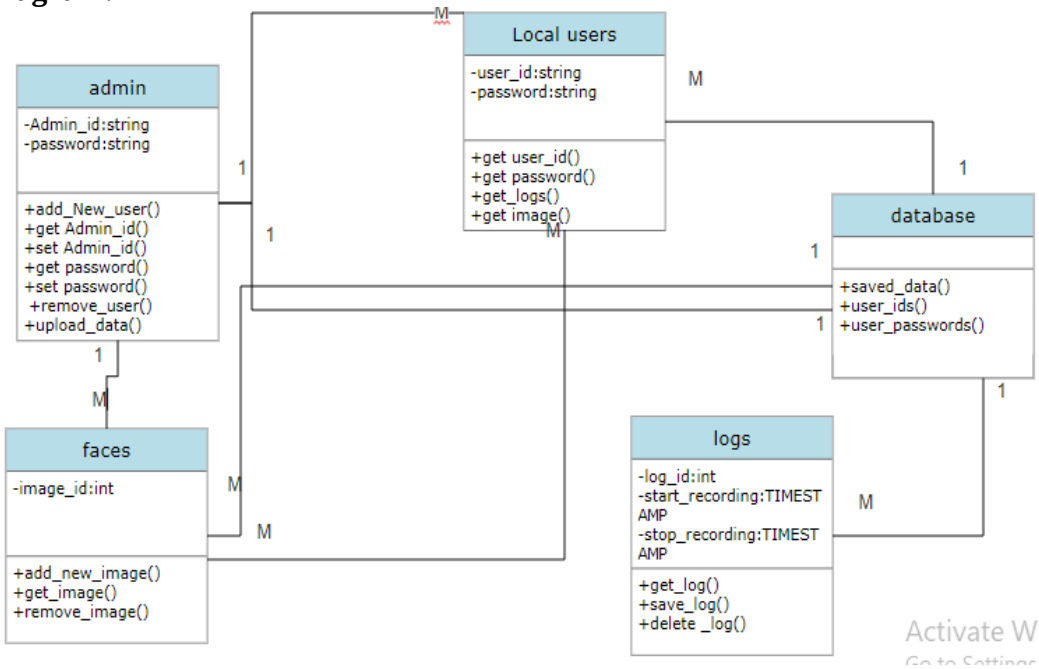

FIGURE 3.11

3.8Module Decomposition:

The system is overall decomposed into two main components.

User

admin

Admin is the user directly interacting with the system through the Human Interface of the system. The admin is superuser and can create and remove the normal user.

Nodes

Nodes are the end systems and network devices of the system which are added by the admin and send logs to the central server for processing the input to get the output.

Server

Server is responsible for collection, processing of the logs to display on the interface (frontend) of the system and generate the output after processing the input into machine learning and artificial intelligence model to give the output.

3.9Process Decomposition:

Use case Diagram

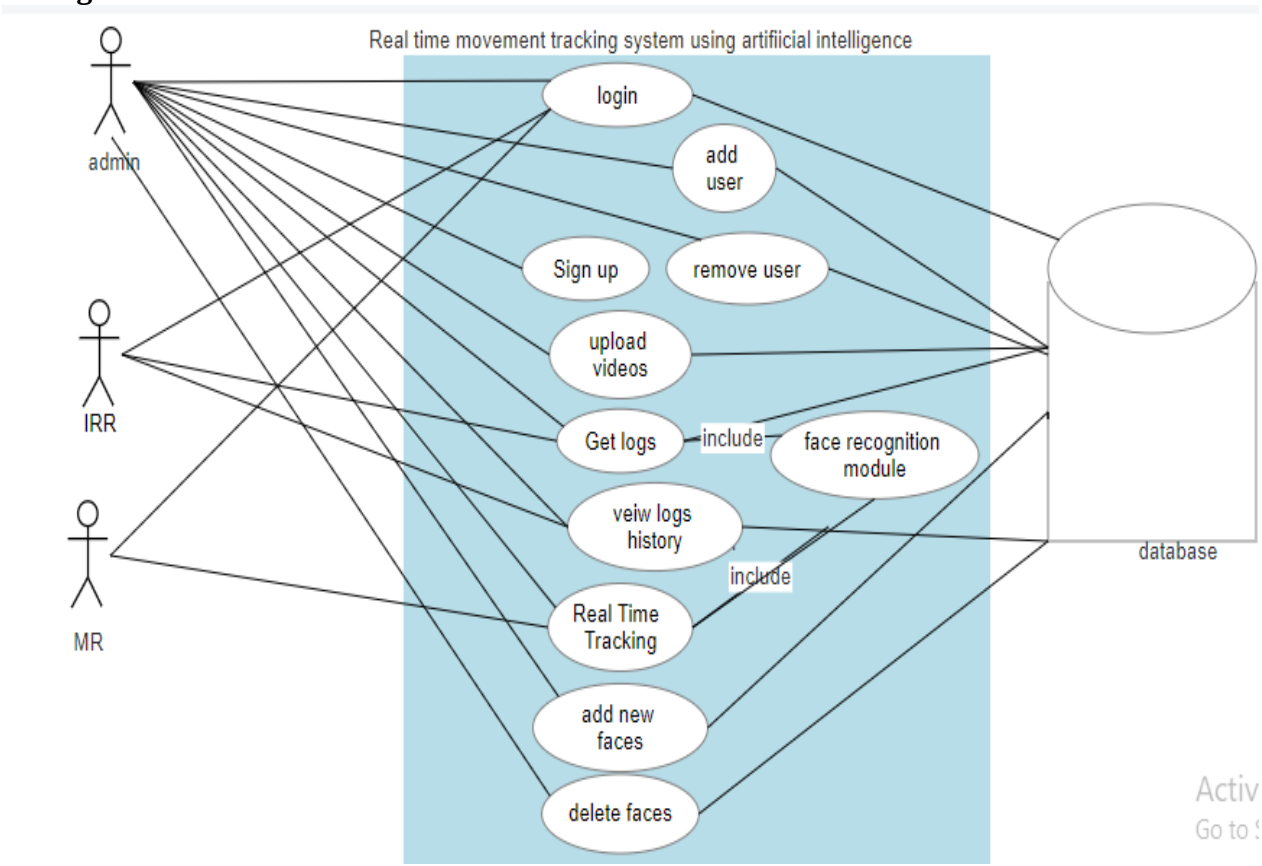

FIGURE 3.11

Sequence Diagram:

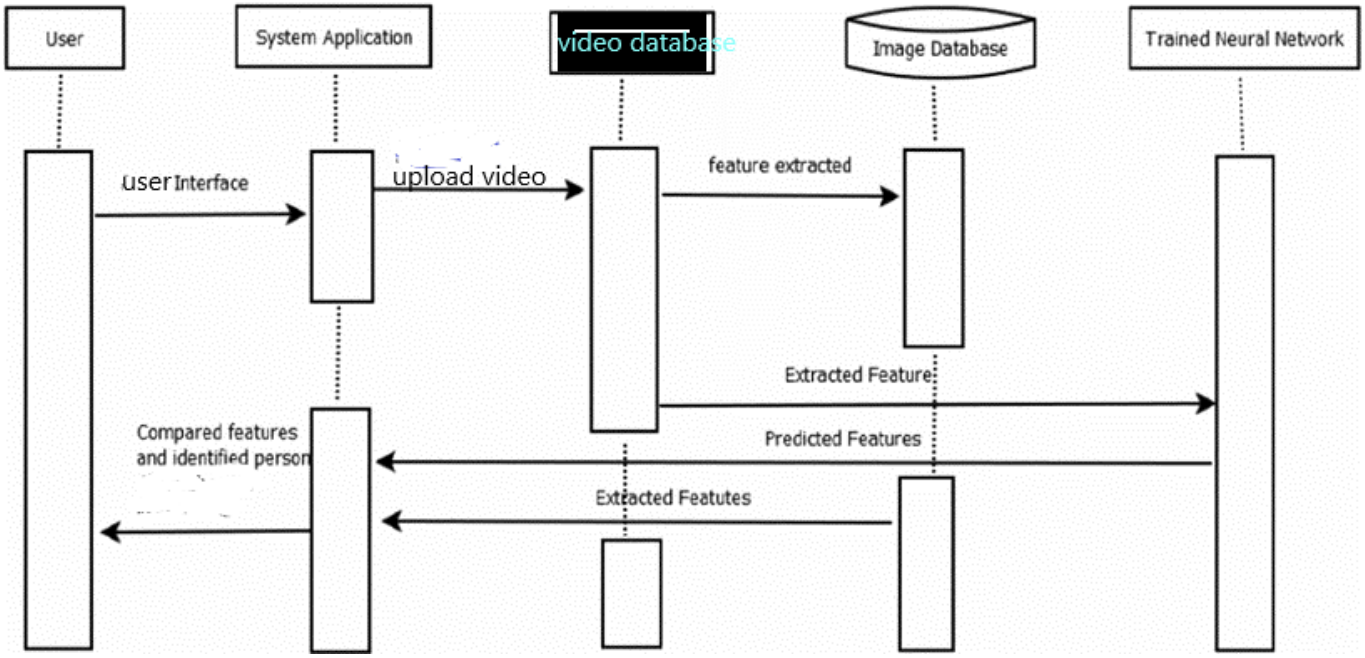

FIGURE 3.12

3.9.1Design Rationale

The architecture for the RTMS is model-view-template. The MVT is an effective way to implement the web application. It is an easy way to implement the microservices, testing and the feasibility of software. The separation of concern means separating the application into divided individual and independent modules user interaction, data processing
and database.

3.10 Data Design

3.10.1 Data Description

Our system uses a single central database to collect and store all the data coming from the camera in the network. This database is located at the central server.

3.11Component Design:
Activity Diagram:

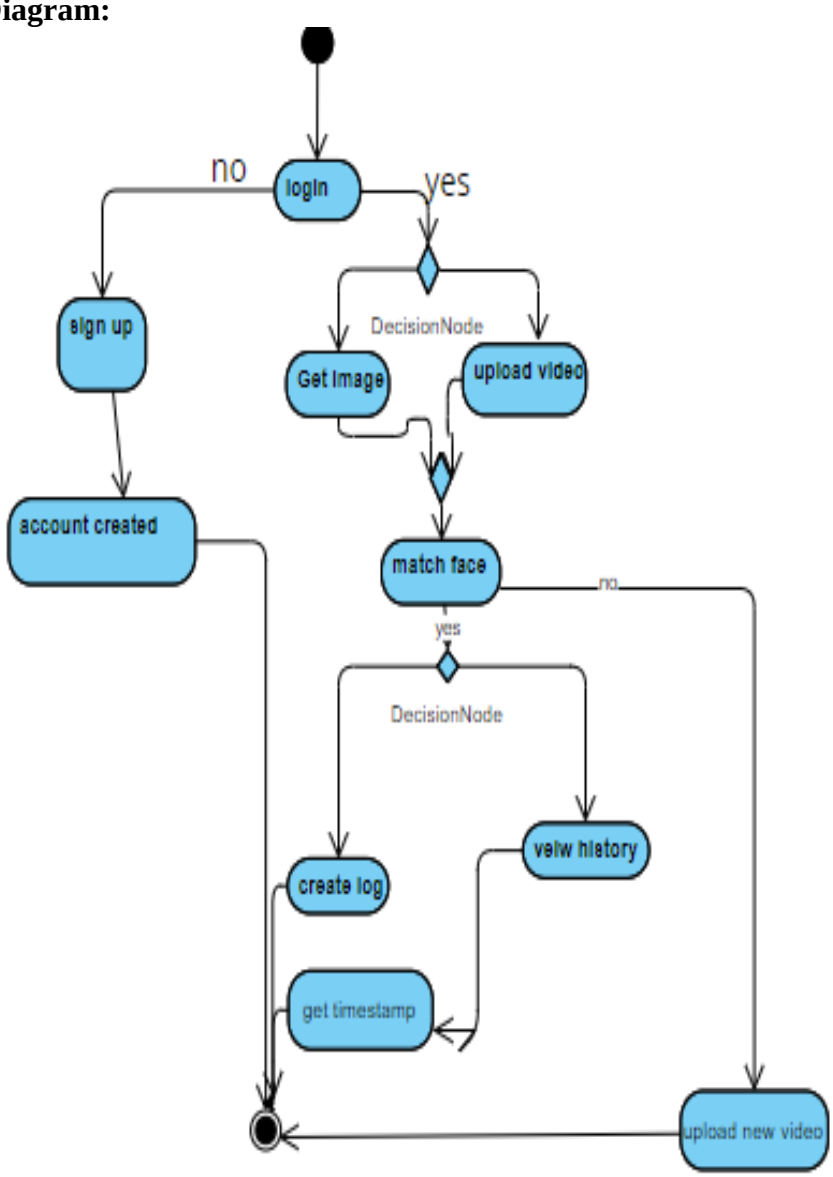

FIGURE 3.13

3.11.1Working Methadology:

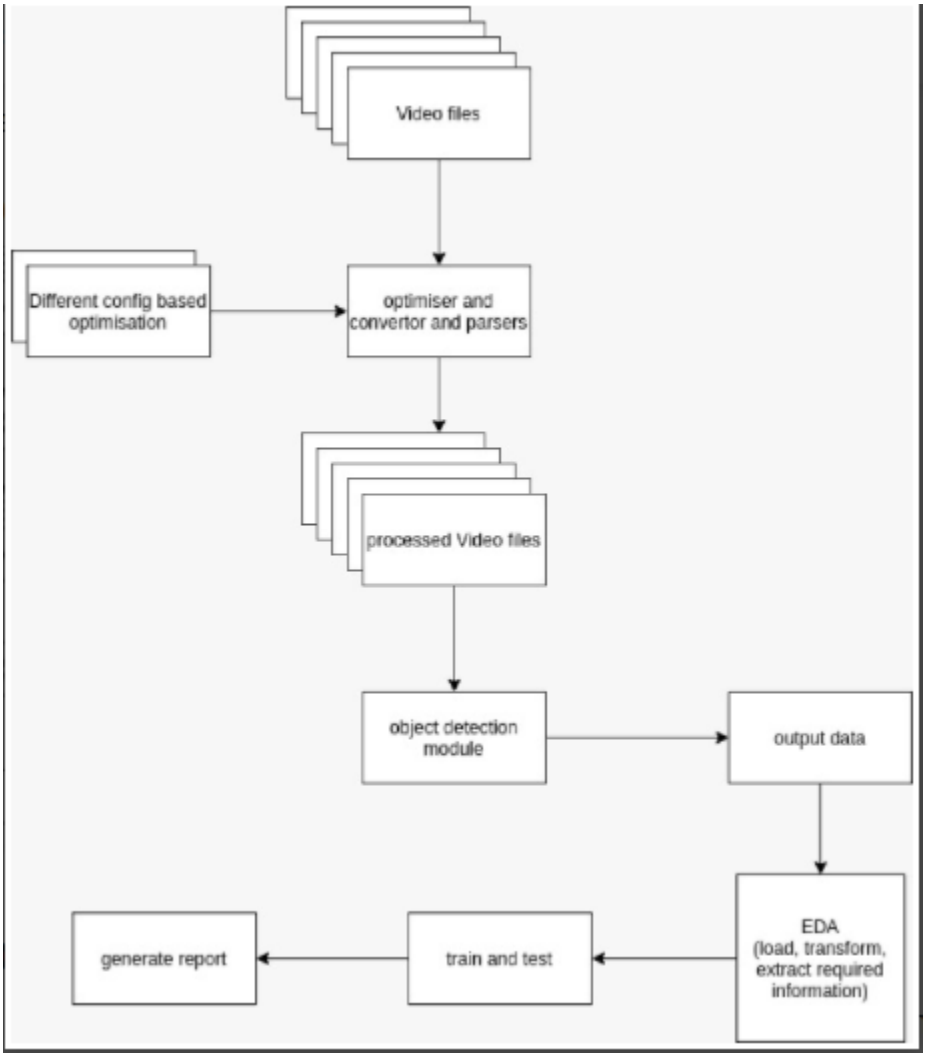

FIGURE 3.14

Chapter 4: Code Analysis And Evaluation

4.1 Pseudo Code for APIs:

4.1.1 Face recognition code :

```
from __future__ import print_function
import click
import os
import re
import face_recognition.api as face_recognition
import multiprocessing
import sys
import itertools
```

```
def print_result(filename, location):
    top, right, bottom, left = location
    print("{},{},{},{},{}".format(filename, top, right, bottom, left))


def test_image(image_to_check, model):
    unknown_image = face_recognition.load_image_file(image_to_check)
    face_locations = face_recognition.face_locations(unknown_image, number_of_times_to_upsample=0, model=model)

    for face_location in face_locations:
        print_result(image_to_check, face_location)


def image_files_in_folder(folder):
    return [os.path.join(folder, f) for f in os.listdir(folder) if re.match(r'.*\.(jpg|jpeg|png)', f, flags=re.I)]


def process_images_in_process_pool(images_to_check, number_of_cpus, model):
    if number_of_cpus == -1:
        processes = None
    else:
        processes = number_of_cpus

    # macOS will crash due to a bug in libdispatch if you don't use 'forkserver'
    context = multiprocessing
    if "forkserver" in multiprocessing.get_all_start_methods():
        context = multiprocessing.get_context("forkserver")

    pool = context.Pool(processes=processes)

    function_parameters = zip(
        images_to_check,
        itertools.repeat(model),
    )

    pool.starmap(test_image, function_parameters)


@click.command()
@click.argument('image_to_check')
@click.option('--cpus', default=1, help='number of CPU cores to use in parallel. -1 means "use all in system"')
@click.option('--model', default="hog", help='Which face detection model to use. Options are "hog" or "cnn".')
def main(image_to_check, cpus, model):
    # Multi-core processing only supported on Python 3.4 or greater
    if (sys.version_info < (3, 4)) and cpus != 1:
        click.echo("WARNING: Multi-processing support requires Python 3.4 or greater. Falling back to single-threaded
processing!")
        cpus = 1

    if os.path.isdir(image_to_check):
        if cpus == 1:
            [test_image(image_file, model) for image_file in image_files_in_folder(image_to_check)]
        else:
            process_images_in_process_pool(image_files_in_folder(image_to_check), cpus, model)
    else:
        test_image(image_to_check, model)
```

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

4.1.2Face Recognition code:

```
# -*- coding: utf-8 -*-
from __future__ import print_function
import click
import os
import re
import face_recognition.api as face_recognition
import multiprocessing
import itertools
import sys
import PIL.Image
import numpy as np


def scan_known_people(known_people_folder):
    known_names = []
    known_face_encodings = []

    for file in image_files_in_folder(known_people_folder):
        basename = os.path.splitext(os.path.basename(file))[0]
        img = face_recognition.load_image_file(file)
        encodings = face_recognition.face_encodings(img)

        if len(encodings) > 1:
            click.echo("WARNING: More than one face found in {}. Only considering the first face.".format(file))

        if len(encodings) == 0:
            click.echo("WARNING: No faces found in {}. Ignoring file.".format(file))
        else:
            known_names.append(basename)
            known_face_encodings.append(encodings[0])

    return known_names, known_face_encodings


def print_result(filename, name, distance, show_distance=False):
    if show_distance:
        print("{},{},{}".format(filename, name, distance))
    else:
        print("{},{}".format(filename, name))


def test_image(image_to_check, known_names, known_face_encodings, tolerance=0.6, show_distance=False):
    unknown_image = face_recognition.load_image_file(image_to_check)

    # Scale down image if it's giant so things run a little faster
    if max(unknown_image.shape) > 1600:
        pil_img = PIL.Image.fromarray(unknown_image)
        pil_img.thumbnail((1600, 1600), PIL.Image.LANCZOS)
        unknown_image = np.array(pil_img)

    unknown_encodings = face_recognition.face_encodings(unknown_image)

    for unknown_encoding in unknown_encodings:
        distances = face_recognition.face_distance(known_face_encodings, unknown_encoding)
        result = list(distances <= tolerance)
```

```
        if True in result:
            [print_result(image_to_check, name, distance, show_distance) for is_match, name, distance in zip(result,
known_names, distances) if is_match]
        else:
            print_result(image_to_check, "unknown_person", None, show_distance)

    if not unknown_encodings:
        # print out fact that no faces were found in image
        print_result(image_to_check, "no_persons_found", None, show_distance)


def image_files_in_folder(folder):
    return [os.path.join(folder, f) for f in os.listdir(folder) if re.match(r'.*\.(jpg|jpeg|png)', f, flags=re.I)]


def process_images_in_process_pool(images_to_check, known_names, known_face_encodings, number_of_cpus,
tolerance, show_distance):
    if number_of_cpus == -1:
        processes = None
    else:
        processes = number_of_cpus

    # macOS will crash due to a bug in libdispatch if you don't use 'forkserver'
    context = multiprocessing
    if "forkserver" in multiprocessing.get_all_start_methods():
        context = multiprocessing.get_context("forkserver")

    pool = context.Pool(processes=processes)

    function_parameters = zip(
        images_to_check,
        itertools.repeat(known_names),
        itertools.repeat(known_face_encodings),
        itertools.repeat(tolerance),
        itertools.repeat(show_distance)
    )

    pool.starmap(test_image, function_parameters)


@click.command()
@click.argument('known_people_folder')
@click.argument('image_to_check')
@click.option('--cpus', default=1, help='number of CPU cores to use in parallel (can speed up processing lots of images). -1
means "use all in system"')
@click.option('--tolerance', default=0.6, help='Tolerance for face comparisons. Default is 0.6. Lower this if you get
multiple matches for the same person.')
@click.option('--show-distance', default=False, type=bool, help='Output face distance. Useful for tweaking tolerance
setting.')
def main(known_people_folder, image_to_check, cpus, tolerance, show_distance):
    known_names, known_face_encodings = scan_known_people(known_people_folder)

    # Multi-core processing only supported on Python 3.4 or greater
    if (sys.version_info < (3, 4)) and cpus != 1:
        click.echo("WARNING: Multi-processing support requires Python 3.4 or greater. Falling back to single-threaded
processing!")
        cpus = 1
```

```
    if os.path.isdir(image_to_check):
        if cpus == 1:
            [test_image(image_file, known_names, known_face_encodings, tolerance, show_distance) for image_file in
image_files_in_folder(image_to_check)]
        else:
            process_images_in_process_pool(image_files_in_folder(image_to_check), known_names, known_face_encodings,
cpus, tolerance, show_distance)
    else:
        test_image(image_to_check, known_names, known_face_encodings, tolerance, show_distance)


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

4.1.3Face recognition API:

```
# -*- coding: utf-8 -*-

import PIL.Image
import dlib
import numpy as np
from PIL import ImageFile

try:
    import face_recognition_models
except Exception:
    print("Please install `face_recognition_models` with this command before using
`face_recognition`:\n")
    print("pip install git+https://github.com/ageitgey/face_recognition_models")
    quit()

ImageFile.LOAD_TRUNCATED_IMAGES = True

face_detector = dlib.get_frontal_face_detector()

predictor_68_point_model = face_recognition_models.pose_predictor_model_location()
pose_predictor_68_point = dlib.shape_predictor(predictor_68_point_model)

predictor_5_point_model                                                                    =
face_recognition_models.pose_predictor_five_point_model_location()
pose_predictor_5_point = dlib.shape_predictor(predictor_5_point_model)

cnn_face_detection_model = face_recognition_models.cnn_face_detector_model_location()
cnn_face_detector = dlib.cnn_face_detection_model_v1(cnn_face_detection_model)

face_recognition_model = face_recognition_models.face_recognition_model_location()
face_encoder = dlib.face_recognition_model_v1(face_recognition_model)


def _rect_to_css(rect):
    """
```

```
    Convert a dlib 'rect' object to a plain tuple in (top, right, bottom, left) order

    :param rect: a dlib 'rect' object
    :return: a plain tuple representation of the rect in (top, right, bottom, left) order
    """
    return rect.top(), rect.right(), rect.bottom(), rect.left()


def _css_to_rect(css):
    """
    Convert a tuple in (top, right, bottom, left) order to a dlib `rect` object

    :param css:  plain tuple representation of the rect in (top, right, bottom, left) order
    :return: a dlib `rect` object
    """
    return dlib.rectangle(css[3], css[0], css[1], css[2])


def _trim_css_to_bounds(css, image_shape):
    """
    Make sure a tuple in (top, right, bottom, left) order is within the bounds of the image.

    :param css:  plain tuple representation of the rect in (top, right, bottom, left) order
    :param image_shape: numpy shape of the image array
    :return: a trimmed plain tuple representation of the rect in (top, right, bottom, left) order
    """
    return max(css[0], 0), min(css[1], image_shape[1]), min(css[2], image_shape[0]), max(css[3],
0)


def face_distance(face_encodings, face_to_compare):
    """
    Given a list of face encodings, compare them to a known face encoding and get a euclidean
distance
    for each comparison face. The distance tells you how similar the faces are.

    :param faces: List of face encodings to compare
    :param face_to_compare: A face encoding to compare against
    :return: A numpy ndarray with the distance for each face in the same order as the 'faces' array
    """
    if len(face_encodings) == 0:
        return np.empty((0))

    return np.linalg.norm(face_encodings - face_to_compare, axis=1)
```

```
def load_image_file(file, mode='RGB'):
    """
    Loads an image file (.jpg, .png, etc) into a numpy array

    :param file: image file name or file object to load
    :param mode: format to convert the image to. Only 'RGB' (8-bit RGB, 3 channels) and 'L' (black and white) are supported.
    :return: image contents as numpy array
    """
    im = PIL.Image.open(file)
    if mode:
        im = im.convert(mode)
    return np.array(im)


def _raw_face_locations(img, number_of_times_to_upsample=1, model="hog"):
    """
    Returns an array of bounding boxes of human faces in a image

    :param img: An image (as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for faces. Higher numbers find smaller faces.
    :param model: Which face detection model to use. "hog" is less accurate but faster on CPUs. "cnn" is a more accurate
                  deep-learning model which is GPU/CUDA accelerated (if available). The default is "hog".
    :return: A list of dlib 'rect' objects of found face locations
    """
    if model == "cnn":
        return cnn_face_detector(img, number_of_times_to_upsample)
    else:
        return face_detector(img, number_of_times_to_upsample)


def face_locations(img, number_of_times_to_upsample=1, model="hog"):
    """
    Returns an array of bounding boxes of human faces in a image

    :param img: An image (as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for faces. Higher numbers find smaller faces.
    :param model: Which face detection model to use. "hog" is less accurate but faster on CPUs. "cnn" is a more accurate
                  deep-learning model which is GPU/CUDA accelerated (if available). The default is "hog".
    :return: A list of tuples of found face locations in css (top, right, bottom, left) order
```

```
    """
    if model == "cnn":
        return [_trim_css_to_bounds(_rect_to_css(face.rect), img.shape) for face in _raw_face_locations(img, number_of_times_to_upsample, "cnn")]
    else:
        return [_trim_css_to_bounds(_rect_to_css(face), img.shape) for face in _raw_face_locations(img, number_of_times_to_upsample, model)]


def _raw_face_locations_batched(images, number_of_times_to_upsample=1, batch_size=128):
    """
    Returns an 2d array of dlib rects of human faces in a image using the cnn face detector

    :param img: A list of images (each as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for faces. Higher numbers find smaller faces.
    :return: A list of dlib 'rect' objects of found face locations
    """
    return cnn_face_detector(images, number_of_times_to_upsample, batch_size=batch_size)


def batch_face_locations(images, number_of_times_to_upsample=1, batch_size=128):
    """
    Returns an 2d array of bounding boxes of human faces in a image using the cnn face detector
    If you are using a GPU, this can give you much faster results since the GPU
    can process batches of images at once. If you aren't using a GPU, you don't need this function.

    :param img: A list of images (each as a numpy array)
    :param number_of_times_to_upsample: How many times to upsample the image looking for faces. Higher numbers find smaller faces.
    :param batch_size: How many images to include in each GPU processing batch.
    :return: A list of tuples of found face locations in css (top, right, bottom, left) order
    """
    def convert_cnn_detections_to_css(detections):
        return [_trim_css_to_bounds(_rect_to_css(face.rect), images[0].shape) for face in detections]

    raw_detections_batched = _raw_face_locations_batched(images, number_of_times_to_upsample, batch_size)

    return list(map(convert_cnn_detections_to_css, raw_detections_batched))


def _raw_face_landmarks(face_image, face_locations=None, model="large"):
    if face_locations is None:
        face_locations = _raw_face_locations(face_image)
    else:
```

```
        face_locations = [_css_to_rect(face_location) for face_location in face_locations]

    pose_predictor = pose_predictor_68_point

    if model == "small":
        pose_predictor = pose_predictor_5_point

    return [pose_predictor(face_image, face_location) for face_location in face_locations]


def face_landmarks(face_image, face_locations=None, model="large"):
    """
    Given an image, returns a dict of face feature locations (eyes, nose, etc) for each face in the image

    :param face_image: image to search
    :param face_locations: Optionally provide a list of face locations to check.
    :param model: Optional - which model to use. "large" (default) or "small" which only returns 5 points but is faster.
    :return: A list of dicts of face feature locations (eyes, nose, etc)
    """
    landmarks = _raw_face_landmarks(face_image, face_locations, model)
    landmarks_as_tuples = [[(p.x, p.y) for p in landmark.parts()] for landmark in landmarks]

    # For a definition of each point index, see https://cdn-images-1.medium.com/max/1600/1*AbEg31EgkbXSQehuNJBlWg.png
    if model == 'large':
        return [{
            "chin": points[0:17],
            "left_eyebrow": points[17:22],
            "right_eyebrow": points[22:27],
            "nose_bridge": points[27:31],
            "nose_tip": points[31:36],
            "left_eye": points[36:42],
            "right_eye": points[42:48],
            "top_lip": points[48:55] + [points[64]] + [points[63]] + [points[62]] + [points[61]] + [points[60]],
            "bottom_lip": points[54:60] + [points[48]] + [points[60]] + [points[67]] + [points[66]] + [points[65]] + [points[64]]
        } for points in landmarks_as_tuples]
    elif model == 'small':
        return [{
            "nose_tip": [points[4]],
            "left_eye": points[2:4],
            "right_eye": points[0:2],
        } for points in landmarks_as_tuples]
```

```
    else:
        raise ValueError("Invalid landmarks model type. Supported models are ['small', 'large'].")


def face_encodings(face_image, known_face_locations=None, num_jitters=1):
    """
    Given an image, return the 128-dimension face encoding for each face in the image.

    :param face_image: The image that contains one or more faces
    :param known_face_locations: Optional - the bounding boxes of each face if you already know them.
    :param num_jitters: How many times to re-sample the face when calculating encoding. Higher is more accurate, but slower (i.e. 100 is 100x slower)
    :return: A list of 128-dimensional face encodings (one for each face in the image)
    """
    raw_landmarks = _raw_face_landmarks(face_image, known_face_locations, model="small")
    return [np.array(face_encoder.compute_face_descriptor(face_image, raw_landmark_set, num_jitters)) for raw_landmark_set in raw_landmarks]


def compare_faces(known_face_encodings, face_encoding_to_check, tolerance=0.6):
    """
    Compare a list of face encodings against a candidate encoding to see if they match.

    :param known_face_encodings: A list of known face encodings
    :param face_encoding_to_check: A single face encoding to compare against the list
    :param tolerance: How much distance between faces to consider it a match. Lower is more strict. 0.6 is typical best performance.
    :return: A list of True/False values indicating which known_face_encodings match the face encoding to check
    """
    return list(face_distance(known_face_encodings, face_encoding_to_check) <= tolerance)
```

4.1.4Test face recognition module:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

"""
test_face_recognition
----------------------------------

Tests for `face_recognition` module.
"""


import unittest
```

```
import os
import numpy as np
from click.testing import CliRunner

from face_recognition import api
from face_recognition import face_recognition_cli
from face_recognition import face_detection_cli


class Test_face_recognition(unittest.TestCase):

    def test_load_image_file(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        self.assertEqual(img.shape, (1137, 910, 3))

    def test_load_image_file_32bit(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'32bit.png'))
        self.assertEqual(img.shape, (1200, 626, 3))

    def test_raw_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        detected_faces = api._raw_face_locations(img)

        self.assertEqual(len(detected_faces), 1)
        self.assertEqual(detected_faces[0].top(), 142)
        self.assertEqual(detected_faces[0].bottom(), 409)

    def test_cnn_raw_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        detected_faces = api._raw_face_locations(img, model="cnn")

        self.assertEqual(len(detected_faces), 1)
        self.assertAlmostEqual(detected_faces[0].rect.top(), 144, delta=25)
        self.assertAlmostEqual(detected_faces[0].rect.bottom(), 389, delta=25)

    def test_raw_face_locations_32bit_image(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'32bit.png'))
        detected_faces = api._raw_face_locations(img)

        self.assertEqual(len(detected_faces), 1)
        self.assertEqual(detected_faces[0].top(), 290)
```

```
        self.assertEqual(detected_faces[0].bottom(), 558)

    def test_cnn_raw_face_locations_32bit_image(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'32bit.png'))
        detected_faces = api._raw_face_locations(img, model="cnn")

        self.assertEqual(len(detected_faces), 1)
        self.assertAlmostEqual(detected_faces[0].rect.top(), 259, delta=25)
        self.assertAlmostEqual(detected_faces[0].rect.bottom(), 552, delta=25)

    def test_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        detected_faces = api.face_locations(img)

        self.assertEqual(len(detected_faces), 1)
        self.assertEqual(detected_faces[0], (142, 617, 409, 349))

    def test_cnn_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        detected_faces = api.face_locations(img, model="cnn")

        self.assertEqual(len(detected_faces), 1)
        self.assertAlmostEqual(detected_faces[0][0], 144, delta=25)
        self.assertAlmostEqual(detected_faces[0][1], 608, delta=25)
        self.assertAlmostEqual(detected_faces[0][2], 389, delta=25)
        self.assertAlmostEqual(detected_faces[0][3], 363, delta=25)

    def test_partial_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama_partial_face.jpg'))
        detected_faces = api.face_locations(img)

        self.assertEqual(len(detected_faces), 1)
        self.assertEqual(detected_faces[0], (142, 191, 365, 0))

        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama_partial_face2.jpg'))
        detected_faces = api.face_locations(img)

        self.assertEqual(len(detected_faces), 1)
        self.assertEqual(detected_faces[0], (142, 551, 409, 349))

    def test_raw_face_locations_batched(self):
```

```
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        images = [img, img, img]
        batched_detected_faces = api._raw_face_locations_batched(images,
number_of_times_to_upsample=0)

        for detected_faces in batched_detected_faces:
            self.assertEqual(len(detected_faces), 1)
            self.assertEqual(detected_faces[0].rect.top(), 154)
            self.assertEqual(detected_faces[0].rect.bottom(), 390)

    def test_batched_face_locations(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        images = [img, img, img]

        batched_detected_faces = api.batch_face_locations(images,
number_of_times_to_upsample=0)

        for detected_faces in batched_detected_faces:
            self.assertEqual(len(detected_faces), 1)
            self.assertEqual(detected_faces[0], (154, 611, 390, 375))

    def test_raw_face_landmarks(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        face_landmarks = api._raw_face_landmarks(img)
        example_landmark = face_landmarks[0].parts()[10]

        self.assertEqual(len(face_landmarks), 1)
        self.assertEqual(face_landmarks[0].num_parts, 68)
        self.assertEqual((example_landmark.x, example_landmark.y), (552, 399))

    def test_face_landmarks(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        face_landmarks = api.face_landmarks(img)

        self.assertEqual(
            set(face_landmarks[0].keys()),
            set(['chin', 'left_eyebrow', 'right_eyebrow', 'nose_bridge',
                 'nose_tip', 'left_eye', 'right_eye', 'top_lip',
                 'bottom_lip']))
        self.assertEqual(
            face_landmarks[0]['chin'],
            [(369, 220), (372, 254), (378, 289), (384, 322), (395, 353),
```

```
            (414, 382), (437, 407), (464, 424), (495, 428), (527, 420),
            (552, 399), (576, 372), (594, 344), (604, 314), (610, 282),
            (613, 250), (615, 219)])

    def test_face_landmarks_small_model(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        face_landmarks = api.face_landmarks(img, model="small")

        self.assertEqual(
            set(face_landmarks[0].keys()),
            set(['nose_tip', 'left_eye', 'right_eye']))
        self.assertEqual(face_landmarks[0]['nose_tip'], [(496, 295)])

    def test_face_encodings(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        encodings = api.face_encodings(img)

        self.assertEqual(len(encodings), 1)
        self.assertEqual(len(encodings[0]), 128)

    def test_face_distance(self):
        img_a1 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama.jpg'))
        img_a2 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama2.jpg'))
        img_a3 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'obama3.jpg'))

        img_b1 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'biden.jpg'))

        face_encoding_a1 = api.face_encodings(img_a1)[0]
        face_encoding_a2 = api.face_encodings(img_a2)[0]
        face_encoding_a3 = api.face_encodings(img_a3)[0]
        face_encoding_b1 = api.face_encodings(img_b1)[0]

        faces_to_compare = [
            face_encoding_a2,
            face_encoding_a3,
            face_encoding_b1]

        distance_results = api.face_distance(faces_to_compare, face_encoding_a1)
```

```
        # 0.6 is the default face distance match threshold. So we'll spot-check that the numbers returned
        # are above or below that based on if they should match (since the exact numbers could vary).
        self.assertEqual(type(distance_results), np.ndarray)
        self.assertLessEqual(distance_results[0], 0.6)
        self.assertLessEqual(distance_results[1], 0.6)
        self.assertGreater(distance_results[2], 0.6)

    def test_face_distance_empty_lists(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images', 'biden.jpg'))
        face_encoding = api.face_encodings(img)[0]

        # empty python list
        faces_to_compare = []

        distance_results = api.face_distance(faces_to_compare, face_encoding)
        self.assertEqual(type(distance_results), np.ndarray)
        self.assertEqual(len(distance_results), 0)

        # empty numpy list
        faces_to_compare = np.array([])

        distance_results = api.face_distance(faces_to_compare, face_encoding)
        self.assertEqual(type(distance_results), np.ndarray)
        self.assertEqual(len(distance_results), 0)

    def test_compare_faces(self):
        img_a1 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg'))
        img_a2 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images', 'obama2.jpg'))
        img_a3 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images', 'obama3.jpg'))

        img_b1 = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images', 'biden.jpg'))

        face_encoding_a1 = api.face_encodings(img_a1)[0]
        face_encoding_a2 = api.face_encodings(img_a2)[0]
        face_encoding_a3 = api.face_encodings(img_a3)[0]
        face_encoding_b1 = api.face_encodings(img_b1)[0]

        faces_to_compare = [
            face_encoding_a2,
```

```
            face_encoding_a3,
            face_encoding_b1]

        match_results = api.compare_faces(faces_to_compare, face_encoding_a1)

        self.assertEqual(type(match_results), list)
        self.assertTrue(match_results[0])
        self.assertTrue(match_results[1])
        self.assertFalse(match_results[2])

    def test_compare_faces_empty_lists(self):
        img = api.load_image_file(os.path.join(os.path.dirname(__file__), 'test_images',
'biden.jpg'))
        face_encoding = api.face_encodings(img)[0]

        # empty python list
        faces_to_compare = []

        match_results = api.compare_faces(faces_to_compare, face_encoding)
        self.assertEqual(type(match_results), list)
        self.assertListEqual(match_results, [])

        # empty numpy list
        faces_to_compare = np.array([])

        match_results = api.compare_faces(faces_to_compare, face_encoding)
        self.assertEqual(type(match_results), list)
        self.assertListEqual(match_results, [])

    def test_command_line_interface_options(self):
        target_string = 'Show this message and exit.'
        runner = CliRunner()
        help_result = runner.invoke(face_recognition_cli.main, ['--help'])
        self.assertEqual(help_result.exit_code, 0)
        self.assertTrue(target_string in help_result.output)

    def test_command_line_interface(self):
        target_string = 'obama.jpg,obama'
        runner = CliRunner()
        image_folder = os.path.join(os.path.dirname(__file__), 'test_images')
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')

        result = runner.invoke(face_recognition_cli.main, args=[image_folder, image_file])

        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)
```

```
    def test_command_line_interface_big_image(self):
        target_string = 'obama3.jpg,obama'
        runner = CliRunner()
        image_folder = os.path.join(os.path.dirname(__file__), 'test_images')
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama3.jpg')

        result = runner.invoke(face_recognition_cli.main, args=[image_folder, image_file])

        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)

    def test_command_line_interface_tolerance(self):
        target_string = 'obama.jpg,obama'
        runner = CliRunner()
        image_folder = os.path.join(os.path.dirname(__file__), 'test_images')
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')

        result = runner.invoke(face_recognition_cli.main, args=[image_folder, image_file, "--
tolerance", "0.55"])

        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)

    def test_command_line_interface_show_distance(self):
        target_string = 'obama.jpg,obama,0.0'
        runner = CliRunner()
        image_folder = os.path.join(os.path.dirname(__file__), 'test_images')
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')

        result = runner.invoke(face_recognition_cli.main, args=[image_folder, image_file, "--
show-distance", "1"])

        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)

    def test_fd_command_line_interface_options(self):
        target_string = 'Show this message and exit.'
        runner = CliRunner()
        help_result = runner.invoke(face_detection_cli.main, ['--help'])
        self.assertEqual(help_result.exit_code, 0)
        self.assertTrue(target_string in help_result.output)

    def test_fd_command_line_interface(self):
        runner = CliRunner()
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')
```

```
        result = runner.invoke(face_detection_cli.main, args=[image_file])
        self.assertEqual(result.exit_code, 0)
        parts = result.output.split(",")
        self.assertTrue("obama.jpg" in parts[0])
        self.assertEqual(len(parts), 5)

    def test_fd_command_line_interface_folder(self):
        runner = CliRunner()
        image_file = os.path.join(os.path.dirname(__file__), 'test_images')

        result = runner.invoke(face_detection_cli.main, args=[image_file])
        self.assertEqual(result.exit_code, 0)
        self.assertTrue("obama_partial_face2.jpg" in result.output)
        self.assertTrue("obama.jpg" in result.output)
        self.assertTrue("obama2.jpg" in result.output)
        self.assertTrue("obama3.jpg" in result.output)
        self.assertTrue("biden.jpg" in result.output)

    def test_fd_command_line_interface_hog_model(self):
        target_string = 'obama.jpg'
        runner = CliRunner()
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')

        result = runner.invoke(face_detection_cli.main, args=[image_file, "--model", "hog"])
        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)

    def test_fd_command_line_interface_cnn_model(self):
        target_string = 'obama.jpg'
        runner = CliRunner()
        image_file = os.path.join(os.path.dirname(__file__), 'test_images', 'obama.jpg')

        result = runner.invoke(face_detection_cli.main, args=[image_file, "--model", "cnn"])
        self.assertEqual(result.exit_code, 0)
        self.assertTrue(target_string in result.output)
```

4.1.5 Models code:

```
from __future__ import unicode_literals
from django.db import models

class UserManager(models.Manager):
    def validator(self, postData):
        errors = {}
        if (postData['first_name'].isalpha()) == False:
            if len(postData['first_name']) < 2:
                errors['first_name'] = "First name can not be shorter than 2 characters"

        if (postData['last_name'].isalpha()) == False:
            if len(postData['last_name']) < 2:
```

```
            errors['last_name'] = "Last name can not be shorter than 2 characters"

        if len(postData['email']) == 0:
            errors['email'] = "You must enter an email"

        if len(postData['password']) < 8:
            errors['password'] = "Password is too short!"

        return errors

class User(models.Model):
    first_name = models.CharField(max_length=255)
    last_name = models.CharField(max_length=255)
    email = models.CharField(max_length=255,default=None)
    password = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add = True)
    updated_at = models.DateTimeField(auto_now = True)
    objects = UserManager()

class personLocation(models.Model):
    name = models.CharField(max_length=255)
    national_id = models.CharField(max_length=255)
    address = models.CharField(max_length=255)
    picture = models.CharField(max_length=255)
    status = models.CharField(max_length=255)
    latitude = models.CharField(max_length=255)
    longitude = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

class Person(models.Model):
    name = models.CharField(max_length=255)
    national_id = models.CharField(max_length=255,default=None)
    address = models.CharField(max_length=255)
    picture = models.CharField(max_length=255)
    status = models.CharField(max_length=255)
    created_at = models.DateTimeField(auto_now_add=True)
    updated_at = models.DateTimeField(auto_now=True)

class File(models.Model):
  file = models.FileField(blank=False, null=False)
  remark = models.CharField(max_length=20)
      timestamp = models.DateTimeField(auto_now_add=True)
```

4.1.6 **Urls code:**

```
from django.conf.urls import url
from django.urls import path
from django.conf import settings
from django.conf.urls.static import static

from main.views import FileView
from . import views

urlpatterns = [
    url(r'^$', views.index),
    url(r'^view_admins', views.viewAdmins),
    url(r'^success$', views.success),

    url(r'^add_member', views.addMember),
    url(r'^save_member', views.saveMember),
```

```
    url(r'^view_members', views.viewMember),

    path('wanted_member/<int:member_id>/', views.wantedMember, name='wanted_member'),
    path('free_member/<int:member_id>/', views.freeMember, name='free_member'),



    url(r'^login$', views.login),
    url(r'^logout', views.logout_view),
    url(r'^detectImage', views.detectImage),
    url(r'^detectWithWebcam', views.detectWithWebcam),
    url(r'^upload', FileView.as_view(), name='file-upload'),

    url(r'^spotted_members', views.spottedMembers),
    path('member_location/<int:member_id>/', views.viewMemberLocation, name='member_location'),
    path('found_member/<int:member_id>/', views.foundMember, name='found_member'),

    url(r'^reports', views.viewReports),



]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

4.1.6 Views Code:

```
import this
from django.core.files.storage import FileSystemStorage
from django.shortcuts import render, HttpResponse, redirect
from django.contrib import messages
import bcrypt
import face_recognition
from PIL import Image, ImageDraw
import numpy as np
import cv2
from rest_framework.views import APIView
from rest_framework.parsers import MultiPartParser, FormParser
from rest_framework.response import Response
from rest_framework import status
from .serializers import FileSerializer
from django.contrib.auth import logout


from main.models import User, Person, ThiefLocation


class FileView(APIView):
    parser_classes = (MultiPartParser, FormParser)

    def post(self, request, *args, **kwargs):
        file_serializer = FileSerializer(data=request.data)
        if file_serializer.is_valid():
            file_serializer.save()
            return Response(file_serializer.data, status=status.HTTP_201_CREATED)
```

```
        else:
            return Response(file_serializer.errors, status=status.HTTP_400_BAD_REQUEST)


def index(request):
    return render(request, 'session/login.html')


def addPerson(request):
    return render(request, 'home/add_user.html')


def addMember(request):
    return render(request, 'home/add_member.html')


def logout_view(request):
    logout(request)
    messages.add_message(request, messages.INFO, "Successfully logged out")
    return redirect(index)


def viewAdmins(request):
    users = User.objects.all()
    context = {
        "users": users
    }
    return render(request, 'home/view_admins.html', context)


def savePerson(request):
    errors = User.objects.validator(request.POST)
    if len(errors):
        for tag, error in errors.iteritems():
            messages.error(request, error, extra_tags=tag)
        return redirect(addPerson)

    hashed_password = bcrypt.hashpw(request.POST['password'].encode(), bcrypt.gensalt())
    user = User.objects.create(
        first_name=request.POST['first_name'],
        last_name=request.POST['last_name'],
        email=request.POST['email'],
        password=hashed_password)
    user.save()
    messages.add_message(request, messages.INFO, 'User successfully added')
    return redirect(viewAdmins)


def saveMember(request):
    if request.method == 'POST':
        citizen = Person.objects.filter(national_id=request.POST["national_id"])
        if citizen.exists():
            messages.error(request, "Member with that ID already exists")
            return redirect(addMember)
        else:
            myfile = request.FILES['image']
            fs = FileSystemStorage()
            filename = fs.save(myfile.name, myfile)
            uploaded_file_url = fs.url(filename)
```

```
            person = Person.objects.create(
                name=request.POST["name"],
                national_id=request.POST["national_id"],
                address=request.POST["address"],
                picture=uploaded_file_url[1:],
                status="Free"
            )
            person.save()
            messages.add_message(request, messages.INFO, "Member successfully added")
            return redirect(viewMember)


    def viewMember(request):
        citizens = Person.objects.all()
        context = {
            "citizens": citizens
        }
        return render(request, 'home/view_citizenz.html', context)


    def wantedMember(request, member_id):
        wanted = Person.objects.filter(pk=member_id).update(status='Wanted')
        if (wanted):
            person = Person.objects.filter(pk=member_id)
            thief = ThiefLocation.objects.create(
                name=person.get().name,
                national_id=person.get().national_id,
                address=person.get().address,
                picture=person.get().picture,
                status='Wanted',
                latitude='20202020',
                longitude='040404040'
            )
            thief.save()
            messages.add_message(request, messages.INFO, "User successfully changed status to wanted")
        else:
            messages.error(request, "Failed to change the status of the citizen")
        return redirect(viewMember)


    def freeMember(request, member_id):
        free = Person.objects.filter(pk=member_id).update(status='Free')
        if (free):
            messages.add_message(request, messages.INFO, "User successfully changed status to Found and Free from
Search")
        else:
            messages.error(request, "Failed to change the status of the citizen")
        return redirect(viewMember)


    def spottedMembers(request):
        thiefs = ThiefLocation.objects.filter(status="Wanted")
        context = {
            'thiefs': thiefs
        }
        return render(request, 'home/spotted_persons.html', context)
```

```
    def foundMember(request, member_id):
        free = ThiefLocation.objects.filter(pk=member_id)
        freectzn = personLocation.objects.filter(national_id=free.get().national_id).update(status='Found')
        if(freectzn):
            thief = ThiefLocation.objects.filter(pk=member_id)
            free = Person.objects.filter(national_id=thief.get().national_id).update(status='Found')
            if(free):
                messages.add_message(request, messages.INFO, "Thief updated to found, congratulations")
            else:
                messages.error(request, "Failed to update thief status")
        return redirect(spottedMembers)


    def viewMemberLocation(request, member_id):
        thief = ThiefLocation.objects.filter(pk=member_id)
        context = {
            "thief": thief
        }
        return render(request, 'home/loc.html', context)


    def viewReports(request):
        return render(request, "home/reports.html")


    def login(request):
        if (User.objects.filter(email=request.POST['login_email']).exists()):
            user = User.objects.filter(email=request.POST['login_email'])[0]
            hashed = bcrypt.hashpw(request.POST['login_password'], bcrypt.gensalt())
            if (bcrypt.checkpw(request.POST['login_password'].encode(), user.password.encode())):
                request.session['id'] = user.id
                request.session['name'] = user.first_name
                request.session['surname'] = user.last_name
                messages.add_message(request, messages.INFO, 'Welcome to real time face detection system ' +
user.first_name+' '+user.last_name)
                return redirect(success)
            else:
                messages.error(request, 'Oops, Wrong password, please try a diffrerent one')
                return redirect('/')
        else:
            messages.error(request, 'Oops, That ID do not exist')
            return redirect('/')


    def success(request):
        user = User.objects.get(id=request.session['id'])
        context = {
            "user": user
        }
        return render(request, 'home/welcome.html', context)


    def detectImage(request):
        # This is an example of running face recognition on a single image
        # and drawing a box around each person that was identified.

        # Load a sample picture and learn how to recognize it.

        # upload image
```

```
    if request.method == 'POST' and request.FILES['image']:
        myfile = request.FILES['image']
        fs = FileSystemStorage()
        filename = fs.save(myfile.name, myfile)
        uploaded_file_url = fs.url(filename)
        #person=Person.objects.create(name="Swimoz",user_id="1",address="2020
Nehosho",picture=uploaded_file_url[1:])
        # person.save()

    images = []
    encodings = []
    names = []
    files = []

    prsn = Person.objects.all()
    for crime in prsn:
        images.append(crime.name+'_image')
        encodings.append(crime.name+'_face_encoding')
        files.append(crime.picture)
        names.append(crime.name + ' ' + crime.address)

    for i in range(0, len(images)):
        images[i] = face_recognition.load_image_file(files[i])
        encodings[i] = face_recognition.face_encodings(images[i])[0]

    # Create arrays of known face encodings and their names
    known_face_encodings = encodings
    known_face_names = names

    # Load an image with an unknown face
    unknown_image = face_recognition.load_image_file(uploaded_file_url[1:])

    # Find all the faces and face encodings in the unknown image
    face_locations = face_recognition.face_locations(unknown_image)
    face_encodings = face_recognition.face_encodings(unknown_image, face_locations)

    # Convert the image to a PIL-format image so that we can draw on top of it with the Pillow library
    # See http://pillow.readthedocs.io/ for more about PIL/Pillow
    pil_image = Image.fromarray(unknown_image)
    # Create a Pillow ImageDraw Draw instance to draw with
    draw = ImageDraw.Draw(pil_image)

    # Loop through each face found in the unknown image
    for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
        # See if the face is a match for the known face(s)
        matches = face_recognition.compare_faces(known_face_encodings, face_encoding)

        name = "Unknown"

        # If a match was found in known_face_encodings, just use the first one.
        # if True in matches:
        #     first_match_index = matches.index(True)
        #     name = known_face_names[first_match_index]

        # Or instead, use the known face with the smallest distance to the new face
        face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
        best_match_index = np.argmin(face_distances)
        if matches[best_match_index]:
            name = known_face_names[best_match_index]
```

```
            # Draw a box around the face using the Pillow module
            draw.rectangle(((left, top), (right, bottom)), outline=(0, 0, 255))

            # Draw a label with a name below the face
            text_width, text_height = draw.textsize(name)
            draw.rectangle(((left, bottom - text_height - 10), (right, bottom)), fill=(0, 0, 255), outline=(0, 0, 255))
            draw.text((left + 6, bottom - text_height - 5), name, fill=(255, 255, 255, 255))

        # Remove the drawing library from memory as per the Pillow docs
        del draw

        # Display the resulting image
        pil_image.show()
        return redirect('/success')

        # You can also save a copy of the new image to disk if you want by uncommenting this line
        # pil_image.save("image_with_boxes.jpg")


    def detectWithWebcam(request):
        # Get a reference to webcam #0 (the default one)
        video_capture = cv2.VideoCapture(0)

        # Load a sample picture and learn how to recognize it.
        images = []
        encodings = []
        names = []
        files = []
        nationalIds = []

        prsn = Person.objects.all()
        for crime in prsn:
            images.append(crime.name+'_image')
            encodings.append(crime.name+'_face_encoding')
            files.append(crime.picture)
            names.append('Name: '+crime.name + ', ID: ' + crime.national_id+', Address '+crime.address)
            nationalIds.append(crime.national_id)

        for i in range(0, len(images)):
            images[i] = face_recognition.load_image_file(files[i])
            encodings[i] = face_recognition.face_encodings(images[i])[0]

        # Create arrays of known face encodings and their names
        known_face_encodings = encodings
        known_face_names = names
        n_id = nationalIds

        while True:
            # Grab a single frame of video
            ret, frame = video_capture.read()

            # Convert the image from BGR color (which OpenCV uses) to RGB color (which face_recognition uses)
            rgb_frame = frame[:, :, ::-1]

            # Find all the faces and face enqcodings in the frame of video
            face_locations = face_recognition.face_locations(rgb_frame)
            face_encodings = face_recognition.face_encodings(rgb_frame, face_locations)
```

```
        # Loop through each face in this frame of video
        for (top, right, bottom, left), face_encoding in zip(face_locations, face_encodings):
          # See if the face is a match for the known face(s)
           matches = face_recognition.compare_faces(known_face_encodings, face_encoding)

           name = "Unknown"

           # If a match was found in known_face_encodings, just use the first one.
           # if True in matches:
           #     first_match_index = matches.index(True)
           #     name = known_face_names[first_match_index]

           # Or instead, use the known face with the smallest distance to the new face
           face_distances = face_recognition.face_distance(known_face_encodings, face_encoding)
           best_match_index = np.argmin(face_distances)
           if matches[best_match_index]:
              ntnl_id = n_id[best_match_index]
              person = Person.objects.filter(national_id=ntnl_id)
              name = known_face_names[best_match_index]+', Status: '+person.get().status

              if(person.get().status == 'Wanted'):
                 thief = ThiefLocation.objects.create(
                    name=person.get().name,
                    national_id=person.get().national_id,
                    address=person.get().address,
                    picture=person.get().picture,
                    status='Wanted',
                    latitude='20202020',
                    longitude='040404040'
                 )
                 thief.save()

           # Draw a box around the face
           cv2.rectangle(frame, (left, top), (right, bottom), (0, 0, 255), 2)

           # Draw a label with a name below the face
           cv2.rectangle(frame, (left, bottom - 35), (right, bottom), (0, 0, 255), cv2.FILLED)
           font = cv2.FONT_HERSHEY_DUPLEX
           cv2.putText(frame, name, (left + 6, bottom - 6), font, 1.0, (255, 255, 255), 1)

        # Display the resulting image
        cv2.imshow('Video', frame)

        # Hit 'q' on the keyboard to quit!
        if cv2.waitKey(1) & 0xFF == ord('q'):
           break

     # Release handle to the webcam
     video_capture.release()
     cv2.destroyAllWindows()
     return redirect('/success')
```

4.1.7 Setup Code:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

from setuptools import setup

with open('README.rst') as readme_file:
```

```
    readme = readme_file.read()

with open('HISTORY.rst') as history_file:
    history = history_file.read()

requirements = [
    'face_recognition_models>=0.3.0',
    'Click>=6.0',
    'dlib>=19.7',
    'numpy',
    'Pillow'
]

test_requirements = [
    'tox',
    'flake8==2.6.0'
]

setup(
    name='face_recognition',
    version='1.2.3',
    description="Recognize faces from Python or from the command line",
    long_description=readme + '\n\n' + history,
    author="Adam Geitgey",
    author_email='ageitgey@gmail.com',
    url='https://github.com/ageitgey/face_recognition',
    packages=[
        'face_recognition',
    ],
    package_dir={'face_recognition': 'face_recognition'},
    package_data={
        'face_recognition': ['models/*.dat']
    },
    entry_points={
        'console_scripts': [
            'face_recognition=face_recognition.face_recognition_cli:main',
            'face_detection=face_recognition.face_detection_cli:main'
        ]
    },
    install_requires=requirements,
    license="MIT license",
    zip_safe=False,
    keywords='face_recognition',
    classifiers=[
        'Development Status :: 4 - Beta',
        'Intended Audience :: Developers',
        'License :: OSI Approved :: MIT License',
        'Natural Language :: English',
        "Programming Language :: Python :: 2",
        'Programming Language :: Python :: 2.6',
        'Programming Language :: Python :: 2.7',
        'Programming Language :: Python :: 3',
        'Programming Language :: Python :: 3.3',
        'Programming Language :: Python :: 3.4',
        'Programming Language :: Python :: 3.5',
        'Programming Language :: Python :: 3.6',
    ],
    test_suite='tests',
    tests_require=test_requirements
```

Chapter 5: Conclusion

In this thesis, we discussed a real time movement tracking system that can handle person movemnet tracking problem inside a secured area smartly and more efficiently than the typically used sensor-based or fixed time slot systems by using real-time processing. Our proposed system has an advantage over other traditional systems due to the latest and simple algorithms having more accuracy used for the detection and tracking of objects of interest. Techniques used in our proposed system; Image Processing techniques to process the video, frame by frame, and included algorithms simple face recognition , which were used to detect the objects of interest; are also briefly explained included their working and importance. The purpose of increasing productivity and overcoming problems in existing solutions is being achieved by using the modern techniques. Additionally, the objectives; smooth real time tracking are attained. Hence saves time as well as the prestigious lives of patients.

Simulated results are shown for two live feeds using OpenCV, which is open-source software, easy to install, and can be used in real-time. The same algorithm can be used to have live feeds for all four lanes. Only the interfacing of the cameras would be required.

Our proposed system, RTMTSUAI, is cost-effective as it is purely made for the core purpose of serving Pakistan, any system that could be beneficial to its citizens. Else, similar solutions provided by other countries are very costly. Moreover, STCS provides an ease to adoption which can be adopted by beneficiaries, mass deployment can also be done and most importantly no product training is required.

Chapter 6: Future Work

Future milestones that need to be achieved to commercialize this project are the following.

6.1 Access of this system over the internet:

There must be some mechanism to access the whole real time movement tracking system over the internet to control it remotly . To do this some extra effort must be put to get the edge over the existing solution of real time tracking and detection systems.So if this is implemented in our project it will give our project competitive advantage over the existing systems and the institutions would be encouraged to use our system for the security purposes.

References and Work Cited

[1]S. Divya.M, Dr.S.Padmavathi "Identification of the movements of Human in a Video" International Journal of Engineering Research in Computer Science and Engineering (IJERCSE) Vol 3, Issue 3, March 2016
[2] Swati Gossain, Jagbir Gill "a novel approach to enhance object detection using integrated detection algorithms" International Journal of Computer Science and Mobile Computing, Vol.3 Issue.3, March- 2014, pg. 1018-1023
[3] Tomasz Kryjak, Marek Gorgoń "Real-Time Implementation of Moving Object Detection in Video Surveillance Systems Using Fpga" Computer Science • Vol. 12 • 2011
[4] Rupesh Kumar Rout "A Survey on Object Detection and Tracking Algorithms" Department of Computer Science and Engineering Natio nal Institute of Technology Rourkela Rourkela – 769 008, India
[5] Prithviraj Banerjee and Somnath Sengupta "Human Motion Detection and Tracking for Video Surveillance"
[6] Kermani and Asemani "A robust adaptive algorithm of moving object detection for video surveillance" EURASIP Journal on Imag e and Video Processing2014,2014:27
[7] Payal Panchal, Gaurav Prajapati, Savan Patel, Hinal Shah and Jitendra Nasriwala "A Review on Object Detection and Tracking M ethods" International Journal For Research In Emerging Science And Technology, Volume-2, Issue-1, January-2015
[8] Nameirakpam Dhanachandra, Khumanthem Manglem and Yambem Jina Chanu "Image Segmentation usingK -means Clustering Algorithm and Subtractive Clustering Algorithm" Eleventh International Multi-Conference on Information Processing-2015 (IMCIP-2015)
[9] Jacinto Nascimento, Member, IEEE Jorge Marques "Performance evaluation of object detection algorithms for video surveillance" IST/ISR, Torre Norte, Av. Rovisco Pais, 1049-001, Lisboa Portugal.
[10] David Moore "A real-world system for human motion detection and tracking" California Institute of Technology June 5, 2003
[11] Nishu Singla "Motion Detection Based on Frame Di fference Method" International Journal of Information & Computation Technology. ISSN 0974-2239 Volume 4, Number 15 (2014), pp. 1559-1565

[12] Jianpeng Zhou and Jack Hoang “Real Time Robust Human Detection and Tracking System” I3DVR International Inc 780 Birchmount Road, Unit 16, Scarborough, Ontario, Canada M1K 5H4
[13] A survey of activity recognition and understanding the behaviour in video surveillance, A.R.Revathi and dhananjay kumar
[14] Development of Human Tracking in Video Surveillance System for Activity Analysis, Neelam V. Puri and Prof. P. R. Devale, IOSR Journal of Computer Engineering (IOSRJCE) ISSN: 2278-0661 Volume 4, Issue 2 (Sep.-Oct. 2012), PP 26-30 www.iosrjournals.org
[15] Visual Surveillance of Human Activity,Larry Davis, Sandor Fejes et.al, ACCV-98, Mumbai-India, Material Subject to ACCV Copy-Rights
[16] Nikhil Sharma, Niharika Mehta, Region Filling and Object Removal by Exempeler Based Image Inpainting, International Journal of Inventive Engineering and Sciences (IJIES) ISSN: 2319–9598, Volume-1, Issue-3, February 2013 26.
[17] Mr. Deepjoy Das and Dr. Sarat Saharia,”Implementation and Performance Evalu ation of Background Subtraction Algorithms”, International Journal on Computational Sciences & Applications (IJCSA) Vol.4, No.2, April 2014.
[18] Thomas Andzi-Quainoo Tawiah,” Video Content Analysis For Automated Detection And Tracking Of Humans In cctv Surveillance Applications”, School of Engineering and Design Brunel University, August 2010.
[19] Navneet Dalal, Bill Triggs, and Cordelia Schmid,” Human Detection Using Oriented Histograms of Flow and Appearance”, April 2006.
[20] Seema Kumari, Manpreet Kaur, Birmohan Singh,”Detection And Tracking Of Moving Object In Visual Surveillance System”, International Journal of Advanced Research in Electrical Electronics and Instrumentation Engineering (IJAREEIE) Vol. 2, Issue 8, August 2013.
[21] J.Joshan Athanesious, P.Suresh,"Systematic Survey on Object Tracking Methods in Video”, International Journal of Advanced Research in Computer Engineering & Technology (IJARCET), Volume 1, Issue 8, October 2012.
[22] Rupali S.Rakibe, Bharati D.Patil,”Background Subtraction Algorithm Based Human Motion Detection -published at: "International Journal of Scientific and Research Publications (IJSRP), Vol. 3, Issue 5, May 2013 Edition"
1.