A Mini Project Report On

REAL TIME SMOKE SURVEILLANCE IN PUBLIC ARENA USING YOLOV4

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

In

Department of Computer Science and Engineering

By

Ambajipeta Nandini	21241A0503
Arutla Siri Chandana	21241A0505
Jeddi Akshara	21241A0521
Tanuku Vaibhavi	21241A0556
Vanguri Akanksha	21241A0559

Under the Esteemed guidance of

S.Ritwika

Assistant Professor

Department of Computer Science and Engineering

GOKARAJU RANGARAJU INSTITUTE OF ENGINEERING AND TECHNOLOGY

(Autonomous)

Bachupalli, Kukatpally, Hyderabad,Telangana,India,500090

2023-2024

GOKARAJU RANGARAJU
INSTITUTE OF ENGINEERING AND TECHNOLOGY
(Autonomous)

CERTIFICATE

This is to certify that the major project entitled "**Real Time Smoke Surveillance in Public Arena using YOLOV4**" is submitted by **Ambajipeta Nandini (21241A0503), Arutla Siri Chandana (21241A0505), Jeddi Akshara (21241A0521), Tanuku Vaibhavi (21241A0556), Vanguri Akanksha (21241A0559)** in fulfillment of the award of the degree in BACHELOR OF TECHNOLOGY in Computer Science and Engineering during the academic year **2023-2024.**

INTERNAL GUIDE	HEAD OF THE DEPARTMENT
S.Ritwika	**Dr. B. SANKARA BABU**
Assistant Professor	**HOD, Professor**

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

Many people helped us directly and indirectly to complete our project successfully. We would like to take this opportunity to thank one and all. First, we wish to express our deep gratitude to our internal guide **S.Ritwika, Assistant Professor**, Department of CSE for her support in the completion of our project report. We wish to express our honest and sincere thanks to **Dr. B. Sankara Babu, HOD,** Department of CSE, and to our principal **Dr. J. Praveen** for providing the facilities to complete our mini project. We would like to thank all our faculty and friends for their help and constructive criticism during the project completion phase. Finally, we are very much indebted to our parents for their moral support and encouragement to achieve goals.

Ambajipeta Nandini 21241A0503
Arutla Siri Chandana 21241A0505
Jeddi Akshara 21241A0521
Tanuku Vibhavi 21241A0556
Vanguri Akanksha 21241A0559

DECLARATION

We hereby declare that the Mini Project entitled "**Real Time Smoke Surveillance in Public Arena using YOLOV4**" and is submitted in the partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Computer Science and Engineering from **Gokaraju Rangaraju Institute of Engineering and Technology.** The results embodied in this project have not been submitted to any other university or Institution for the award of any degree or diploma.

Ambajipeta Nandini	**21241A0503**
Arutla Siri Chandana	**21241A0505**
Jeddi Akshara	**21241A0521**
Tanuku Vibhavi	**21241A0556**
Vanguri Akanksha	**21241A0559**

	Table of Contents	
Chapter	TITLE	Page No
	Abstract	1
1	Introduction	2
2	Literature Survey	5
3	System Requirements	6
	3.1 Software Requirements	6
	3.2 Hardware Requirements	12
	3.3 Data Set	12
4	Propose Approach , Modules Description, and UML Diagrams	14
	4.1 Proposed Approach	14
	4.2 Modules	15
	4.3 UML Diagrams	18
5	Implementation, Experimental Results &Test Cases	20
6	Conclusion and Future Scope	46
7	References	47
	Appendix Snapshot Of the Result	49

	LIST OF FIGURES	
Fig No	**Fig Title**	**Page No**

1	Register Account	16
2	Login with registered account	16
3	Detection Module	17
4	Alert Module	17
5	WebApp Module	18
6	Use Case Diagram	18
7	Class Diagram	19
8	Sequence Diagram	19
9	User Registration	35
10	Registration of Account	35
11	Reset Password	36
12	Password reset message	36
13	User can reset password by clicking link	37
14	Login with registered account	37
15	Invalid Credentials	38
16	Enter a camera location and Mail address	38
17	Detection of smoking people in public places	39
18	Alert message received to the mail that user log into	39
19	More detailed view of smoker from the link in mail	40
20	Alert to web app	40
21	Login with registered account	41
22	Enter Camera Location and mobile number	41
23	Smoking people are detected and frame saved to the server	42
24	Alert of smoke detection is sent to mobile number by which user login	43
25	Alert is also updated to Web App	43
26	Login	49
27	Enter camera location and mail address	49

28	Smoke detection and saving of frame	50
29	Alert Message to mail	50
30	Web App alert	51
31	Mobile alert	52
32	Web App alert	52

	LIST OF TABLES	
Table No	**Table Name**	**Page No**

| 1 | Test Cases | 44 |

ABSTRACT

Air pollution coming from various sources and pathways, is among the major causes of environmental deteriorations. It has unhealthy implications to the health of the public. Among the most dangerous ways to identify smoky air is when people smoke a cigarette, a habit that not only exposes them to a high level of risk but even non-smokers who are also experiencing the danger of secondhand smoke inhalation. In this paper, we seek to devise a system that shall help avoid the occurrence of all of these dangers by creating completely smoke-free environments and a healthy as well as safe environment.

Two are the directions of our work. On one hand, video frame preprocessing is done, and then the YOLOv4 model, which is quite accurate and helpful for real-time smoke detection in video streams, is utilized. This advanced model is quite a suitable one for the purpose owing to its enhanced object detection with high efficiency over video data processing. We are integrating OpenCV: Open Source Computer Vision Library for performing two folds. It would be capturing the video footages. It would be offering us an interface without hassle with the YOLOv4 model. Its rich feature-set allows the videos to be rapidly processed and hence will be most appropriate for our system.

It triggers both immediate visual analysis of smoke detection with buzzer alerts to notify nearby personnel and email alerts to relevant authorities for further action so as to have automatic real-time notifications, thereby enforcing strict no-smoking policy and smoke-free environments. The YOLOv4 real-time smoke surveillance system is a new upgrade in the area of public health monitoring, which apart from discouraging smoking at prohibited places, also opens avenues to have more effective fire detection and early warning systems in the future.

CHAPTER 1
INTRODUCTION

1.1 Existing System

Smoke detection has taken a very broad dimension as far as public health and life are concerned. In this paper, the authors have proposed a method for smoke detection from video surveillance based on convolutional neural network or CNN. Their CNN architecture comprised four convolutional layers with unequal kernel sizes, succeeded by a max-pooling layer performing the part of spatial down-sampling.

They train and test this model in a diversified dataset, containing labeled examples of smoking along with non-smoking activities. Stochastic gradient descent accompanied by Nesterov momentum, they are used to train this model. Additionally, binary cross-entropy is used as the loss function to optimize this network. A validation set is also applied during the training process, they avoid early stopping and the overfitting stage.

Their experimental results were such that the proposed CNN model managed to reach a strong accuracy of 89.51% in detecting smoking activity on the test set. Such a high value of accuracy may demonstrate the effectiveness of any proposed model as in this paper and show the potential of improving important parameters of public health and safety through smoke detection.

1.1.1 Gaps in the Existing System

Lack of Real-Time Processing: A number of systems are not in a position to process video in real time. Real-time processing is very important so that immediate detection and action is taken upon infringement of the smoking ban in public places.

Limited Notification Mechanisms: Most current systems lack various features that are automatically triggered so as to notify the authority in charge, thereby reducing a response's effectiveness.

Poor User Interface Design: A poor user interface can reduce the effective monitoring and management of support system administrators over the detection system.

Inadequate Speed and Efficiency: Most available models of detection usually fail in speed optimization. Hence, there is a resultant delay in detecting smoking activities, objectively vital operations in real-time.

1.2 Proposed System

The proposed real-time smoke surveillance system monitors the public arenas by leveraging the YOLOv4 model for smoke detection. This model enjoys high efficiency and accuracy in its object detection capability that fits video data in real-time. OpenCV has been integrated to capture video in order to work seamlessly with the YOLOv4 model for real-time and high-quality smoke detection.

The system alerts, as soon as it detects smoke, in accordance with generating quick response. Immediate buzzer triggers staff/patrons in the vicinity so as to sound the alarm for immediate action or parade in the first instance. In the second instance, an email warning is triggered and displayed to relevant authorities on the occurrence and where this has taken place to take appropriate measures. This double-alarm process enables both localized and bureaucratic responses to be rapidly coordinated and facilitates no-smoking policies and smoke-free ambient conditions inside public places.

1.2.1 Advantages over Existing System

Real-Time Detection: YOLOv4 along with OpenCV detects smoke in real-time. Therefore, timely action against risks in public areas is possible.

Improved Speed of Training: In case of integration, the speed of the model gets improved for training. This is required for managing huge video data and thus ensuring optimal performance for smoke detection.

Selection of Region of Interest: Selection of ROI accurately helps in focusing more on sensitive regions. This is how better smoke detection efficiency could become possible in highly trafficked/sensitive areas.

Alerting Mechanisms: Email, SMS, or calls produce substantial alerts to the authority immediately post smoke detection, thus helping effective and prompt intervention and enforcement of compliance.

Log-in and Reporting: Log-in and integrated reporting provide overall oversight and historical analysis for smoke detection events from an administrative perspective.

Better Data and Video Processing : OpenCV provides better efficiency in data and video processing, thereby facilitating faster analysis as well as accurate detection of smoke during real-time situations.

Reduced False Alarms: YOLOv4, due to its advanced algorithms of detection, gives fewer false alarms. It identifies smoke more accurately against the rest of the visual elements, which helps optimize resource utilization and hence reduced operational disruption.

Faster and Reliable Detection: YOLOv4 uses fully developed algorithms that can detect smoke rapidly and reliably under different environmental conditions, thereby ensuring that the system covers various scales, hence uptimes.

User-Friendly Interface: YOLOv4 implementations feature intuitive interfaces for administrators, enabling easy monitoring with live video streaming and real-time alerts, thereby enhancing system usability and interaction.

CHAPTER 2

LITERATURE SURVEY

The paper "Cigarette CNN: A Convolutional Neural Network for Detecting Cigarette Smoking Activity" is focused on the objective of developing a smoking detection model for application in public places meant for real-time monitoring. Smoke detection, in general, may maintain public health and safety, especially in atmospheres hosting special fire risks. During the advent of deep learning, Convolutional Neural Networks have been very vital in image recognition tasks. This paper details some preliminary studies on the application of working mechanisms of CNN in activity detection of cigarette smoking by video feed analysis. In this paper it is tried to evolve a smoke detection system capable of telling more accurately and effortlessly under different settings.

CNN-Based Smoker Classification and Detection in Smart City: This paper proposes a solution for the detection of smokers across no-smoking areas of a smart city, which is based on a Convolutional Neural Network model. This paper presents an AI-based framework for the detection of smokers.

Energy-Efficient Deep CNN for Smoke Detection in Foggy IoT Environment: The authors review the state-of-the-art designs about CNNs for smoke detection in foggy IoT environments and propose an energy-efficient deep CNN framework that would enable the early detection of smoke.

CHAPTER 3

SYSTEM REQUIREMETS

3.1 Software Requirements

At the core of the Real-Time Smoke Surveillance System is the robust dynamic integration of cutting-edge technologies: advanced video processing library – OpenCV; Google Colab for training the model; QtDesigner for creating UI pages; LabelImg for bounding boxes; real-time notification system through email and SMS; a full-fledged web application for continuous monitoring and analysis; and SQL Server. Arming the system with such a fearsome combination makes this very effective and efficient in detecting and reporting smoking incidents in public arenas.

3.1.1 SQL Server

It is a powerful relational database management system. The objective of the same is to store and recover data in accordance with the satisfaction of remaining software; it therefore forms a part of daily operation in many businesses.

SQL Server helps to sustain different transaction processing, business information, and analytical tasks, indicating that software suited for application both in small and large organizations. One of its unique characteristics or features is the seamless integration that it allows with SQL Server.

Another primary characteristic of SQL Server, perhaps, would be that it was developed with principal focus on Microsoft ecosystem integration. It has integrations inbuilt for some other Microsoft products like Excel, Power BI, and Azure cloud services. It provides options for strong security in integrity and confidentiality, high-availability/disaster recovery features in a box.

Well, the dialect of the SQL language in use with SQL Server is called Transact-SQL. In contrast, SQL is slightly but very functionally extended to support procedural programming, capable of developing the most advanced and complex stored procedures, triggers, and functions directly in a database.

3.1.2 OpenCV

OpenCV has essentially been a computer vision library, licensed at no cost, initially developed by Intel and then by Willow Garage and Itseez. With its all-in-one system, it is there to provide any tools; therefore, it indeed makes one feel much more comfortable when working on computer vision and machine learning, especially with involvement in more than 2,500

optimized algorithms. These algorithms can be applied in various functions, such as object detection, image segmentation, face recognition, and motion tracking. Large community support, resources, tutorials identify plenty of developers and researchers.

Video processing via OpenCV fits in here because it has a dedicated set of tools to handle video data. It supports the reading and writing of files and streams in a number of formats; enables interface to a variety of types of cameras, and is therefore relevant to a number of real-time applications. There are inbuilt facilities for frame by frame processing, background subtraction, object tracking, and analysis of optical flow. These are synchronous fields of key importance in video surveillance, sport analytics, and autonomous driving, with applications in augmented reality.

3.1.3 Qt Designer

Qt Designer is a GUI designer for building graphical user interfaces. The system allows a developer to quickly create the GUI without thousands of lines of code; he can do it literally by dragging and dropping widgets into a form from a list of available ones and then connect their signals and slots. Another important reason Qt Designer is in wide usage among Qt developers for the creation of desktop and embedded applications is its very user-friendly interface with great integration into Qt's Development Environment.

However, the most essential thing about Qt Designer is that it features a scheme for generating XML-based UI files, .ui, declaring layout along with properties of user interface features. Next, these .ui files would be compiled into C++ code with the help of Qt's meta-object compiler, MOC. Additionally, this will help a developer realize a seamless integration procedure between the created UI design and application logic. Such separation of the user interface design from the application logic enhances maintainability, which makes easier collaboration between designers and developers possible.

3.1.4 LabelImg

LabelImg is an open-source graphical image annotation tool annotating images for making machine learning and computer vision projects ready for object detection tasks. It provides an interface through which the annotators can, in a friendly way, draw bounding boxes around the relevant objects from the images. This kind of annotation will majorly be used for learning and training models of object detection.

It supports several formats of annotations, including Pascal VOC XML format and YOLO format. Find a place in almost all machine learning platforms. LabelImg supports several classes labeling in a single image and also provides very effective facilities for editing, deleting, and navigating with the annotations.

LabelImg is cross-platform compatible—Windows, macOS, and Linux—since it is written with Python and Qt. It is greatly used in academia and industry due to its simplicity, reliability, or even rock-solidness in processing large datasets for object detection tasks. Hence, this tool is still under very active maintenance and updating, and helpful with a contributing community.

3.1.5 Google Colab

Colab, otherwise referred to as Google Colaboratory, is a peck of free cloud-based platforms that Google provides for users to run and write collaborative Python code. It thereby mainly finds itself quite useful to machine learning practitioners and researchers in that; pairing up with Google Drive has used it to remarkable GPU support—an area accelerating the training phase of their machine learning models at rather high speed.

Probably, the biggest benefit of Google Colab is that it offers high-performance GPUs—Tesla K80 and T4 GPUs—and TPUs. All of these are provided to the user free of cost, which is immensely huge computing power. This would drastically bring down the time taken for a model's training. Specifically, this capability is very helpful while training deep learning models involving large datasets and complex architectures.

Moreover, Google Colab will provide a smooth coding environment by including deep learning-related libraries in line with needs: TensorFlow, PyTorch, and Keras. Other libraries can also be installed using pip commands, making the whole development of the project flexible. Since all Colab notebooks get directly stored to Google Drive, they can easily be shared with many others from one's team or even the entire research community.

3.1.6 Postman

Postman is an ultra-powerful, extendable tool for improving the process of API testing and development. You will work with APIs, especially ease in sending requests to a server that describes its responses for the testing and debugging of APIs with great ease. You will be using

all the HTTP methods like GET, POST, and PUT. You will also learn to DELETE many other versatile ways to hit a large number of different endpoints or APIs.

Developers use Postman to get information from the client side by developing requests that specify an HTTP method—mostly either GET or POST—an endpoint URL, headers, parameters, and data if needed. The information is supposed to be represented as JSON payloads or form data. For this purpose, Postman then opens an intuitive interface to fill in these details directly or import them from previous requests or collections. Once configured, clicking submits the "Send" query. Postman formats this response into headers, status codes, and the actual data returned.

It allows the creation of collections, which group requests saved and shared together. Now, having similar requests in a place makes it much easier to test APIs and document them. This type of collection is saved for re-use, hence saving the developer some time for improving productivity and consistency while interacting with APIs.

A request fired off from Postman will hit an API endpoint specified in the request URL on the server side. The request subsequently gets treated as its logic on the server, which might involve fetching data from a database, computation, or updating of records. The response will come back to Postman from the server for processing, and now it goes ahead to display the response data accompanied by part of the metadata. This bidirectional communication flow, on the other hand, enables testing functionality of the APIs and debugging of problems but will ensure that an API works as expected in all scenarios. With all those features rich and a very user-friendly interface, Postman has grown to be in the heart of any software development pipeline—the core tool used by API developers and testers alike.

3.1.7 Python and its libraries

Python is a high-level, interpreted programming language enriched with simplicity, readability, and versatility. Guido van Rossum was its designer. This language was released in the year 1991, and its guiding philosophy emphasizes the readability and simplicity of code with significant indentation to add to the visual structure of code. Therefore, this makes it an excellent language from a fresher to an experienced developer. It has rich libraries at its disposal and supports several programming paradigms: procedural, object-oriented, and functional programming. That is what results in a flexible toolset for a wide range of tasks. The

standard library is versatile to the extent that it gives access to modules covering almost everything from file I/O and system calls through to web services and data manipulation. Speaking of that, Python can be a strike when developing rapid applications.

It has comprehensive community support and popularity, making there a rich environment of third-party libraries and frameworks that increase its potential. Web development is covered through the Django and Flask frameworks in highly robust, scalable applications. NumPy, Pandas, and Matplotlib stand at one's beck and call in data science and analytics to perform intricate data manipulation and visualization. While machine learning and AI-related functionality is dealt with by libraries like Tensorflow and PyTorch, it can easily be integrated with other languages and technologies and itself be used for different industries like technology, finance, healthcare, and education—what proved Python to be a versatile and vastly used programming language. With its features, it powered itself into wide adoption and huge growth.

3.1.7.1 Libraries installed

- **Django**

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It was developed initially for efficiently troubling the hassle of creating complex database-driven websites; thus, it comes with many features inbuilt, including an ORM system to work with databases, an admin interface to easily manage content, form handling, and a stout URL routing system. It comes "batteries included," propagating robust security features, scalability, and reusability components out of the box. By emphasizing the Tcl/Tk philosophy—reuse and rapid development—and pairing it with maintainability, Django becomes a Python web framework for building web applications for very broad ranges of industries: CMS, e-commerce systems, social network sites, news sites, and scientific computing portals.

Other installed libraries

Asgiref, astroid, boto3, botocore, certify, chardet, django-extensions, django-filter, django-storages, djangorestframework, gunicorn, idna, isort, jmespath, lazy-object-proxy,

Markdown, mccabe, mysqlclient, Pillow, pydotplus, PyJWT, pylint, pyparsing, python-dateutil, pytz, requests, s3transfer, six, sqlparse, toml, twilio, urllib3, whitenoise, wrapt

3.1.8 Integrated Development Environment (IDE)

2.1.8.1 Visual Studio Code

Visual Studio Code, or VS Code, is nothing less than the source code editor par excellence—a tool that's deeply held in the hearts of most developers worldwide for its simplicity of power. The developer behind VSCode just happens to be Microsoft. This is the reason for massive adoption: the text editor supports huge variability in programming language and platforms, all while exhibiting very low overhead for this functionality. So on any platform, Windows, macOS, the Linux distro—experience is just fascinating.

However, it is the wide range of extensibility options that makes VSCode have a large ecosystem of add-ins that allow any edit-developer to tailor their environment to what they really need, from language support tools to debugging tools, and from version control integration to a lot more. Furthermore, these plug-ins provide many flexibilities to developers to build up their governmental workflows in order to increase their productivity and work efficiency.

Another really cool feature in VSCode is intelligent editing. In other words, making use of IntelliSense, context-sensitive code completion, syntax highlighting to enable developers to work faster, thereby avoiding some very common pitfalls of that time—software development tools that will not only speed up the process of code writing but will also prevent errors from appearing in the first place, hence maintainable codebases.

Although light in weight, it doesn't compromise with IDE features. Obviously, it fits very well with many tools and services which by now have become an integral part of most software development workflows. On the tool side, VSCode has a set of very nice integrated terminals and integration with the Git versioning system, fine debugging capabilities, rising up to meet all kinds of developer needs, projects, and languages.

What is more, the VSCode community is very active and is always getting better. An innovative platform is being reached wherein new plug-ins can be made every single day, hence, getting feedbacks and sharing ways or tricks. In this respect, in a way, VSCode largely stays the course to continue to be the most updated development tool in the 21st century since, in itself, it self-updates to satisfy the whims and caprices of the ever-changing developer requirements within the fast-changing technology world.

3.2 Hardware Requirements

Processor : GPU for faster training of model

Memory (RAM): A minimum 4GB of RAM is recommended

Hard Disk: A minimum of 250GB is recommended.

3.3 Data Set

Cigarette Smoker Dataset: This dataset was drawn from Kaggle, which is a large set prepared to back up the development process and evaluation of machine learning models in classifying a smoking and a non-smoking individual. There are 9004 images within this dataset. All pictures are very well sorted and divided into training, testing, and validation subsets—all provide a fair share for the training and testing of models.

It breaks down into the following distribution:

Training Set: This set contains 82% of the whole dataset and includes the same kind of images, either smoking or non-smoking. In this case, the smoking category makes up 4630, while the non-smoking category contains 2818. Since this holds a great proportion of the dataset, it turns into very fertile ground for the training of machine learning models with respect to learning how to generalize the features characterizing smoking and non-smoking people.

Testing Set: The testing set accounts for only 9% of the dataset, with a total of 772 images. Of these, 395 show people smoking and 377 depict non-smoking individuals. A model that has been trained has to be tested to get an unbiased estimate of how good it generalizes to completely new unseen data.

Validation Set: This set accounts for 9% of the total dataset, where it contains 784 images. For instance, this dataset would contain 401 images of persons smoking and 383 not smoking.This dataset will help in tuning the models—i.e., tuning hyperparameters so models perform best on unseen data in training.

This ensures that each set—training, testing, and validation—involves a representative sample of smoking and non-smoking. Represented in the training set were 4630 smoking images and 2818 as non-smoking, while the test case is represented by 395 smoking image and 377 as non-smoking images respectively; the validation set consisted of 401 smoking images and 383 non-smoking images respectively. It will thus remain very important to train a rather balanced model by which it would have an effective way of differentiating between a smoking and a non-smoking person in various scenarios.

CHAPTER 4

PROPOSED APPROACH, MODULE DESCRIPTION AND UML DIAGRAMS

4.1 Proposed Approach

Detecting cigarette smoking is a crucial component of public health and safety.In our research, we use real-time video feeds to detect instances of cigarette smoking in real-time, utilizing machine learning algorithm. Our system can reliably identify smoking scenarios in a variety of settings because it was trained on a meticulously annotated dataset. The technology immediately notifies authorities and sounds a buzzer upon detection. Furthermore, a web application maintains records, guaranteeing public safety.

4.1.1 Dataset Preparation

We have gathered a large dataset of pictures and videos that incorporate both cigarette smoke and no smoke from Kaggale. To enhance the model's robustness, we have made sure the dataset includes a variety of backgrounds, lighting situations, and smoke patterns.We used annotation tools like LabelImg to label the dataset's regions of interest (cigarette smoke) and stored the annotations as text files with bounding box coordinates and class labels.

4.1.2 Model Training

For real-time object identification, we have chosen YOLOv4 because of its optimal speed and accuracy. We have utilized the annotated dataset to train the YOLOv4 model. Metrics like recall, accuracy, and loss are used to track training. To prevent overfitting, we conducted validation using a different validation set.

4.1.3 Real-Time Detection

For the capture of the video feeds from a camera, we developed OpenCV. This is how OpenCV applies functions for the reading and preprocessing of video frames. It also does so in such a way that each frame is passed through the YOLOv4 model for the detection of cigarette smoke and then processing the output from that model to get bounding boxes and confidence scores of objects detected.

4.1.4 Alert Mechanism

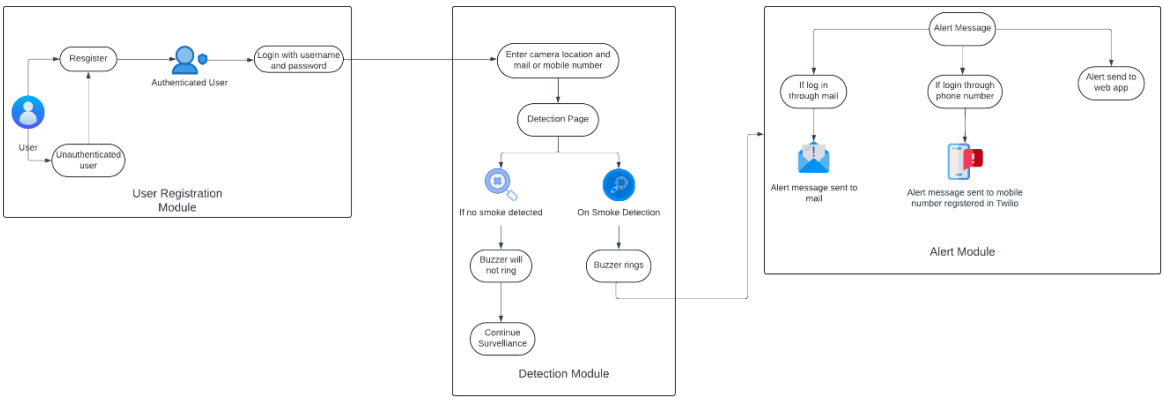
Buzzer Activation: Our system activates a buzzer upon detection. We have implemented it through the playsound library.

Notification System: In order to ensure a timely response to the observed smoke, we created our system to alert authorities via email or SMS utilizing services like Twilio for SMS and SMTP for email.

4.1.5 Web Application for Record Maintenance

We developed a web application to maintain a comprehensive record of all detection events. The backend, built with the web framework Django, handles data storage and management, while the frontend offers a user-friendly interface for monitoring and reviewing detection logs. The application stores detailed records, including timestamps, locations and snapshot images of detections, providing valuable data for analysis and compliance monitoring.

4.1.6 System Architecture



4.2 Modules

4.2.1 Registration Module

The Registration Module is a vital component of our system. It guarantees that monitoring authorities can safely create an account and access the system's features, including administrators and monitoring authorities. The gathering, verification, storage, and security protocols for user data protection are managed by this module.

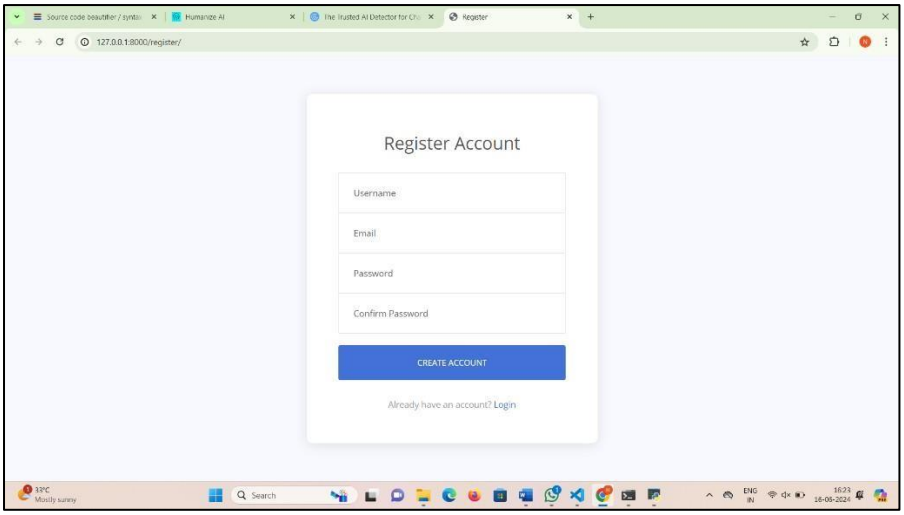


Fig 1 Register Account

4.2.2 Login Module

Our system's login module is an important aspect. It makes sure that the system, notifications, device management, and incident response are all restricted to authorized users only. To safeguard user accounts and private information, this module manages user authentication, sessions, and security protocols.

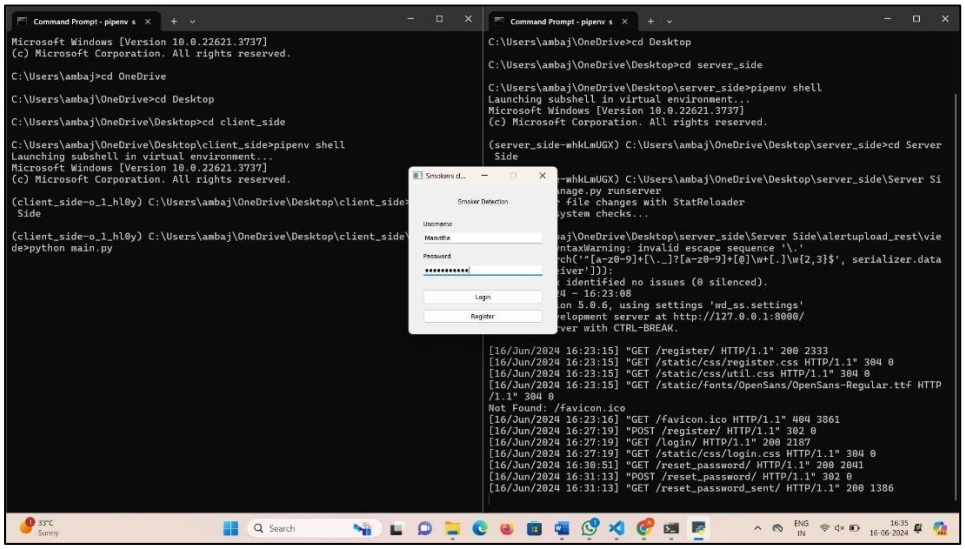


Fig 2. Login with registered account

4.2.3 Detection Module

The essential part of our system is the detection module. It uses the YOLOv4 machine learning algorithm to detect cigarette smoke in real-time from video feeds. It takes individual video stream frames, extracts them, and prepares frames for input into the YOLOv4 model by

preprocessing them. It locates and identifies cigarette smoke inside the image. Upon detection, it marks boundary boxes to visually confirm their detection of smoke.

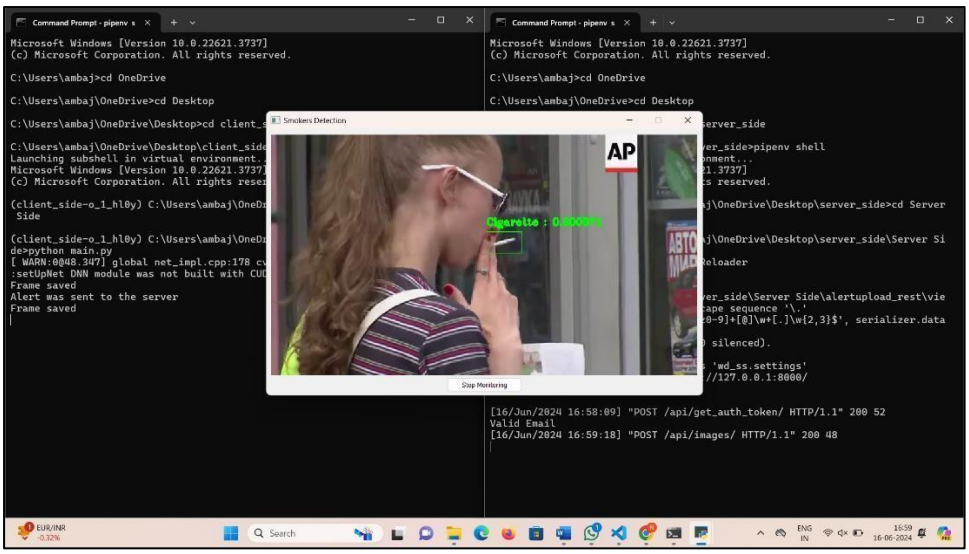


Fig 3. Detection Module

4.2.4 Alert Module:

When smoke is detected, the alert module in our system is in charge of alerting authorities. The three primary parts of this module are the email alerts, SMS notifications, and buzzer activation. When smoke is detected, a buzzer sounds and alert messages are sent to registered authorities via SMS and email.

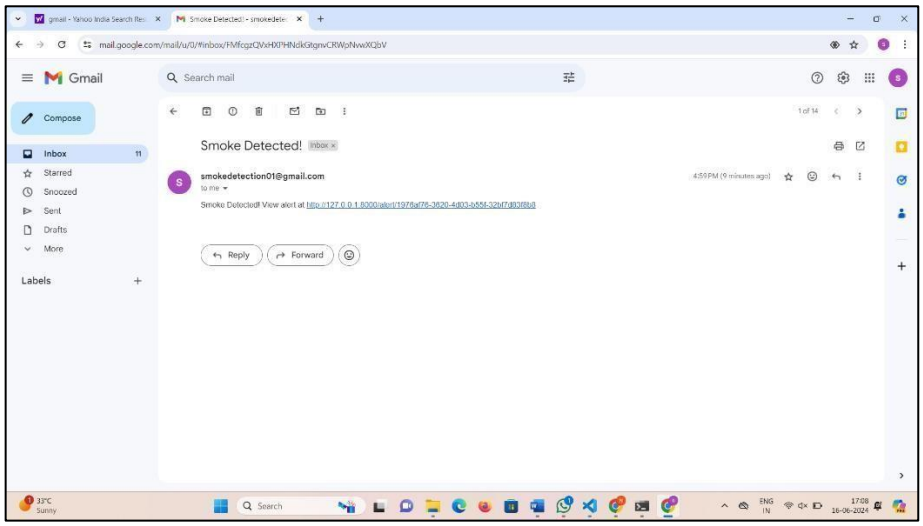


Fig 4. Alert Module

4.2.5 WebApp Module:

This module is made to keep track of and log every event of smoking that is identified. By

making sure that all identified smoking activities are properly recorded, this module enables managers to examine, evaluate, and decide what steps to take in light of previous data.

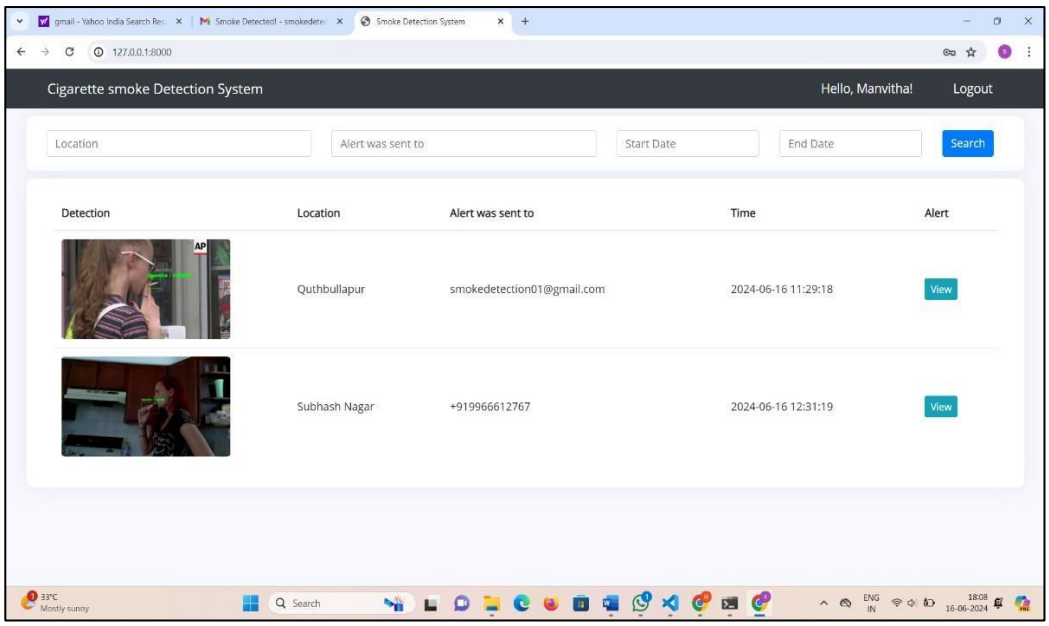


Fig 5. WebApp Module

4.3 UML DIAGRAMS

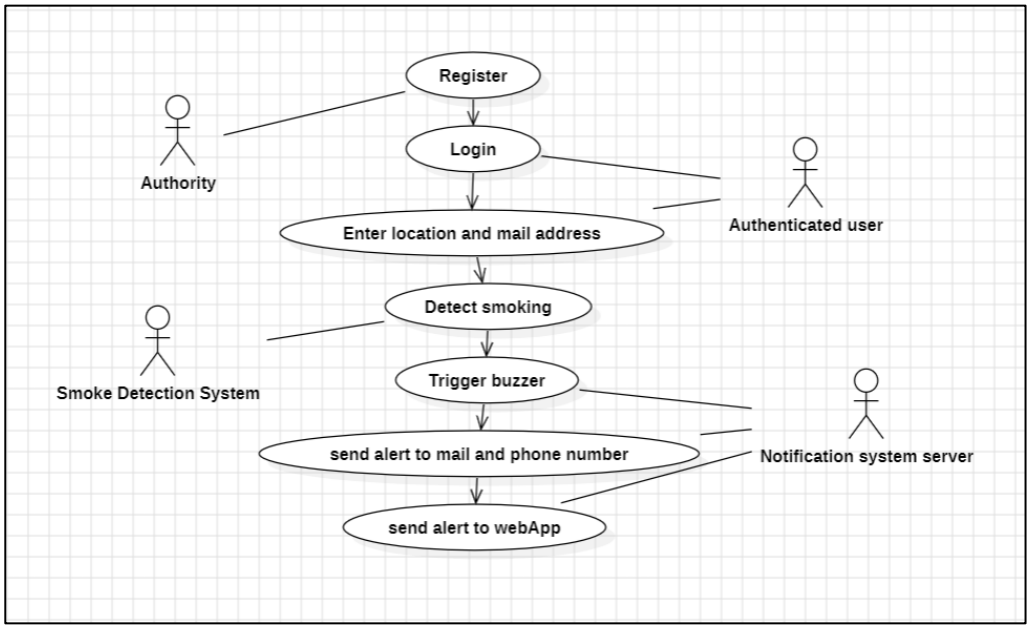


Fig 6. Use case Diagram

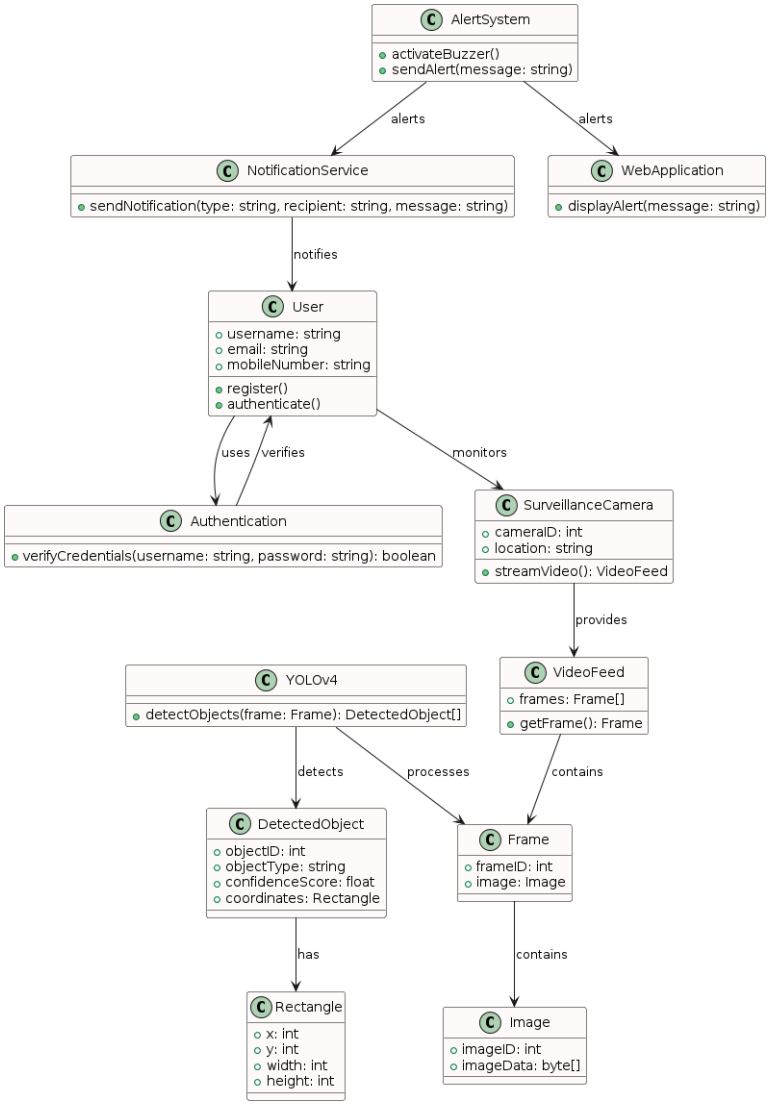


Fig 7. Class Diagram

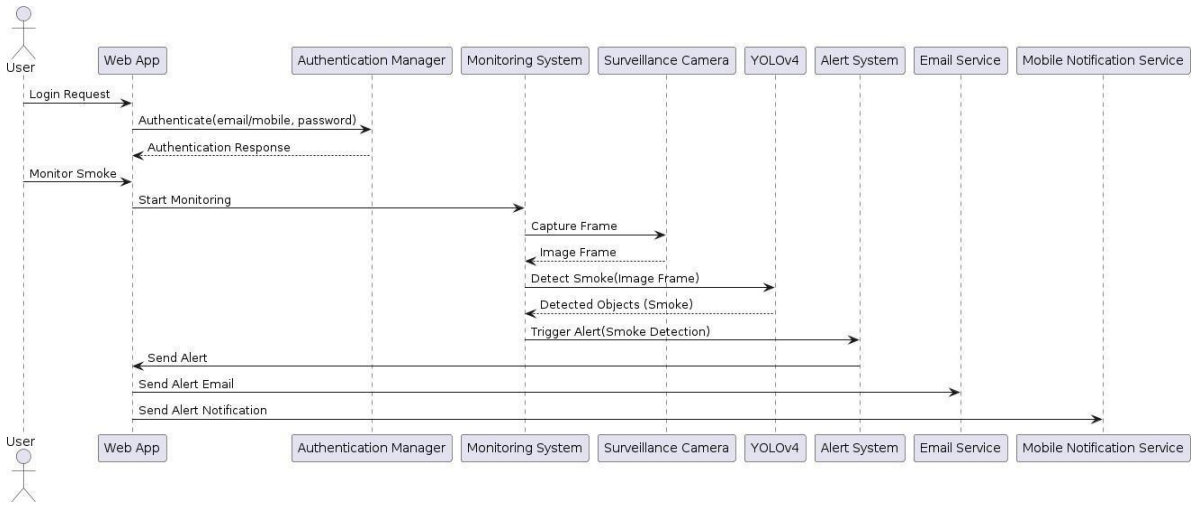


Fig 8. Sequence Diagram

CHAPTER 5

IMPLEMENTATION, EXPERIMENTAL RESULTS AND TEST CASES

5.1 Implementation

Client Side Code

Login_window.py

C:\Users\ambaj\OneDrive\Desktop\client_side\Client Side\login_window.py

```
from PyQt5.uic import loadUi
from settings_window import SettingsWindow
import webbrowser
import requests
import json
class LoginWindow(QMainWindow):
    def __init__(self):
        super(LoginWindow,self).__init__()
        loadUi('UI/login_window.ui',self)
        self.register_button.clicked.connect(self.go_to_register_page)
        self.login_button.clicked.connect(self.login)

        self.popup=QMessageBox()
        self.popup.setWindowTitle("Failed")
        # self.video_path=video_path
        self.show()

    def go_to_register_page(self):
        webbrowser.open('http://127.0.0.1:8000/register/')

    def login(self):
        try:
            url='http://127.0.0.1:8000/api/get_auth_token/'
            response=requests.post(url, data={'username':self.username_input.text(),'password':self.password_input.text()})
            json_response=json.loads(response.text)

            if response.ok:
                self.open_settings_window(json_response['token'])
            else:
                self.popup.setText("Username or Password is not correct")
                self.popup.exec_()
        except:
            self.popup.setText("Unable to access server")
            self.popup.exec_()
```

settings_window.py

```python
from PyQt5.QtWidgets import QMainWindow,QMessageBox
from PyQt5.uic import loadUi
from detection_window import DetectionWindow
class SettingsWindow(QMainWindow):
    def __init__(self,token):
        super(SettingsWindow,self).__init__() #iherits QMainWindow functions
        loadUi('UI/settings_window.ui',self) #load the ui file and map the button

        self.token=token
        # self.video_path=video_path
        self.detection_window=DetectionWindow()

        self.pushButton.clicked.connect(self.go_to_detection)

        self.popup=QMessageBox()
        self.popup.setWindowTitle("Failed")
        self.popup.setText("Fields must not be empty.")

    def displayInfo(self):
        self.show()

    def go_to_detection(self):
        if self.location_input.text() =='' or self.sendTo_input.text() =='':
            self.popup.exec_()
        else:
            if self.detection_window.isVisible():
                print('Detection window is already open!')
            else:
                self.detection_window.create_detection_instance(self.token,self.location_input.text(), self.sendTo_input.text())
                self.detection_window.start_detection()

    def closeEvent(self,event):
        if self.detection_window.isVisible():
            self.detection_window.detection.running=False
            self.detection_window.close()
            event.accept()
```

detection_window.py

```python
from PyQt5.QtWidgets import QMainWindow
from PyQt5.uic import loadUi
from PyQt5.QtCore import pyqtSlot
from PyQt5.QtGui import QImage,QPixmap
from detection import Detection
import cv2 as cv2
from PyQt5.QtCore import QTimer

class DetectionWindow(QMainWindow):
    def __init__(self):
        super(DetectionWindow,self).__init__()
        loadUi('UI/detection_window.ui',self)
        self.stop_detection_button.clicked.connect(self.close)
        self.video_capture = cv2.VideoCapture('vaping.avi')
        self.timer = QTimer(self)
        # self.timer.timeout.connect(self.displayVideo)


    def create_detection_instance(self,token,location,receiver):
        self.detection=Detection(token,location,receiver)

    @pyqtSlot(QImage)
    def setImage(self,image):
        self.label_detection.setPixmap(QPixmap.fromImage(image))

    def start_detection(self):
        # self.timer.start(1000 // 30)
        self.detection.changePixmap.connect(self.setImage)
        self.detection.start()
        self.show()

    def closeEvent(self,event):
        self.detection.running=False
        event.accept()
        self.video_capture.release()
```

detection.py

```python
from PyQt5.QtCore import QThread,Qt,pyqtSignal
from PyQt5.QtGui import QImage
import cv2 as cv2
import numpy as np
import time
import requests
from playsound import playsound
Conf_threshold=0.4
NMS_threshold=0.4
COLORS=[(0,255,0)]
class_name=[]

class Detection(QThread):   #responsible for running detection algorithm, Qthread allows to run detection in seperate
    def __init__(self,token,location,receiver):
        super(Detection,self).__init__()
        self.token=token
        self.location=location
        self.receiver=receiver
    changePixmap=pyqtSignal(QImage)
    def run(self):
        # self.running=True
        # net=cv2.dnn.readNet("weights/yolov4-obj_1000.weights","cfg/yolov4-obj.cfg")
        # classes=[]
        with open("classes.txt","r") as f:
            class_name=[cname.strip() for cname in f.readlines()]
        net=cv2.dnn.readNet("weights/yolov4-obj_1000.weights","cfg/yolov4-obj.cfg")
        net.setPreferableBackend(cv2.dnn.DNN_BACKEND_CUDA)
        net.setPreferableTarget(cv2.dnn.DNN_TARGET_CUDA_FP16)
        # layer_names=net.getLayerNames()
        # output_layers=[layer_names[i-1] for i in net.getUnconnectedOutLayers()]
        # colors=np.random.uniform(0,255,size=(len(classes),3))
        # font=cv2.FONT_HERSHEY_PLAIN
        model=cv2.dnn_DetectionModel(net)
        model.setInputParams(size=(416,416),scale=1/255,swapRB=True)
        starting_time=time.time()

        cap=cv2.VideoCapture('vaping.avi')
```

```
    while True:
        ret,frame=cap.read()
        if ret==False:
            break
        height,width,channels=frame.shape
        frame_resized=cv2.resize(frame,(416,416))
        classes,scores,boxes=model.detect(frame,Conf_threshold,NMS_threshold)
        for(classid,score,box) in zip(classes,scores,boxes):
            color=COLORS[int(classid)%len(COLORS)]
            label="%s : %f" % (class_name[classid],score)
            # print(score)
            cv2.rectangle(frame,box,color,1)
            cv2.putText(frame,label,(box[0],box[1]-10),cv2.FONT_HERSHEY_COMPLEX,0.5,color,2)
        # cv2.imshow('frame',frame)
        indexes=cv2.dnn.NMSBoxes(boxes,scores,0.8,0.3)
        for i in range(len(boxes)):
            if i in indexes:
                elapsed_time=starting_time-time.time()
                if elapsed_time<=-10:
                    starting_time=time.time()
                    self.save_detection(frame)
        rgbImage=cv2.cvtColor(frame,cv2.COLOR_BGR2RGB)
        bytesPerLine=channels * width
        convertToQtFormat=QImage(rgbImage.data,width,height,bytesPerLine,QImage.Format_RGB888)
        p=convertToQtFormat.scaled(854,480,Qt.KeepAspectRatio)
        self.changePixmap.emit(p)


        key=cv2.waitKey(1)
        if key==ord('q'):
            break
    # cap.release()
    # cv2.destroyAllWindows()
def save_detection(self,frame):
    cv2.imwrite("saved_frame/frame.jpg",frame)
```

```
    print('Frame saved')
    playsound('alertsound.mp3')
    self.post_detection()

def post_detection(self):
    try:
        url='http://127.0.0.1:8000/api/images/'
        headers={'Authorization':'Token ' + self.token}
        files={'image':open('saved_frame/frame.jpg','rb')}
        data={'user_ID':self.token,'location':self.location, 'alert_receiver':self.receiver}
        response=requests.post(url,files=files,headers=headers,data=data)

        if response.ok:
            print('Alert was sent to the server')
        else:
            print('Unable to send alert to the server')
    except:
        print('Unable to access server')
```

Server Side Code

Login.html

```
{% load static %}

<!DOCTYPE html>
<html>

<head>
    <title>Login</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.cs
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.1/css/all.css" integrity="sha
    <link rel="stylesheet" type="text/css" href="{% static '/css/login.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static '/css/util.css' %}">


</head>
<body>

    <div class="limiter">
        <div class="container-login100">
            <div class="wrap-login100 p-l-55 p-r-55 p-t-65 p-b-50">
                <form class="login100-form validate-form" method="POST" action="">
                    {% csrf_token %}
                    <span class="login100-form-title p-b-33">
                        Account Login
                    </span>

                    <div class="wrap-input100 validate-input" data-validate = "Valid email is required
                        <input class="input100" type="text" name="username" placeholder="Username">
                    </div>

                    <div class="wrap-input100 rs1 validate-input" data-validate="Password is required"
                        <input class="input100" type="password" name="password" placeholder="Password"
                    </div>

                    <div class="container-login100-form-btn m-t-20 m-b-15">
```

```
                <button class="login100-form-btn" type="submit" value="Login">
                    Sign in
                </button>
            </div>

            {% for message in messages %}
            <p id="messages">{{message}}</p>
            {% endfor %}

            <div class="text-center p-t-15 p-b-4">
                <span class="txt1">
                    Forgot
                </span>

                <a href="{% url 'reset_password' %}" class="txt2 hov1">
                    Password?
                </a>
            </div>

            <div class="text-center">
                <span class="txt1">
                    Create an account?
                </span>

                <a href="{% url 'register' %}" class="txt2 hov1">
                    Sign up
                </a>
            </div>
        </form>
    </div>
</div>
</div>
```

Register.html

```
{% load static %}

<!DOCTYPE html>
<html>

<head>
    <title>Register</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css" integrity="sha384-MCw98/SFnG
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.6.1/css/all.css" integrity="sha384-gfdkjb5BdAXd+lj+gudLWI+BXq4
    <link rel="stylesheet" type="text/css" href="{% static '/css/register.css' %}">
    <link rel="stylesheet" type="text/css" href="{% static '/css/util.css' %}">

</head>
<body>

    <div class="limiter">
        <div class="container-login100">
            <div class="wrap-login100 p-l-55 p-r-55 p-t-65 p-b-50">
                <form class="login100-form validate-form" method="POST" action="">
                    {% csrf_token %}
                    <span class="login100-form-title p-b-33">
                        Register Account
                    </span>

                    <div class="wrap-input100 validate-input" data-validate = "Valid email is required: ex@abc.xyz">
                        <input class="input100" type="text" name="username" placeholder="Username">
                    </div>

                    <div class="wrap-input100 rs1 validate-input" data-validate="Password is required">
                        <input class="input100" type="email" name="email" placeholder="Email">
                    </div>

                    <div class="wrap-input100 rs1 validate-input" data-validate="Password is required">
                        <input class="input100" type="password" name="password1" placeholder="Password">
                    </div>
```

```
                    <div class="wrap-input100 rs1 validate-input" data-validate="Password is required">
                        <input class="input100" type="password" name="password2" placeholder="Confirm Password">
                    </div>

                    <div class="container-login100-form-btn m-t-20 m-b-15">
                        <button class="login100-form-btn" type="submit" value="Register">
                            Create Account
                        </button>
                    </div>

                    {% if form.errors %}
                        {% for field in form %}
                            {% for error in field.errors %}
                            <div class="error">
                                <p class="field_label">{{field.label}}:</p>
                                <p>{{ error|escape }}</p>
                            </div>
                            {% endfor %}
                        {% endfor %}
                    {% endif %}



                    <div class="text-center p-t-15 p-b-4">
                        <span class="txt1">
                            Already have an account?
                        </span>

                        <a href="{% url 'login' %}" class="txt2 hov1">
                            Login
                        </a>
                    </div>
```

Dashboard.html

```
{%  extends 'detection/main.html' %}

{% block content %}

{% load event_tags %}

{% load static %}

<div class="limiter">
<div class="container-login100 p-b-0 p-t-0">
    <div class="wrap-login100 p-l-30 p-r-30 p-t-20 p-b-20">
        <div class="Search">
            <form method="get">
                <div class="row">

                    <div class="col">
                        <!-- Location Input -->
                        <input class="form-control" type="text"  placeholder="Location"
                        name="{{ myFilter.form.location.name }}"
                        id="{{ myFilter.form.location.id_for_label }}"
                        {% if myFilter.form.location.value != None %}value="{{ myFilter.form.location.value|stringformat:'s' }}"{% endi
                        maxlength="{{ myFilter.form.location.field.max_length }}"
                        {% if myFilter.form.location.field.required %}required{% endif %}>
                    </div>

                    <div class="col">
                        <!-- Alert was sent to Input -->
                        <input class="form-control" type="text"  placeholder="Alert was sent to"
                        name="{{ myFilter.form.alert_receiver.name }}"
                        id="{{ myFilter.form.alert_receiver.id_for_label }}"
                        {% if myFilter.form.alert_receiver.value != None %}value="{{ myFilter.form.alert_receiver.value|stringformat:'s
                        maxlength="{{ myFilter.form.alert_receiver.field.max_length }}"
                        {% if myFilter.form.alert_receiver.field.required %}required{% endif %}>
                    </div>

                    <div class="col-2">
```

```
                        <!-- Start Date Input -->
                        <input class="form-control" type="text" id="date" placeholder="Start Date" autocomplete="off"
                        name="{{ myFilter.form.start_date.name }}"
                        id="{{ myFilter.form.start_date.id_for_label }}"
                        {% if myFilter.form.start_date.value != None %}value="{{ myFilter.form.start_date.value|stringformat:'s' }}"{%
                        maxlength="{{ myFilter.form.start_date.field.max_length }}"
                        {% if myFilter.form.start_date.field.required %}required{% endif %}>
                    </div>

                    <div class="col-2">
                        <!-- End Date Input Input -->
                        <input class="form-control" type="text" id="date" placeholder="End Date" autocomplete="off"
                        name="{{ myFilter.form.end_date.name }}"
                        id="{{ myFilter.form.end_date.id_for_label }}"
                        {% if myFilter.form.end_date.value != None %}value="{{ myFilter.form.end_date.value|stringformat:'s' }}"{% endi
                        maxlength="{{ myFilter.form.end_date.field.max_length }}"
                        {% if myFilter.form.end_date.field.required %}required{% endif %}>
                    </div>

                    <div class="col-1">
                        <button class="btn btn-primary" type="submit">Search</button>
                    </div>
                  </div>
            </form>
        </div>

</div>
</div>
</div>

    <script>
```

```
<script>
 jQuery(document).ready(function($){
   var date_input=$('input[id="date"]');
   var container=$('.bootstrap-iso form').length>0 ? $('.bootstrap-iso form').parent() : "body";
   var options={
     format: 'yyyy-mm-dd',
     container: container,
     todayHighlight: true,
     autoclose: true,
   };
   date_input.datepicker(options);
 });
 </script>

<div class="limiter">
<div class="container-login100 p-t-0">
<div class="wrap-login100 p-l-40 p-r-40 p-t-25 p-b-15">

        <table class="table table-image">
            <tr>
                <th scope="col" class="top-row">Detection</th>
                <th scope="col" class="top-row">Location</th>
                <th scope="col" class="top-row">Alert was sent to</th>
                <th scope="col" class="top-row">Time</th>
                <th scope="col" class="top-row">Alert</th>
            </tr>
            {% for alert in uploadAlert %}

            <tr>
                <td class="w-25"><img style="max-width: 250px; max-height: 150x; border-radius:
                    class="img-fluid" src='http://127.0.01:8000/static/images/{{alert.image}}'/>
                <td>{{alert.location}}</td>
                <td>{{alert.alert_receiver}}</td>
                <td>{{alert.date_created|date:"Y-m-d H:i:s"}}</td>

                {% with uuid=alert.image|split:"." %}
```

Views.py

```python
from django.shortcuts import render,redirect
from django.contrib.auth.forms import UserCreationForm

from django.contrib.auth import authenticate, login, logout
from django.contrib import messages
from .forms import CreateUserForm
from rest_framework.authtoken.models import Token
from django.contrib.auth.decorators import login_required
from .filters import DetectionFilter
from .models import UploadAlert

def loginPage(request):
    if request.user.is_authenticated:
        return redirect('home')
    else:
        if request.method == 'POST':
            username = request.POST.get('username')
            password =request.POST.get('password')

            user = authenticate(request, username=username, password=password)

            if user is not None:
                login(request, user)
                return redirect('home')
            else:
                messages.info(request, 'Username OR password is incorrect')

        context = {}
        return render(request, 'detection/login.html', context)

def registerPage(request):

    if request.user.is_authenticated:
        return redirect('home')
    else:
        form = CreateUserForm()
        if request.method == 'POST':
```

```
            form = CreateUserForm(request.POST)
            if form.is_valid():
                form.save()
                user = form.cleaned_data.get('username')
                messages.success(request, 'Account was successfully created for ' + user)

                return redirect('login')

        context = {'form':form}
        return render(request, 'detection/register.html', context)

@login_required(login_url='login')
def home(request):

    token = Token.objects.get(user=request.user)

    uploadAlert = UploadAlert.objects.filter(user_ID = token)

    myFilter = DetectionFilter(request.GET, queryset=uploadAlert)
    uploadAlert = myFilter.qs

    context = {'myFilter':myFilter, 'uploadAlert':uploadAlert}

    return render(request, 'detection/dashboard.html', context)

def logoutUser(request):
    logout(request)
    return redirect('login')

def alert(request, pk):

    uploadAlert = UploadAlert.objects.filter(image = str(pk) + ".jpg")

    myFilter = DetectionFilter(request.GET, queryset=uploadAlert)
    uploadAlert = myFilter.qs
```

Models.py

```python
import os
import uuid
from django.db import models

from django.conf import settings
from django.db.models.signals import post_save
from django.dispatch import receiver
from rest_framework.authtoken.models import Token

from django.contrib.auth.models import User

# Changes uploaded file name
def scramble_uploaded_filename(instance, filename):
    """
    Scramble / uglify the filename of the uploaded file, but keep the files extension (e.g., .jpg or .png)
    :param instance:
    :param filename:
    :return:
    """
    extension = filename.split(".")[-1]
    return "{}.{}".format(uuid.uuid4(), extension)

# Data model
class UploadAlert(models.Model):
    image = models.ImageField("Uploaded image", upload_to=scramble_uploaded_filename)
    user_ID = models.ForeignKey(Token, on_delete=models.CASCADE)
    alert_receiver = models.CharField(max_length=200)
    location = models.CharField(max_length=200)
    date_created = models.DateTimeField(auto_now_add=True)

# Generate and save a token each time a user is saved in a database
@receiver(post_save, sender=settings.AUTH_USER_MODEL)
def create_auth_token(sender, instance=None, created=False, **kwargs):
    if created:
        Token.objects.create(user=instance)
```

Forms.py

```python
from django.forms import ModelForm
from django.contrib.auth.forms import UserCreationForm
from django.contrib.auth.models import User
from django import forms

# User registration form
class CreateUserForm(UserCreationForm):

    email = forms.EmailField(required=True)

    class Meta:
        model = User
        fields = ['username', 'email', 'password1', 'password2']

    # Checks if the provided email exists
    def clean_email(self):
        if User.objects.filter(email=self.cleaned_data['email']).exists():
            raise forms.ValidationError("The given email is already registered.")
        return self.cleaned_data['email']
```

serializers.py

```
from rest_framework import serializers
from detection.models import UploadAlert

# Serializer for UploadAlert Model
class UploadAlertSerializer(serializers.ModelSerializer):

    class Meta:
        model = UploadAlert
        fields = ('pk', 'image', 'user_ID', 'location', 'date_created', 'alert_receiver')
```

Views.py (alertupload)

```
from alertupload_rest.serializers import UploadAlertSerializer
from rest_framework.response import Response
from rest_framework.decorators import api_view, permission_classes
#from rest_framework.permissions import IsAuthenticated
from django.http import JsonResponse
from django.core.mail import send_mail
import re
from threading import Thread
from twilio.rest import Client
from django.conf import settings



def start_new_thread(function):
    def decorator(*args, **kwargs):
        t = Thread(target = function, args=args, kwargs=kwargs)
        t.daemon = True
        t.start()
    return decorator

@api_view(['POST'])
#@permission_classes((IsAuthenticated, ))
def post_alert(request):
    serializer = UploadAlertSerializer(data=request.data)

    if serializer.is_valid():
        serializer.save()
        identify_email_sms(serializer)

    else:
        return JsonResponse({'error':'Unable to process data!'},status=400)

    return Response(request.META.get('HTTP_AUTHORIZATION'))

def identify_email_sms(serializer):

    if(re.search('^[a-z0-9]+[\._]?[a-z0-9]+[@]\w+[.]\w{2,3}$', serializer.data['alert_receiver'])):
```

```
        print("Valid Email")
        send_email(serializer)
    elif re.compile("[+91][0-9]{10}").match(serializer.data['alert_receiver']):
        # 1) Begins with +3706
        # 2) Then contains 7 digits
        print("Valid Mobile Number")
        send_sms(serializer)
    else:
        print("Invalid Email or Mobile number")

@start_new_thread
def send_email(serializer):
    send_mail('Smoke Detected!',
    prepare_alert_message(serializer),
    'smokedetection01@gmail.com',
    [serializer.data['alert_receiver']],
    fail_silently=True,)

@start_new_thread
def send_sms(serializer):
    client = Client(settings.TWILIO_ACCOUNT_SID, settings.TWILIO_AUTH_TOKEN)

    message = client.messages.create(body=prepare_alert_message(serializer),
                                     from_=settings.TWILIO_NUMBER,
                                     to=serializer.data['alert_receiver'])


def prepare_alert_message(serializer):
    image_data= split(serializer.data['image'], ".")
    uuid=image_data[0]
    url='http://127.0.0.1:8000/alert' + uuid

    return 'Smoke Detected! View alert at ' + url

def split(value, key):
    return str(value).split(key)
```

urls.py (main)

```
from django.contrib import admin
from django.urls import path,include
#from django.conf.urls import url

urlpatterns = [
    path('admin/', admin.site.urls),
    path('',include('detection.urls')),
    path('api/', include(('alertupload_rest.urls', 'alertupload_rest'), namespace='api')),
]
```

5.2 Experimental Results

5.2.1 Respected authority registration for monitoring video stream

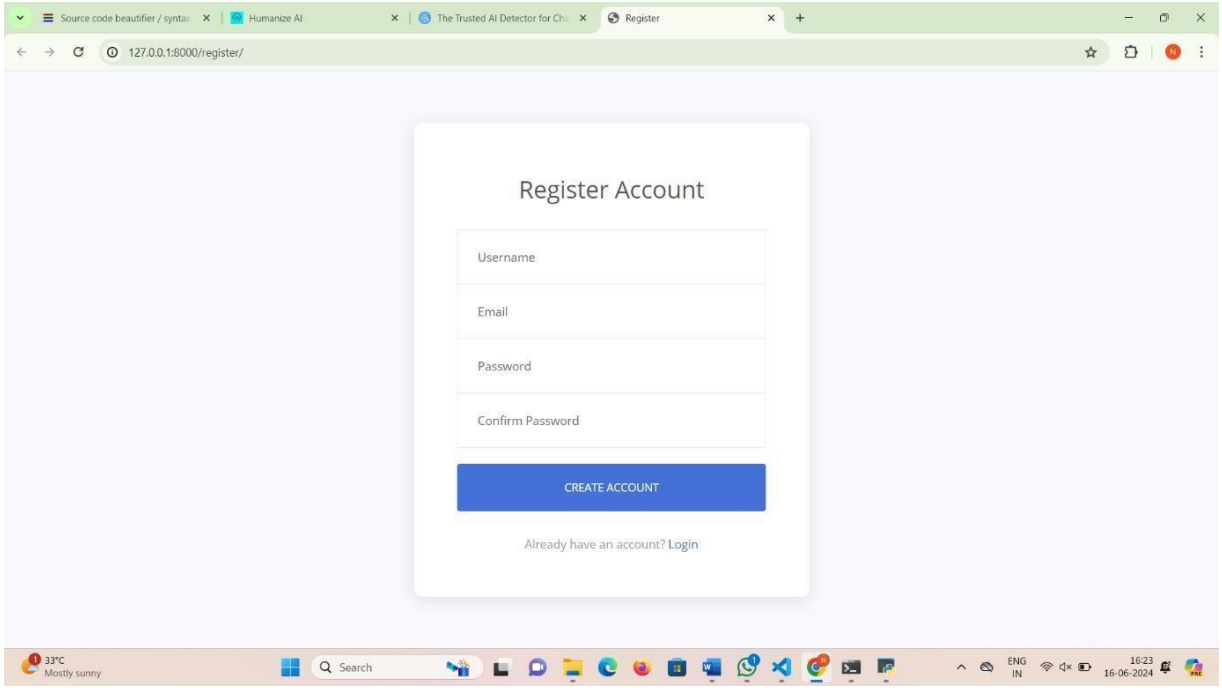


Fig 9. User Registration

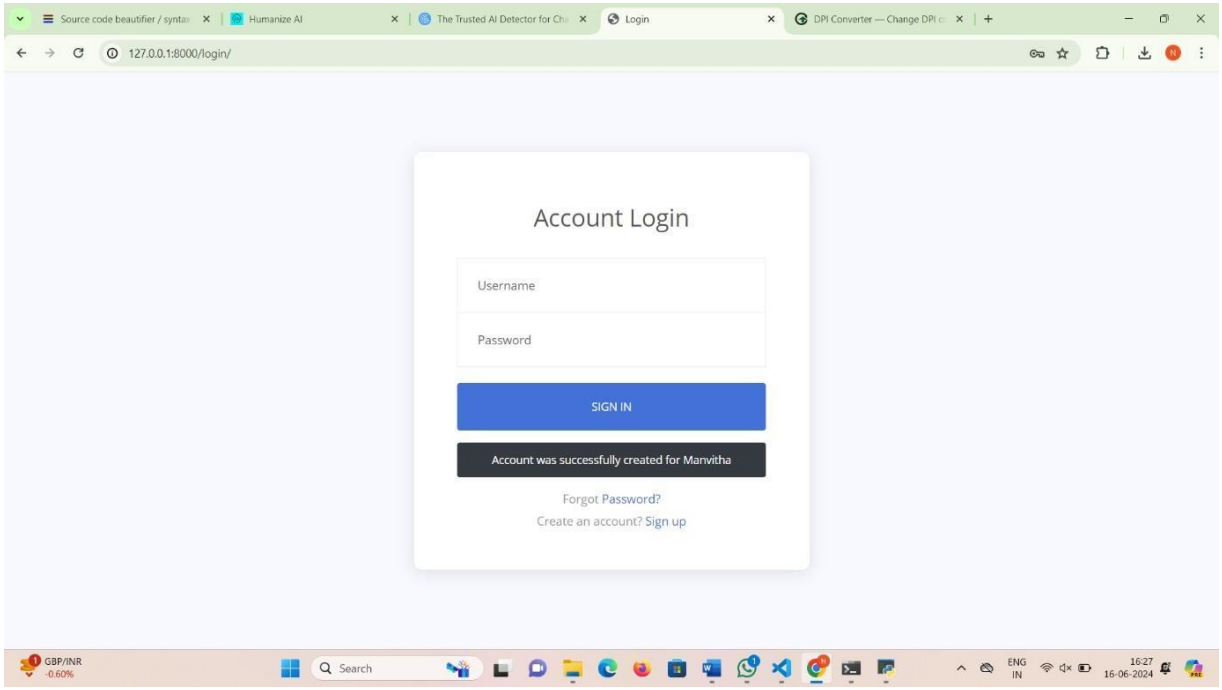


Fig 10. Registration of Account

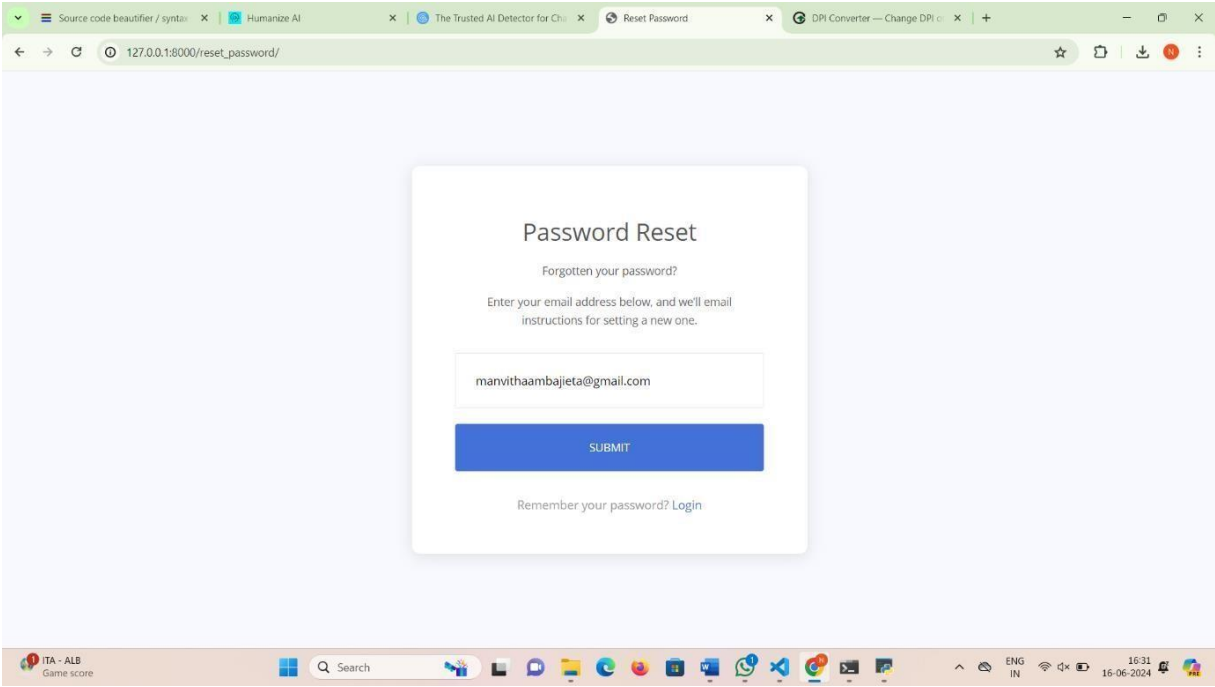


Fig 11. Reset Password

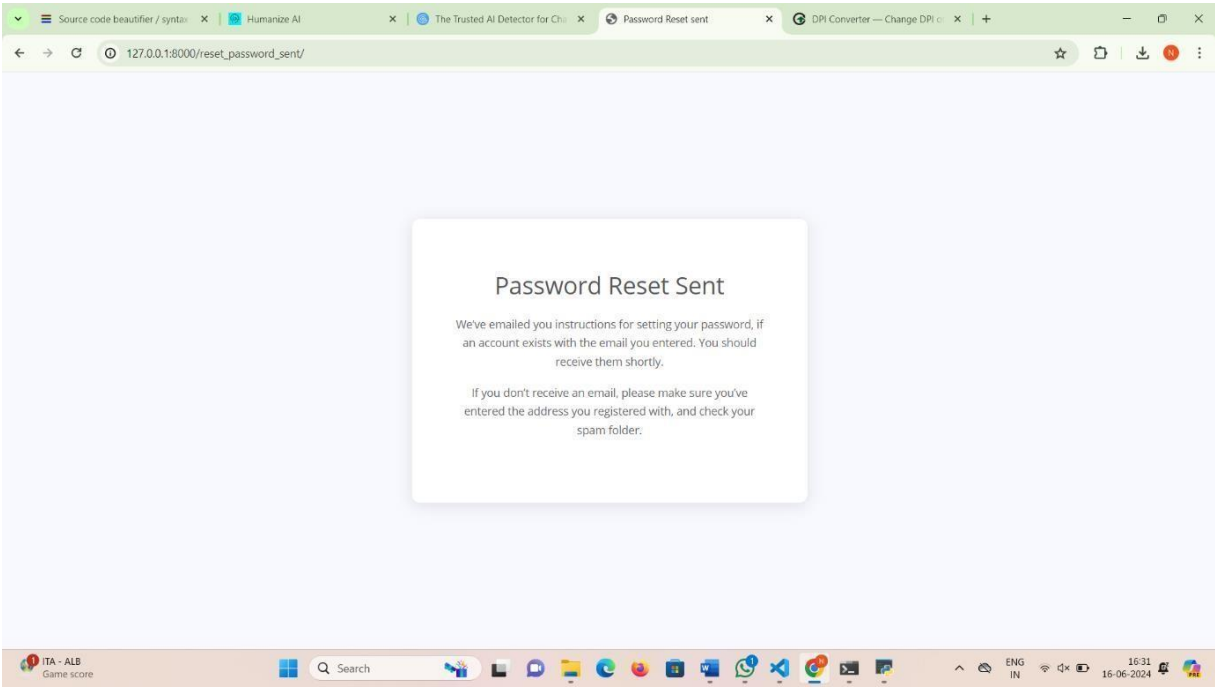


Fig 12. Password reset message

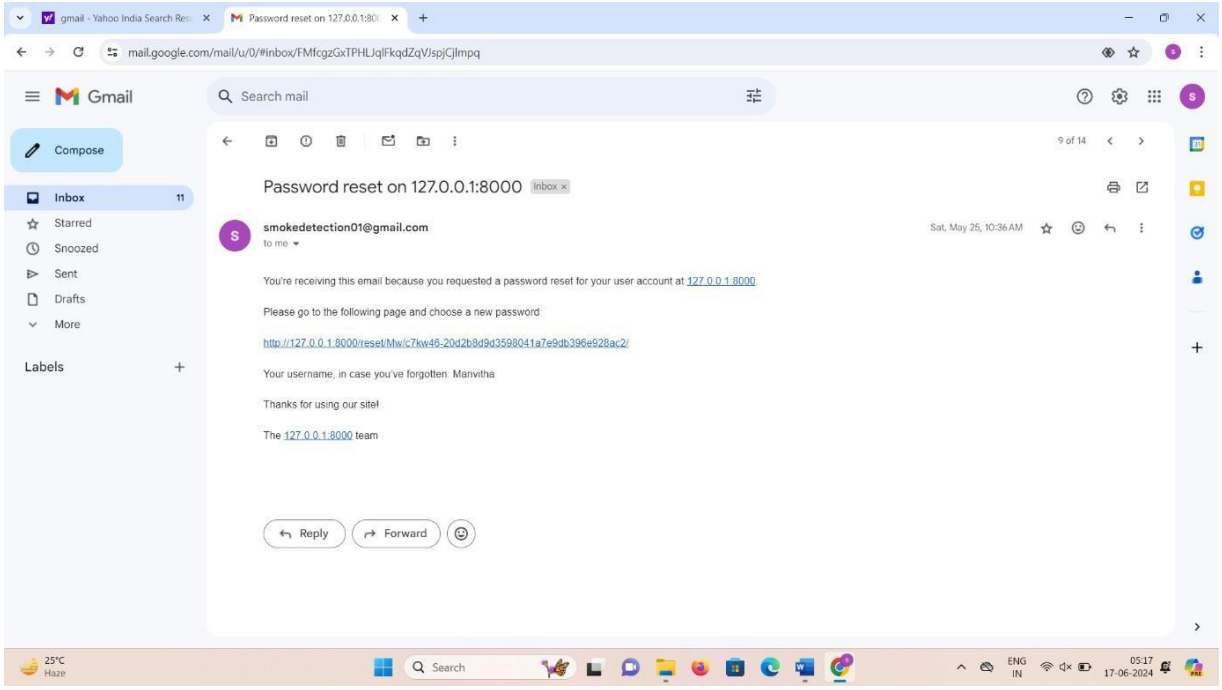


Fig 13. User can reset password by clicking link

- The registration process is very flexible to the user, enabling them to get through it in a manner they feel comfortable with. Moreover, resetting passwords can easily take place via email-based mechanisms by those who need such services. That way, the account details become manageable for the user without straining them.

5.2.2 Login and enter camera location to view portal

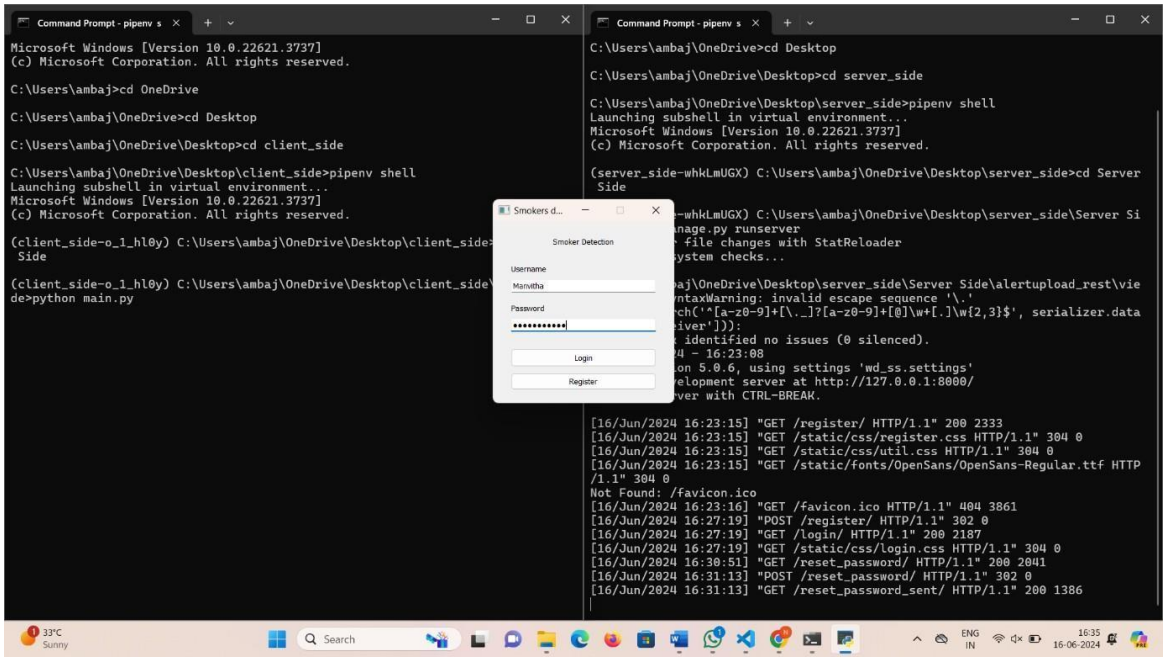


Fig 14. Login with registered account

5.2.3 Incorrect Credentials or Empty Fields

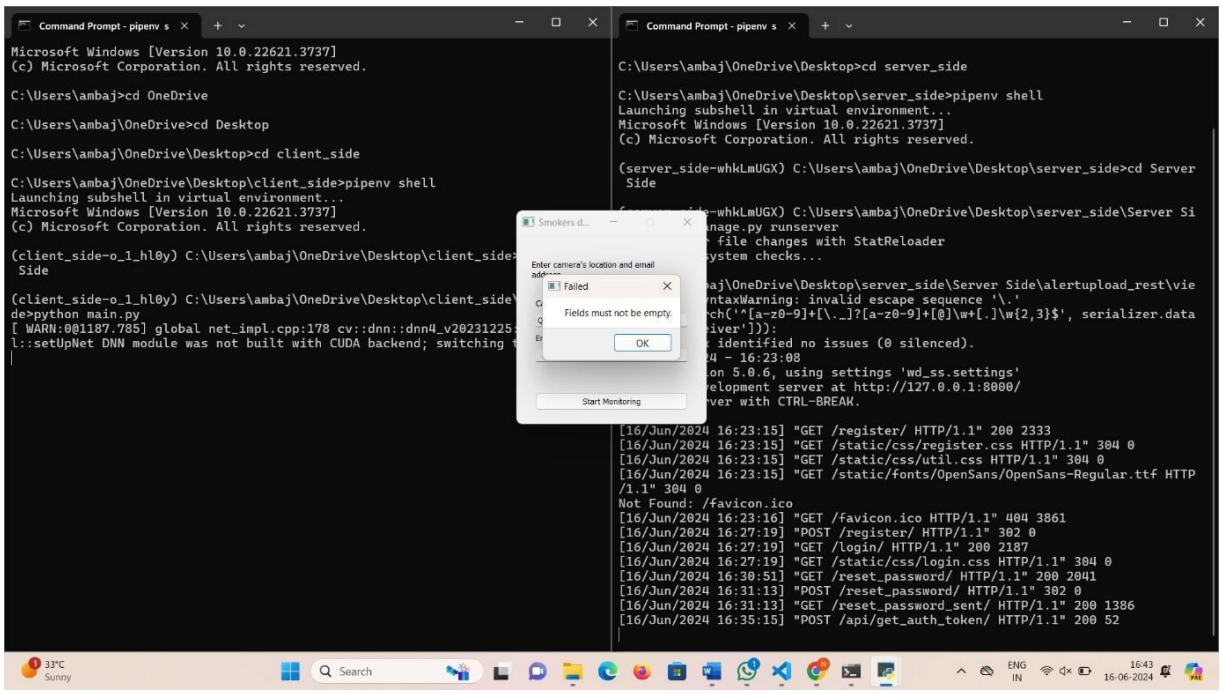


Fig 15. Invalid Credentials

- If the entered Email is invalid or if the user credentials are incorrect or fields are empty then the system will immediately prompts them to enter in the correct form

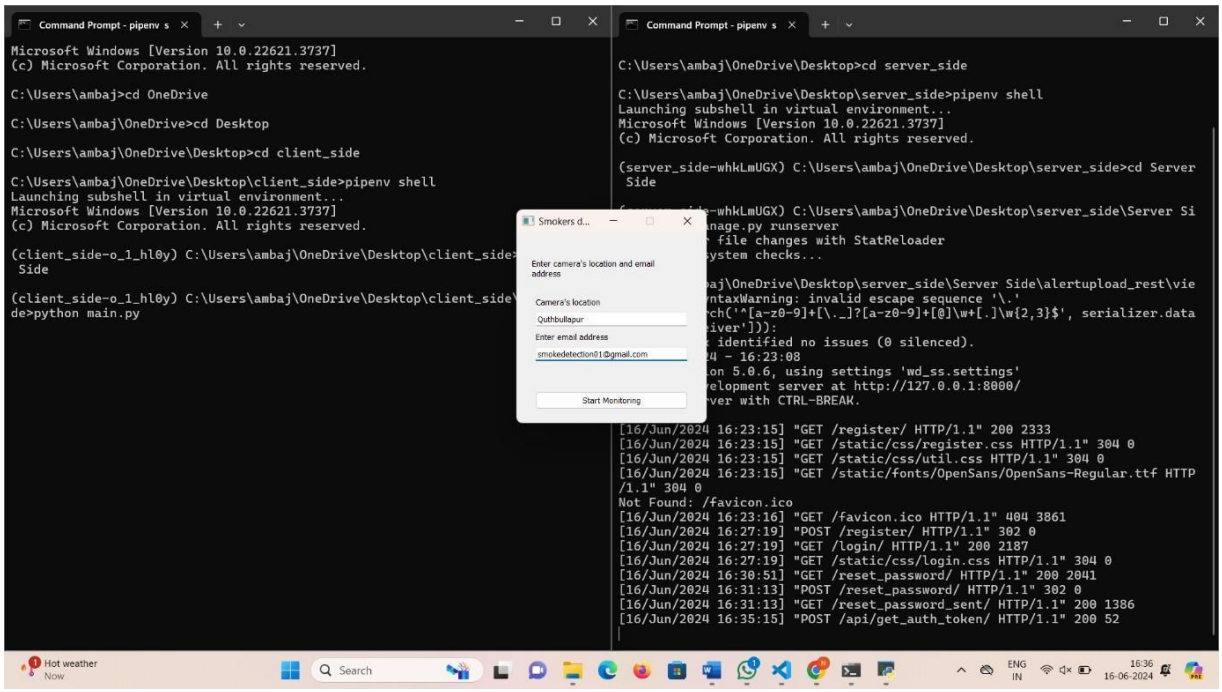


Fig 16. Enter a camera location and Mail address

5.2.4 Detection Portal

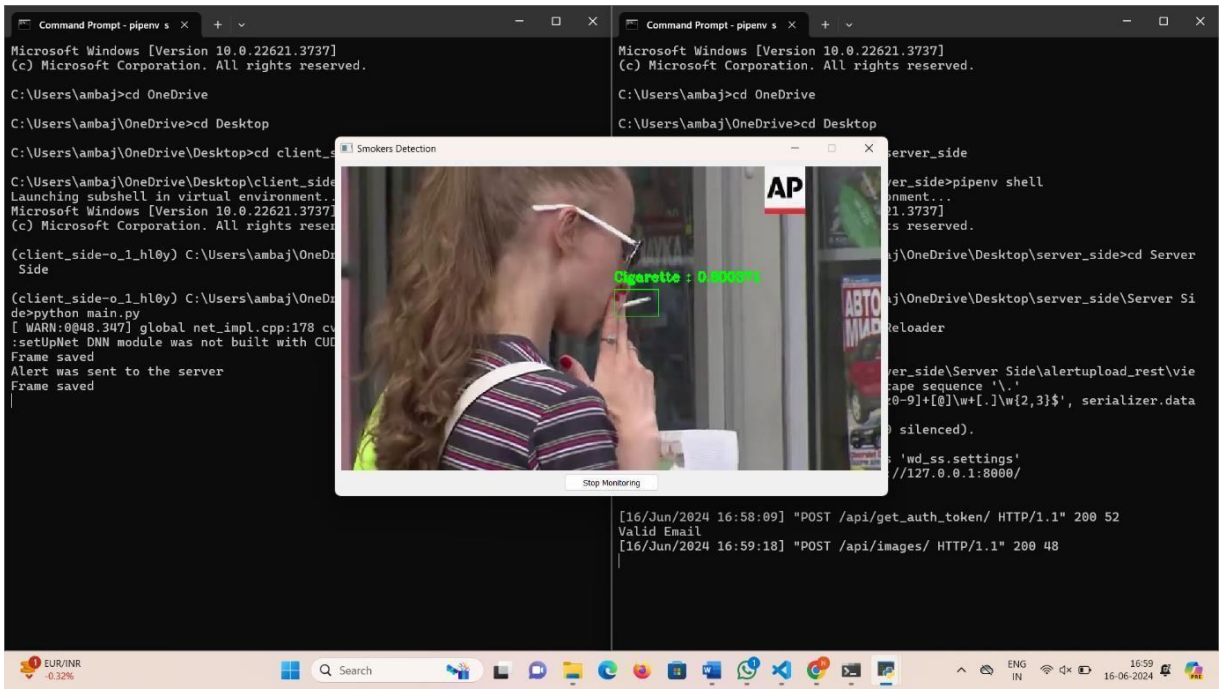


Fig 17. Detection of smoking people in public places

- Detects people who are smoking along with confidence score and immediately a buzzer rings to intimate respected authorities and also frame will be saved to the server.

5.2.5 Alert Message through Mail

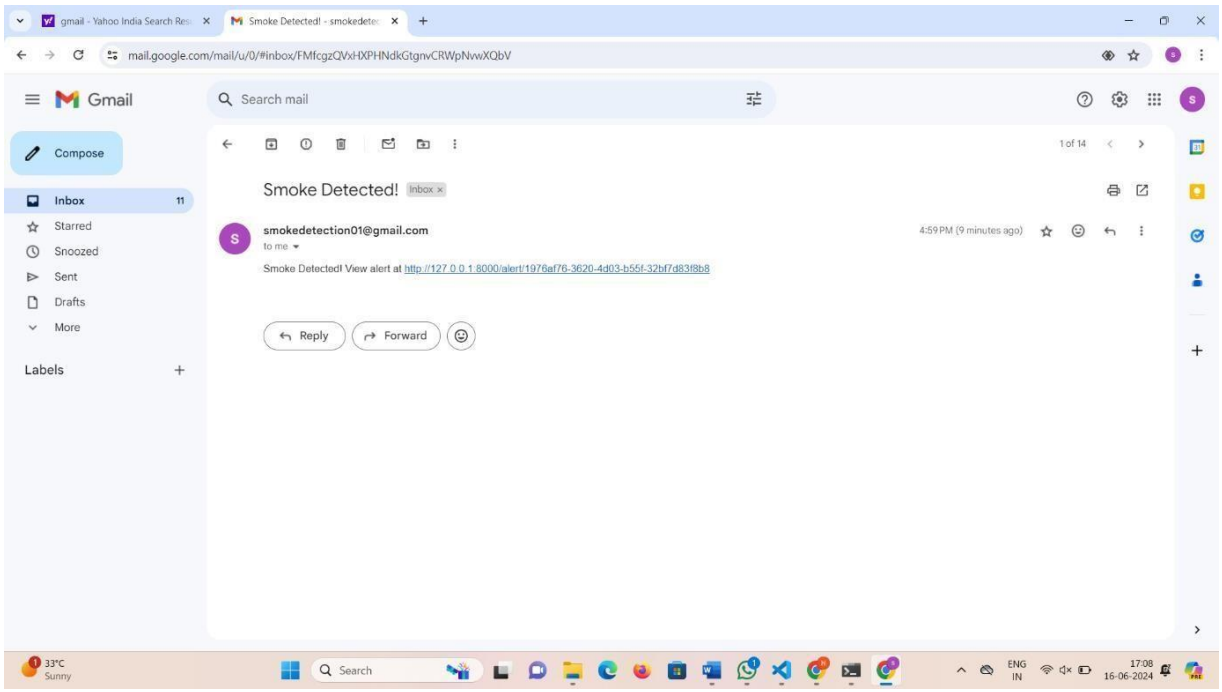


Fig 18. Alert message received to the mail that user log into

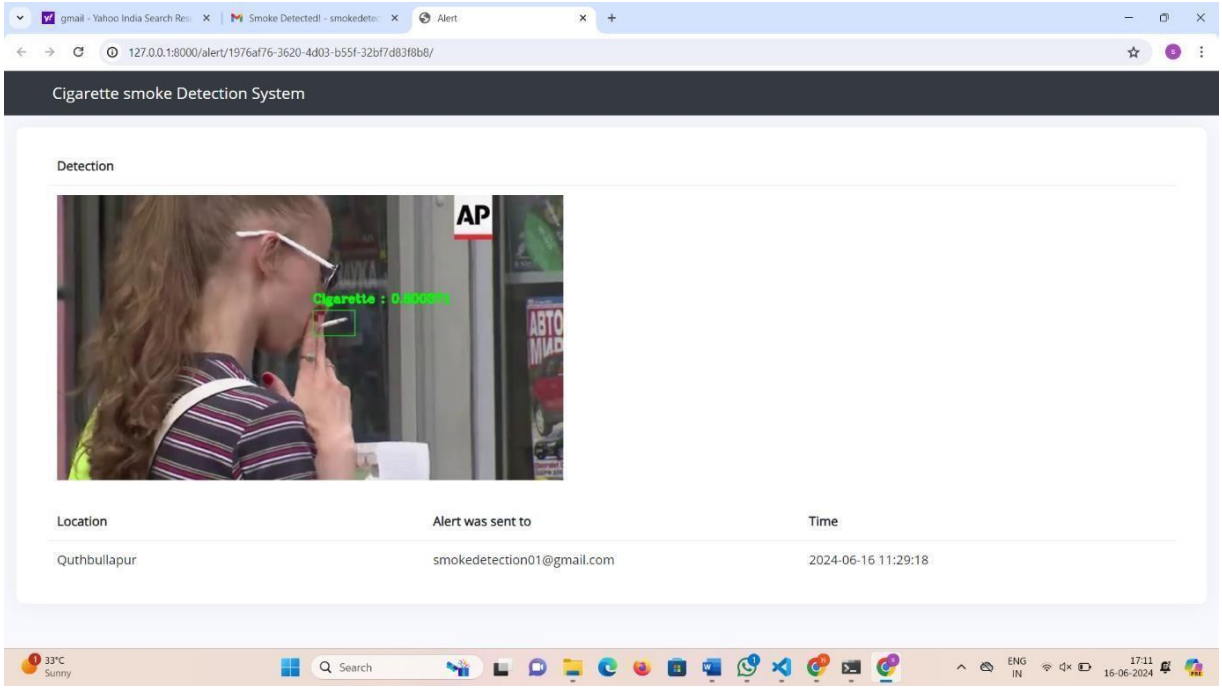


Fig 19. More detailed view of smoker from the link in mail

- It sends an email to a link for notifications with regard to smoking detections when the user is off-site. This link will have full details showing where it happened, the receiver of this alert, and the date and time when the alert was issued.

5.2.6 Alert through Web Application

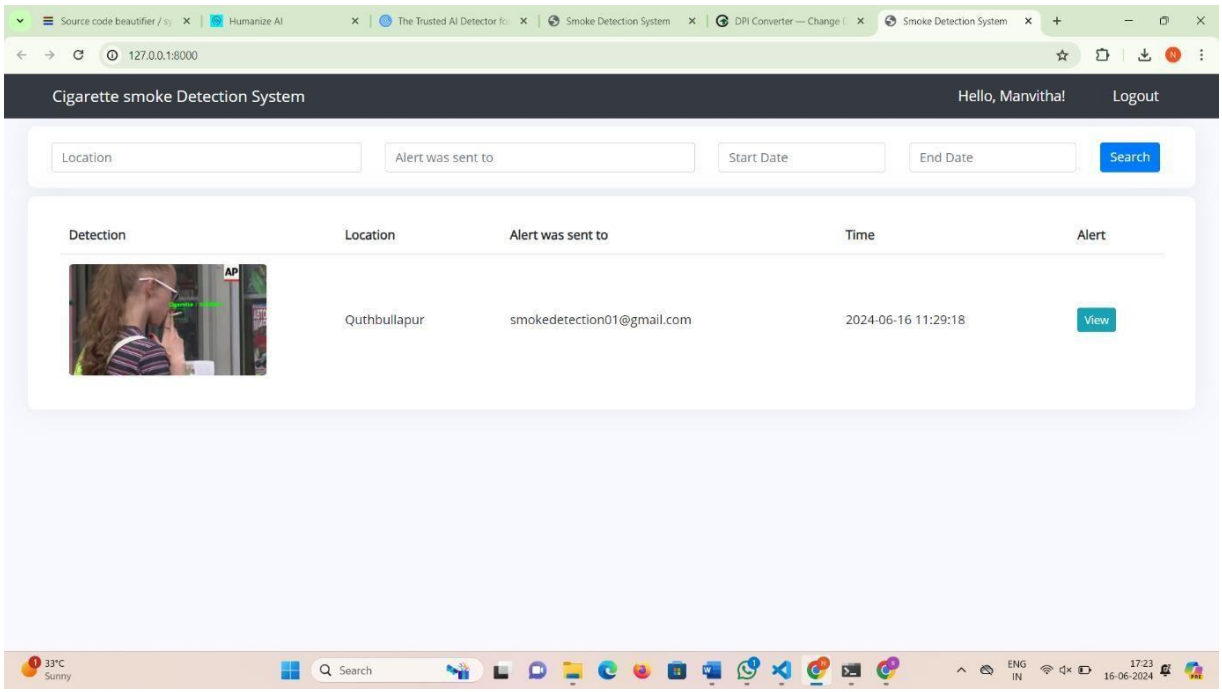


Fig 20. Alert to web app

- The proposed system makes it possible for the users to monitor smoking activities in a routine way through a web application. It also generates a monthly report that helps decide further actions in reducing the intensity of smoking in specified locations.

5.2.7 Signing through mobile number

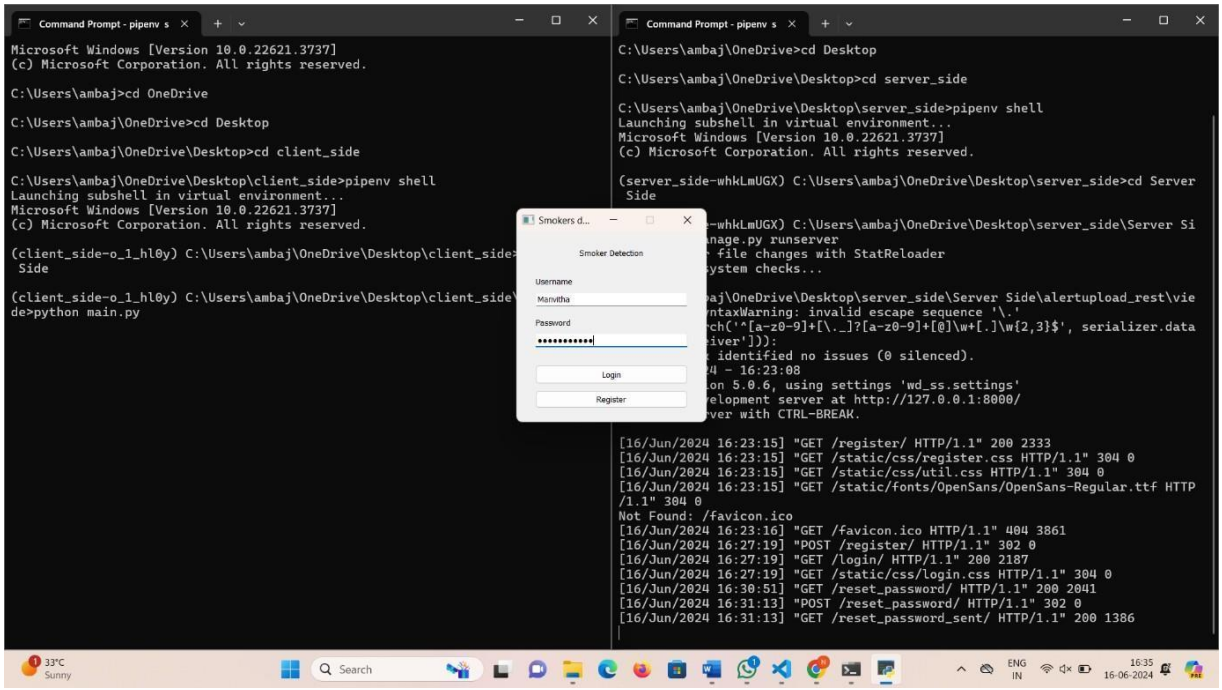


Fig 21. Login with registered account

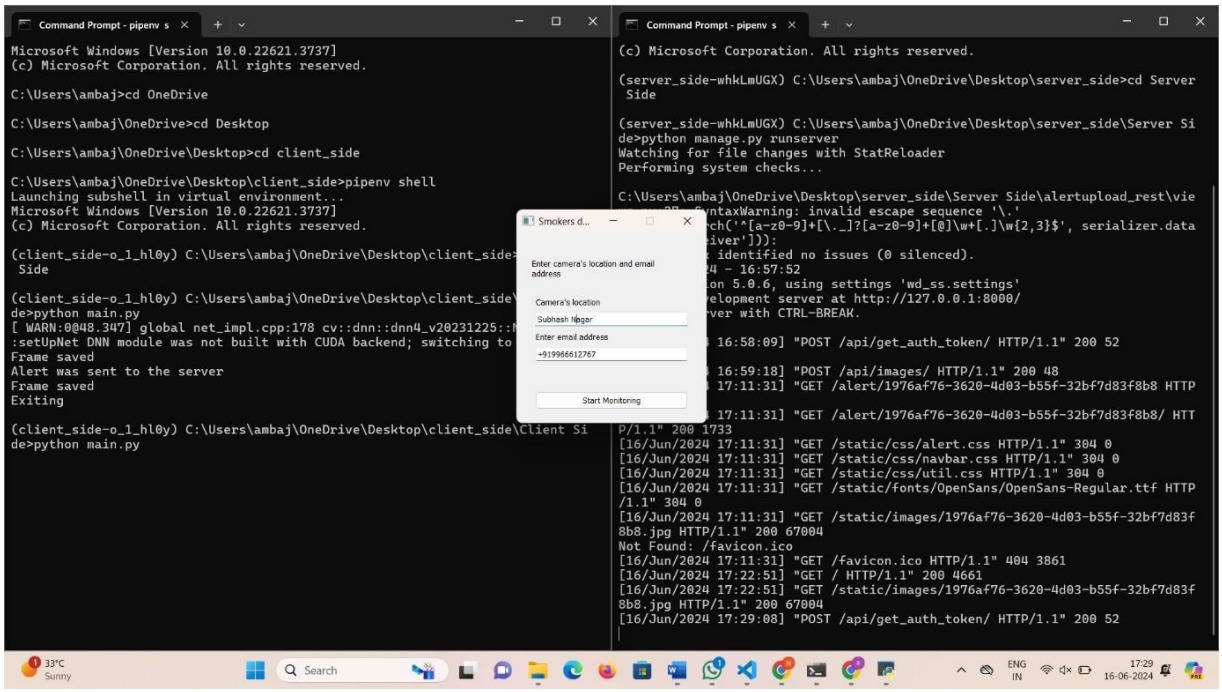


Fig 22. Enter Camera Location and mobile number

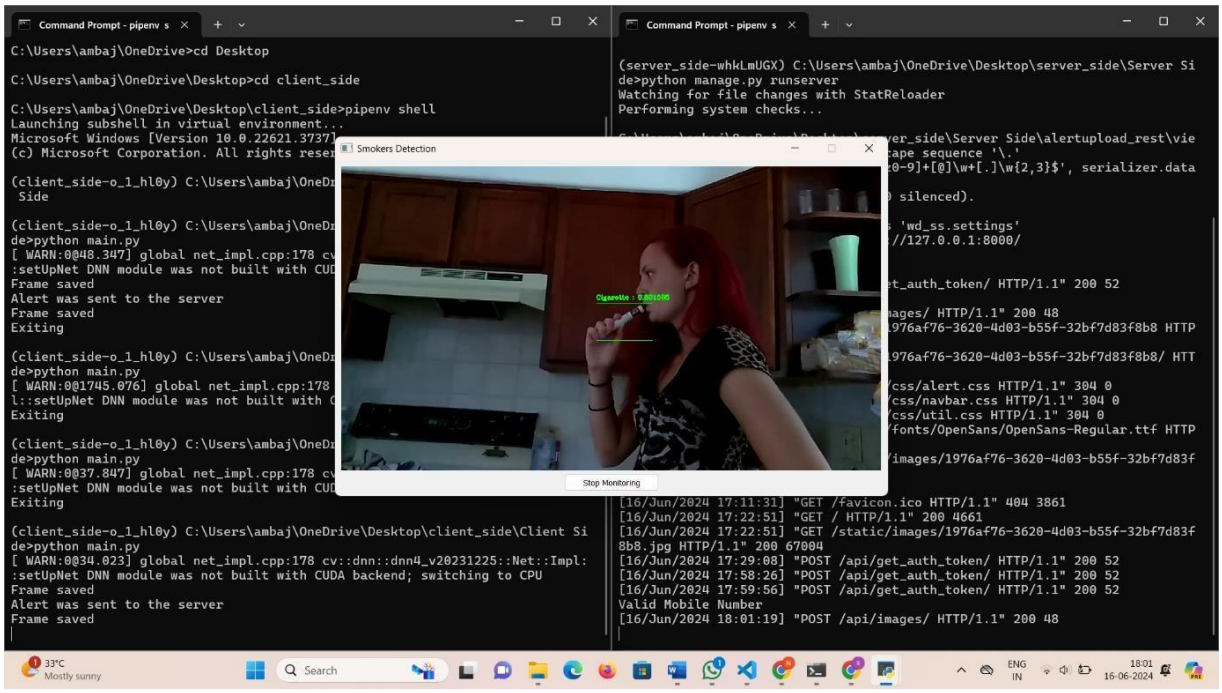


Fig 23. Smoking people are detected and frame saved to the server

5.2.8 Alert Message to Mobile Number and Web App

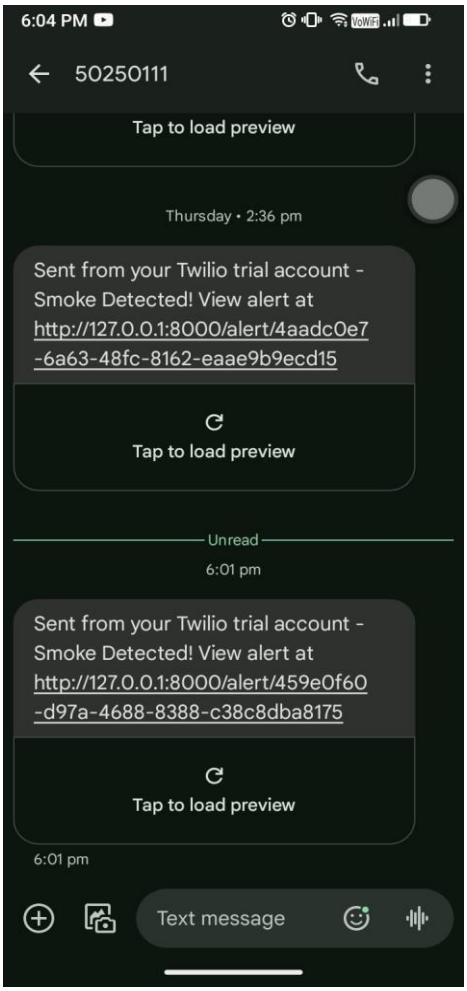


Fig 24. Alert of smoke detection is sent to mobile number by which user login

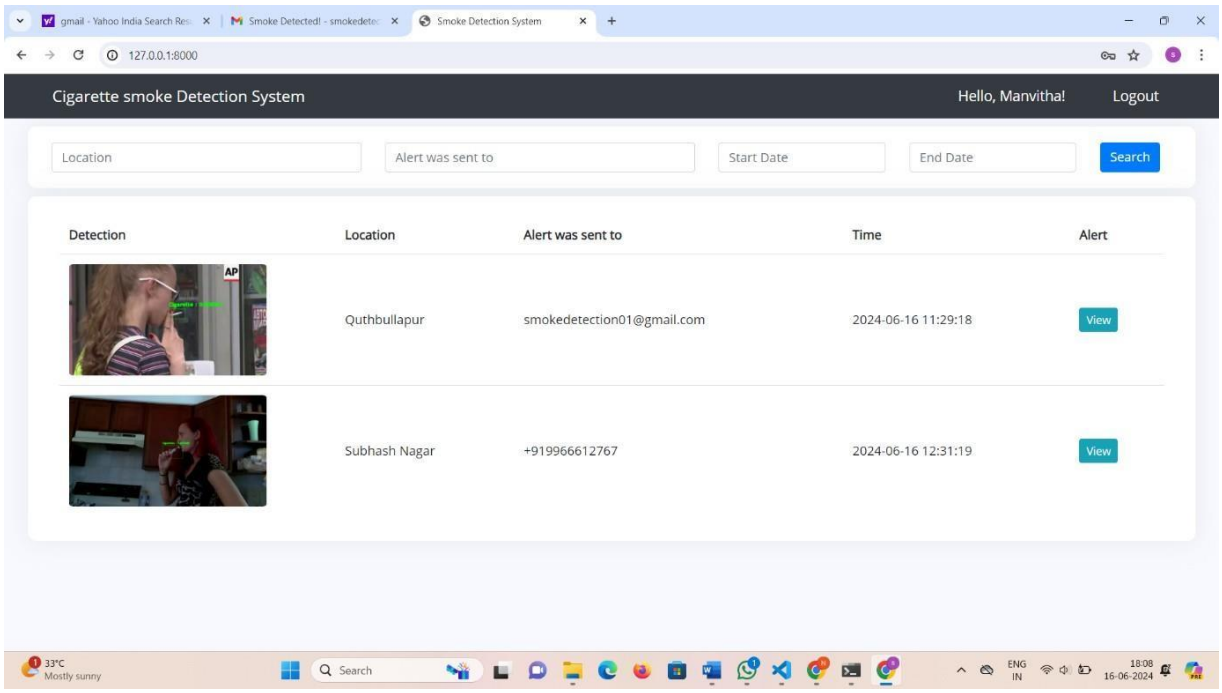


Fig 25. Alert is also updated to Web App

5.3 Test Cases

Test Case ID	Test Scenario	Test Case	Pre-condition	Test Steps	Test Data	Expected Results	Post Conditions	Actual Results	Status
1.	New User	User Authentication	-	User Registration	-	Authentication of user	-	User Authenticated	Pass
2.	New Mobile number	Mobile number Authentication	-	Mobile registration in Twilio	-	Authentication of mobile number	-	Mobile number registered	Pass
3.	Wrong email/ password	Incorrect credentials	-	Incorrect credentials during login	-	User not allowed access	-	User not allowed access	Pass
4.	Monitoring video	Detection of smoker	-	Wait till frame of smoker is saved	-	Alert sent to server	-	Alert sent to server	Pass
5.	Alert Message	Alert message on detection of smoker	-	Wait till alert sent to registered mail or mobile number	-	Alert received to registered mobile or mail	-	Alert received to registered mobile or mail	Pass
6.	Alarm	Buzzer on detection	Frame must be	Wait till the frame saved	-	Buzzer rings on detection of smoking	-	Buzzer rings on detection of smoking	Pass

			sav ed			people		people	
7.	Web App	Login to web app	-	Wait till the smoking activity being recorded	-	Smoking activity along with location, time, date is saved	-	Smoking activity along with location, time, date is saved	Pass

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

- Air pollution is manifested through its existing sources and pathways, leading to environmental degradation and affecting public health. The most detrimental method for detecting polluted air could be observed through the presence of smoking cigarettes. It not only poses health risks to their own selves but also those who surround them.
- Integration of the detection and alert mechanism in a single platform improves the smoke-free environment and mitigates associated risks, thus making the entire process of detection and alert more convenient for the respective authorities.
- This unified approach consists of Ensure Immediate Detection and Response, Provide Real-Time Alerts, Offer Ongoing Monitoring, Foster Smoke-Free Environment, Increased Accountability, and Enforcement, Ease of Use of the System, Support Public Health Initiatives.
- Development of a mobile application to report smoking incidents would hugely enhance user engagement on this platform.
- Future enhancements include improving this model with more diversified train data and its optimization for edge devices.
- The platform can further be developed for enhanced public awareness, by way of fines or penalties against the violations detected, similar to fines against traffic violation. This would further strengthen the enforcement and deter smoking in prohibited areas.

CHAPTER 7

REFERENCES

[1] CHEN Ruilong, LUO Lei, CAI Zhiping, Mawentao, “Algorithm for Real-Time Smoking Detection Based on Deep Learning”, Journal of Frontiers of Computer Science & Technology, (2021), Vol 15, Issue 2, p327, doi: 10.3778/j.issn.1673-9418.2009069

[2] Yuda Irawan, Andre Wahyu Novrianto, Hafizh Sallam, “Cigarette Smoke Detection And Cleaner Based On Internet Of Things (IoT) Using Arduino Microcontroller And MQ-2 Sensor”, Journal of Applied Engineering and Technological Science (JAETS), (2021), Vol. 2 No. 2.

[3] Jiang Chong, “An Intelligent Detection Approach for Smoking Behavior”, International Joural of Cognitive Informatics and Natural Intelligence, (2023), Vol 17 No. 1, pp 1-18.

[4] Mohammad Sala Uddin, “Cigarette CNN: A Convolutional Neural Network for Detecting Cigarette Smoking Activity”, International Conference on Engineering, Applied Sciences and System Modeling, (2024), pp 259-274.

[5] Zhen Zhang, Huiying Chen, Ruijie Xiao, Qingjie Li, Journal of Physics: Conference Series, Volume 2024, 2nd International Conference on Computer Vision and Data Mining (ICCVDM 2021) 20-22 August 2021, Changsha, China, doi: 10.1088/1742-6596/2024/1/012042

[6] Chenbing Bai, You Zhou, Qingkai Guo, “Application Study of Area-Based and YOLOv4 Smoking Behavior Detection”, 2022 4th International Conference on Natural Language Processing (ICNLP), doi: 10.1109/ICNLP55136.2022.00010

[7] Tanya Singh, Akash Singh, Deepak Sahu, Abhishek Tomar, Shivam Pandey, “Real Time Cigarette Detection using Deep Learning Models”, Ijraset Journal For Research I Applied Science and Engineering Technology, 2321-9653, (2023), IJRASET51977.

[8] Tapas Guha, “Detection of Smoking in iside the office using Deep Learning”, 2022 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES), doi: 10.1109/ICSES55317.2022.9914319

[9] Somantri, Rizki Fauzi Ridwanullah, Hendra, DEVI Safitri, “Cigarette Smoke Detection System for Non-Smoking Areas Based on IoT and Face Recognition”, 2020 6th International Conference on Computing Engineering and Design (ICCED), doi: 10.1109/ICCED51276.2020.9415798.

[10] Zhao Rentao, Wang Mengyi, Zhai Zilong, Li Ping, Zeng Qingyu, “Indoor Smoking

Behavior Detection Based on YOLOv3-tiny", 2019 Chinese Automation Congress(CAC), doi: 10.1109/CAC48633.2019.8996951

[11] Jianhao Liao, Jiayu Zou, Ao Shen, Jinfu Liu, Xiaofei Du, "Cigarette end detection based on EfficientDet", Journal of Physics: Conference Series, Volume 1748, The 2020 5th International Seminar on Computer Technology, Mechanical and Electrical Engineering (ISCME 2020) 30 October-1 November 2020, Shenyang, China, doi: 10.1088/1742-6596/1748/6/062015

[12] Yankai Ma, Jun Yang, Zhendong Li, Ziqiang Ma, "YOLO-Cigarette: An effective YOLO Network for outdoor smoking Real-time Object Detection", 2021 Ninth International Conference on Advanced Cloud and Big Data (CBD), doi: 10.1109/CBD54617.2021.00029

[13] David Habsara Hareva, Samuel Teguh, "Capturing Smoker in Non-Smoking Room", ICAAI '19: Proceedings of the 3rd International Conference on Advances in Artificial Intelligence.

[14] Jae Hyuk Cho, "Detection of Smoking in Indoor Environment Using Machine Learning", The Development and Application of Fuzzy Logic,(2020).

[15] Lisu Han, Leilei Rong, Yongquan Li, Zhikang Qin, Yan Xu, "CA-SSD-Based Real-time Smoking Target Detection Algorithm", ICDSP '21: Proceedings of the 2021 5th International Conference on Digital Signal Processing, doi: https://doi.org/10.1145/3458380.3458429

APPENDIX

Snapshots of the Result

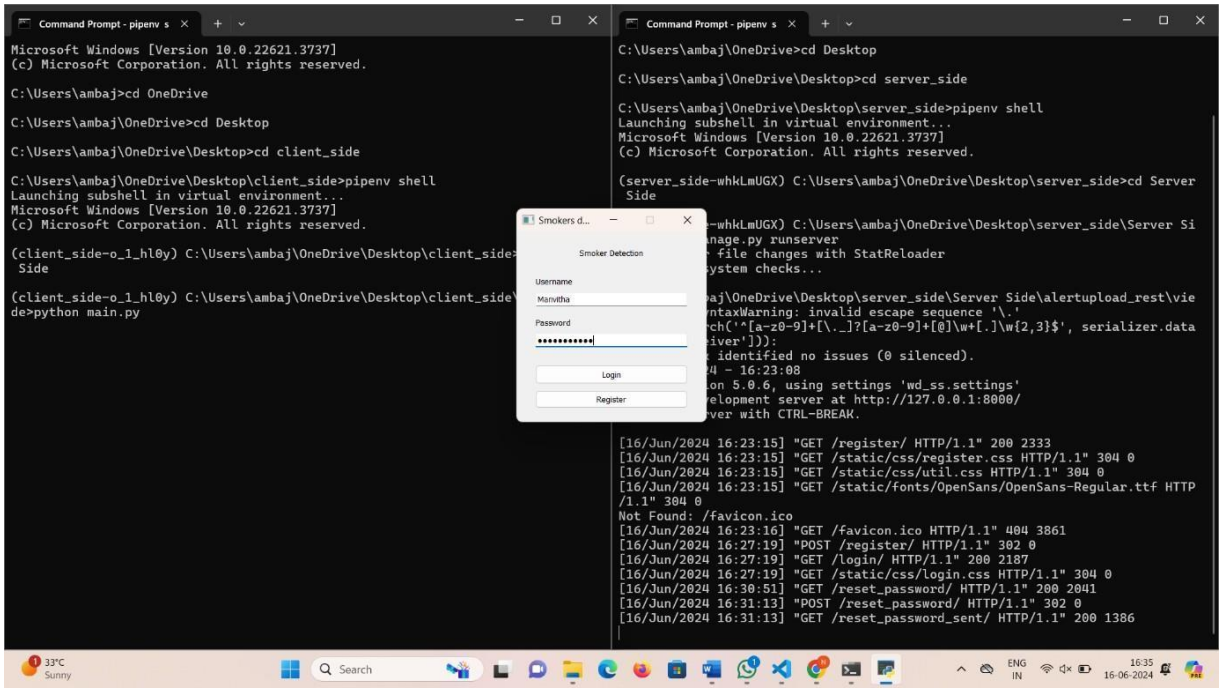


Fig 26. Login

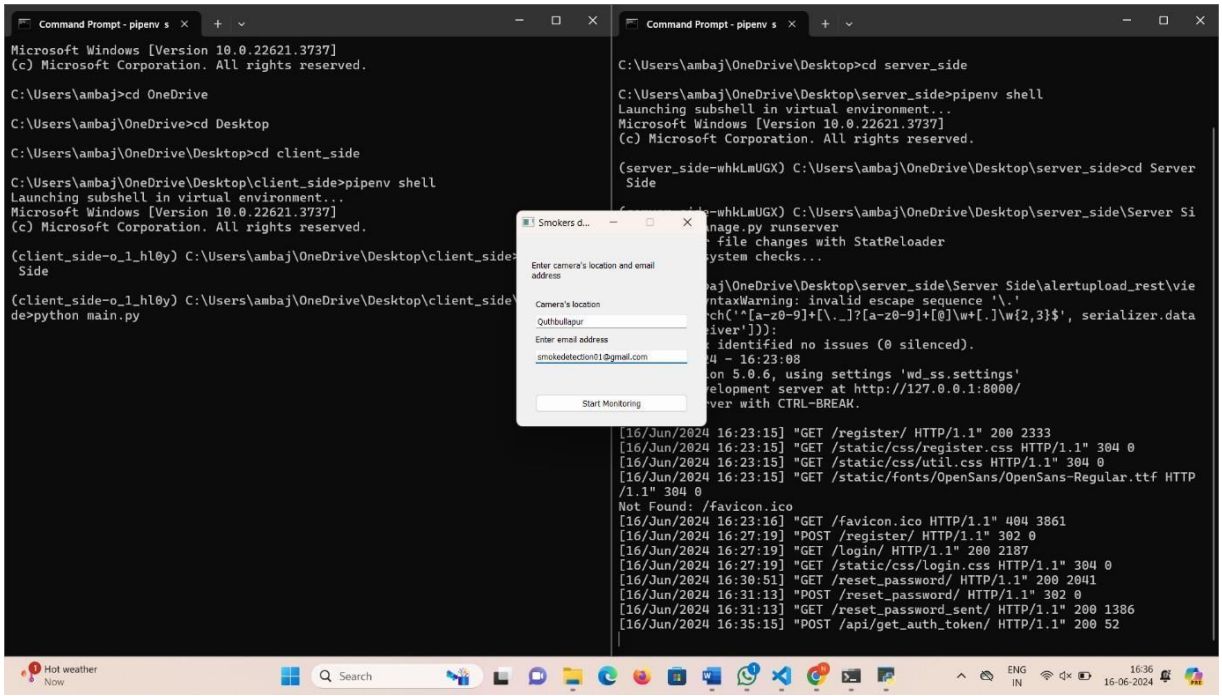


Fig 27. Enter camera location and mail address

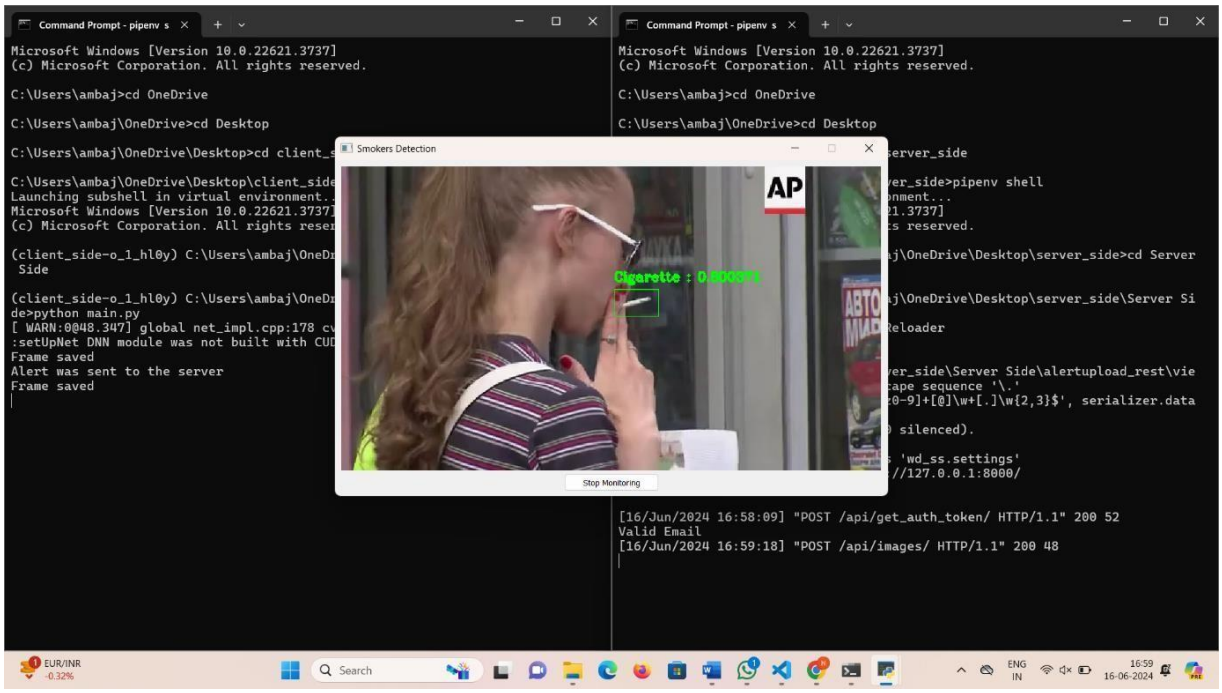


Fig 28. Smoke detection and saving of frame

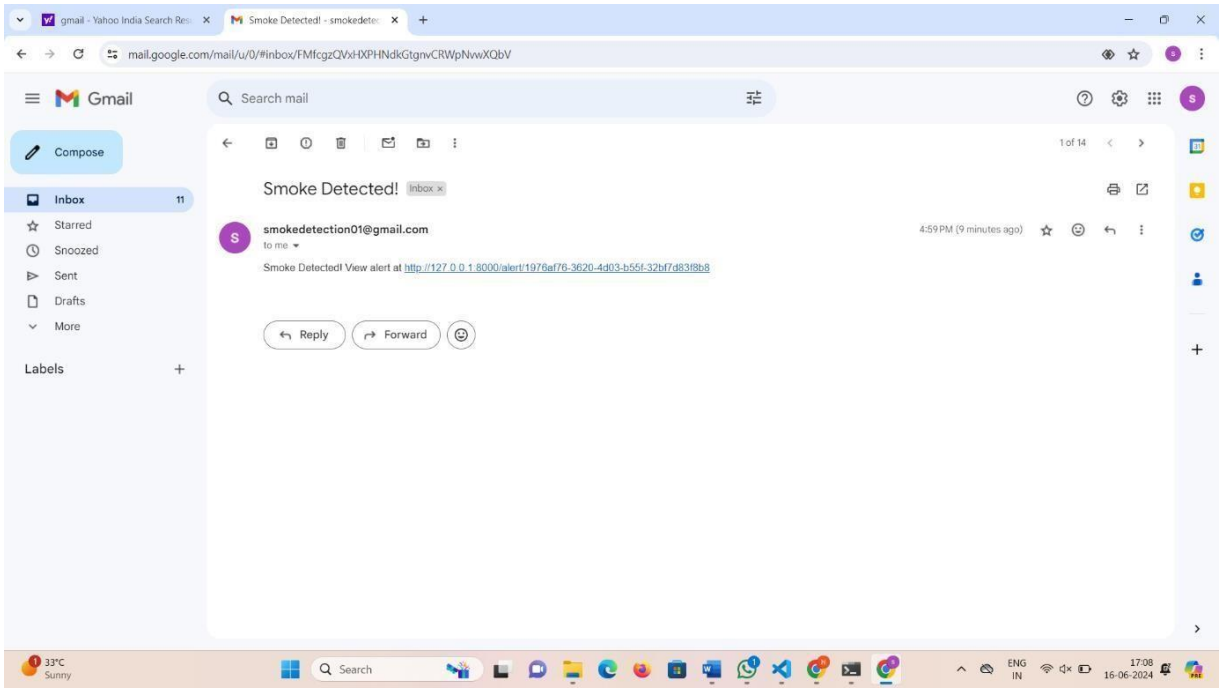


Fig 29. Alert Message to mail

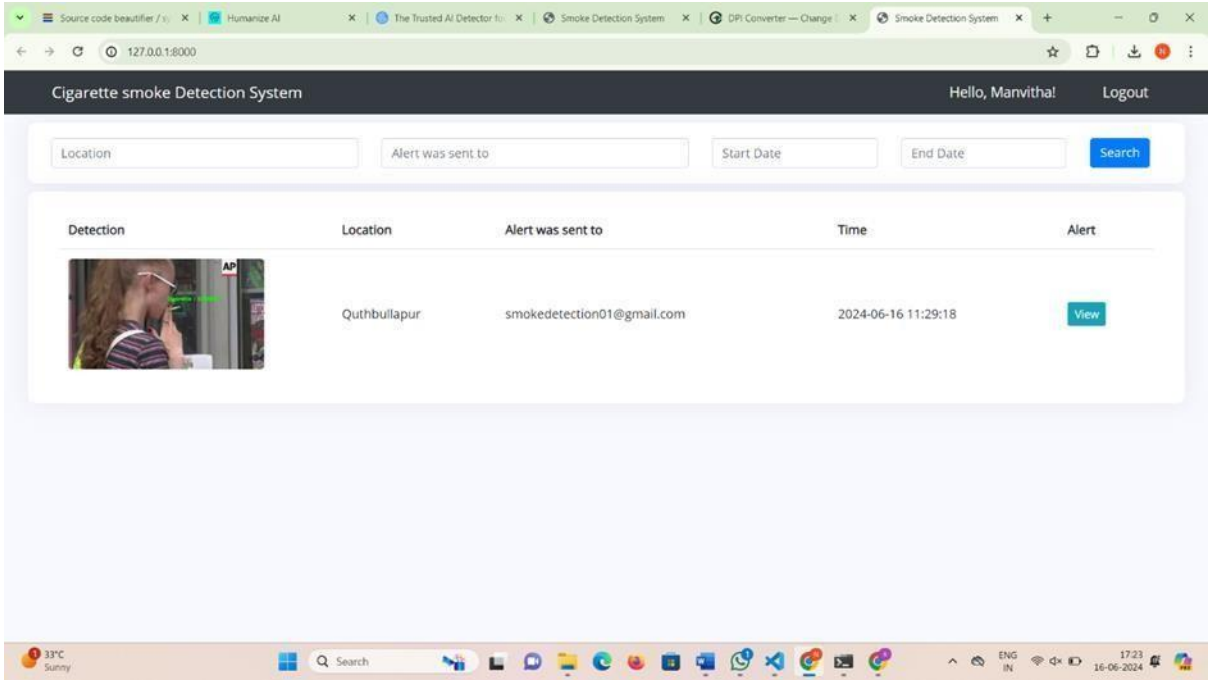


Fig 30. Web App alert

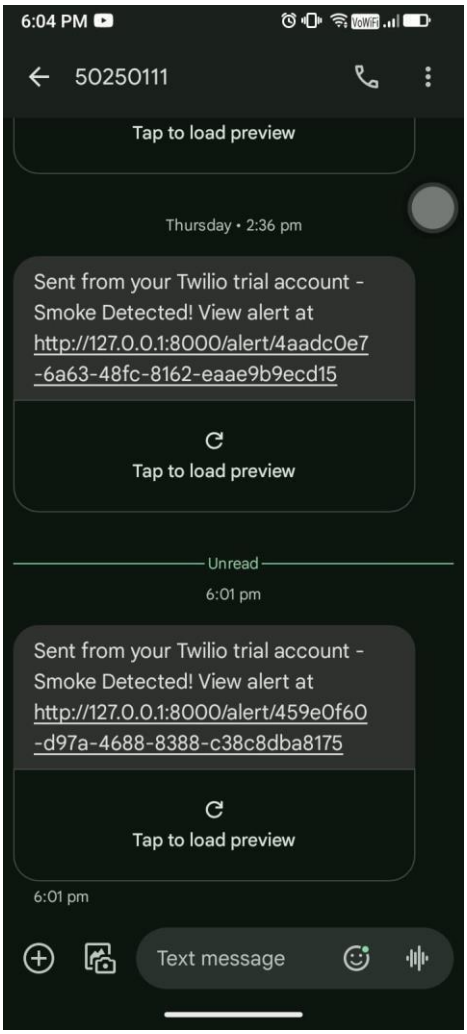


Fig 31. Mobile Alert

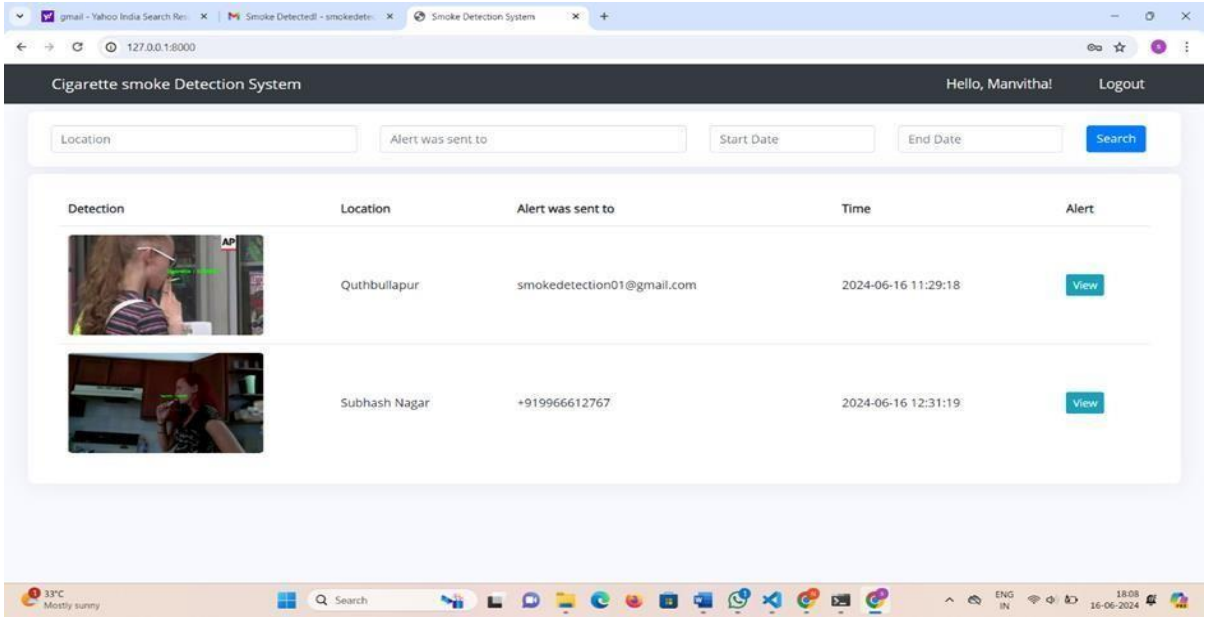


Fig 32. Web App alert

PAPER NAME	AUTHOR
A1.pdf	A1 Griet

WORD COUNT	CHARACTER COUNT
6118 Words	33526 Characters
PAGE COUNT	**FILE SIZE**
57 Pages	16.1MB
SUBMISSION DATE	**REPORT DATE**
Jun 26, 2024 1:37 PM GMT+5:30	Jun 26, 2024 1:37 PM GMT+5:30

2% Overall Similarity

The combined total of all matches, including overlapping sources, for each database.

- **2% Internet database**
- **0% Publications database**
- **Crossref Posted Content database**
- **2% Submitted Works database**

Excluded from Similarity Report

- **Crossref database**
- **Bibliographic material**
- **Small Matches (Less then 8 words)**

2% Overall Similarity

Top sources found in the following databases:

- **2% Internet database**
- **0% Publications database**
- **Crossref Posted Content database**
- **2% Submitted Works database**

TOP SOURCES

The sources with the highest number of matches within the submission. Overlapping sources will not be displayed.

1	Metropolitan State University on 2012-12-13 Submitted works	<1%
2	cdn.techscience.cn Internet	<1%
3	pypi.org Internet	<1%
4	Macao Polytechnic Institute on 2024-04-10 Submitted works	<1%
5	British University in Egypt on 2024-06-10 Submitted works	<1%
6	arxiv-vanity.com Internet	<1%
7	The Hong Kong Polytechnic University on 2006-04-09 Submitted works	<1%
8	Universidad de Jaén on 2020-08-18 Submitted works	<1%
9	University of Central Florida on 2023-06-07 Submitted works	<1%

10	coursehero.com Internet	<1%
11	ijraset.com Internet	<1%