How To Develop Frontend for PPE Violation Detection

Step 1 - Creating a Workspace Directory

First we create a directory using the following command where we keep all the files related to the user interface of our PPE Violation Detection application.

```
mkdir PPE-Violation-Detection
```

```
cd PPE-Violation-Detection/
```

Step 2 - Creating a Virtual Environment

Now we create a virtual environment using the following command in our workspace directory.

```
python3 -m venv venv/
```

```
source venv/bin/activate
```

Step 3 - Installing Requirements

Now, its time to install the packages that will be required for our frontend development.

```
pip3 install -r requirements.txt
```

We will also be installing some additional requiremnets for opencv-python required by Ubuntu platform.

```
sudo apt update
```

```
sudo apt install ffmpeg libsm6 libxext6 -y
```

Step 4 - Creating HTML File

Let's head towards our HTML skeleton for the user interface of our PPE Violation Detection application.

Since it is a flask application, we will be needing a *template/* directory to render our HTML file. So first, we create *template/* directory.

```
mkdir templates
```

```
cd templates/
```

Now, create an `index.html` file using the following command:

```
nano index.html
```

Now copy paste the following script in your *nano* editor.

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

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <!--=============== REMIXICONS ===============-->
    <link href="https://cdn.jsdelivr.net/npm/remixicon@2.5.0/fonts/remixicon.css"
rel="stylesheet">

    <!--=============== CSS ===============-->
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename="css/styles.css") }}">

    <title>PPE Violation Detection</title>
</head>

<body>
    <!--=============== AJAX ===============-->
    <script
src="https://cdnjs.cloudflare.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>

    <script>
        $(document).ready(function () {
            $('#myform').submit(function (event) {
                event.preventDefault()
                //submit_form(event);
            });
        });

        $(document).ready(function () {
            $('#alert_email_checkbox').change(function (event) {
```

```
                event.preventDefault()
                data = {
                    'alert_email_checkbox':
$('#alert_email_checkbox').is(':checked'),
                    'alert_email_textbox': $('#alert_email_textbox').val(),
                }
                $.ajax({
                    type: 'POST',
                    url: '/submit',
                    data: data,
                    success: function (data) {
                        alert(data);
                    },
                    error: function (error) {
                        alert('Checkbox submission failed!');
                    }
                });
            });
        });


        function upload_file() {
            $.ajax({
                type: 'POST',
                url: '/submit',
                data: new FormData($('#myform')[0]),//formData,
                processData: false,
                contentType: false,
                cache: false,          //Required
                success: function (data) {
                    alert(data);
                },
                error: function (error) {
                    alert('Form submission failed!');
                }
            });
        }

        function download_file() {
            data = {
                'download_button': 'True',
            }
            $.ajax({
                type: 'POST',
                url: '/submit',
                data: data,
                xhrFields: {
                    responseType: 'blob' // to avoid binary data being mangled on
charset conversion
                },
                // to download file as an attachment
                //Reference - https://stackoverflow.com/questions/16086162/handle-
file-download-from-ajax-post
                success: function (blob, status, xhr) {
```

```
                    // check for a filename
                    var filename = "";
                    var disposition = xhr.getResponseHeader('Content-
Disposition');
                    if (disposition && disposition.indexOf('attachment') !== -1) {
                        var filenameRegex = /filename[^;=\n]*=((['"]).*?\2|
[^;\n]*)/;
                        var matches = filenameRegex.exec(disposition);
                        if (matches != null && matches[1]) filename =
matches[1].replace(/['"]/g, '');
                    }

                    if (typeof window.navigator.msSaveBlob !== 'undefined') {
                        // IE workaround for "HTML7007: One or more blob URLs were
revoked by closing the blob for which they were created. These URLs will no longer
resolve as the data backing the URL has been freed."
                        window.navigator.msSaveBlob(blob, filename);
                    } else {
                        var URL = window.URL || window.webkitURL;
                        var downloadUrl = URL.createObjectURL(blob);

                        if (filename) {
                            // use HTML5 a[download] attribute to specify filename
                            var a = document.createElement("a");
                            // safari doesn't support this yet
                            if (typeof a.download === 'undefined') {
                                window.location.href = downloadUrl;
                            } else {
                                a.href = downloadUrl;
                                a.download = filename;
                                document.body.appendChild(a);
                                a.click();
                            }
                        } else {
                            window.location.href = downloadUrl;
                        }
                        setTimeout(function () { URL.revokeObjectURL(downloadUrl);
}, 100); // cleanup
                    }
                },
                error: function (error) {
                    alert('Form submission failed!');
                }
            });
        }

        function video_inference() {
            data = {
                'inference_video_button': 'true',
            }
            $.ajax({
                type: 'POST',
                url: '/submit',
                data: data,
```

```
                success: function (data) {
                    //alert(data);
                    //window.location.href = '/';
                },
                error: function (error) {
                    alert('Video inference failed!');
                }
            });
        }

        function live_inference() {
            data = {
                'live_inference_button': 'true',
                'live_inference_textbox': $('#ip_address_textbox').val(),
            }
            $.ajax({
                type: 'POST',
                url: '/submit',
                data: data,
                success: function (data) {
                    //alert(data);
                    //window.location.href = '/';
                },
                error: function (xhr, status, error) {
                    alert(xhr.responseText);
                }
            });
        }
    </script>

    <!-- =============== HEADER =============== -->
    <header class="header" id="header">
        <div class="title">
            <h1>PPE Violation Detection</h1>
        </div>
    </header>

    <!-- =============== VIDEOS =============== -->

    <div class="gallery_container">

        <div class="gallery">
            <img src="{{ url_for('video_raw') }}" />
        </div>

        <div class="gallery">
            <img src="{{ url_for('video_processed') }}" />
        </div>
    </div>


    <!-- =============== OPERATIONS =============== -->
    <div class="operations_wrapper">
        <form id="myform" enctype="multipart/form-data" method="post">
```

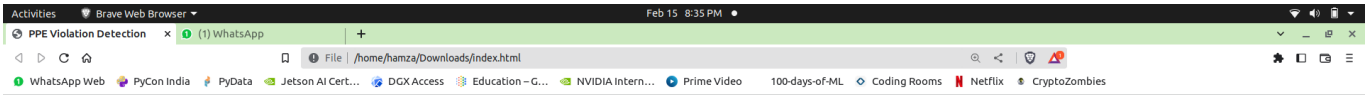
```
            <div class="btn">
                <div class="upload__button">
                    <input type="file" class="custom-file-input" name="video"
id="video" value="video">
                    <button type="submit" class="btn-primary"
name="video_upload_button" id="video_upload_button"
                        onclick="upload_file()">Upload Video
                        <i class="ri-video-upload-fill button__icon"></i>
                    </button>
                </div>
                <div class="live__button">
                    <button class="btn-primary" name="inference_video_button"
id="inference_video_button"
                        onclick="video_inference()">Inference on Video
                        <i class="ri-movie-2-fill button__icon"></i>
                    </button>
                </div>
                <div class="inference__button">
                    <input type="text" class="ip_address-input"
name="ip_address_textbox" id="ip_address_textbox"
                        placeholder="http://192.168.12.10:4747/video"
value="http://192.168.12.10:4747/video">
                    <button type="submit" class="btn-primary"
name="live_inference_button" id="live_inference_button"
                        onclick="live_inference()">Live Inference
                        <i class="ri-settings-4-fill button__icon"></i>
                    </button>
                </div>
                <div class="download__button">
                    <button type="submit" class="btn-primary"
name="download_button" id="download_button"
                        onclick="download_file()">Download Report
                        <i class="ri-file-download-fill button__icon"></i>
                    </button>
                </div>
                <div class="email__sending">
                    <div class="send__email">
                        <input type="email" placeholder="Enter Valid Mail"
value="support.ai@giindia.com"
                            name="alert_email_textbox" id="alert_email_textbox">
                    </div>
                    <div class="toggle__content">
                        <label class="toggle__label">
                            <input type="checkbox" class="toggle__check"
name="alert_email_checkbox"
                                id='alert_email_checkbox' />
                            <span class="email__label">Send Alert</span>
                            <div class="toggle__rail">
                                <span class="toggle__circle"></span>
                                <span class="toggle__border"></span>
                            </div>
                        </label>
                    </div>
                </div>
```

```
            </div>
        </form>
    </div>
</body>

</html>
```

Save the file by pressing `Ctrl+X` → `Y` → `Enter`

If you open `index.html` in a browser, it should be looking someting like the following screenshot.



Move back to parent directory

```
cd ..
```

Step 3 - Styling Our User Interface

Does the user interface looks satisfactory? No, lets add some styling to our user interface.

In flask application, we cannot directly import our CSS file in `index.html`. There something called static files in flask application where all the CSS, JavaScript and other types of scripts are kept. So, lets create our `styles.css` as described below:

1. First, create *static/* directory using the following command:

```
mkdir static
```

```
cd static/
```

2. Now, specify the type folders that we will be needing for storing different files for our flask application.

```
# To store our CSS file

mkdir css
```

```
# To store the uploaded video

mkdir video
```

```
# To save the model rports

mkdir reports
```

```
# To save the model violations

mkdir vilations
```

3. Finally, creating and editing the `styles.css` file.

```
cd css/
```

```
nano styles.css
```

Now copy paste the following script into your *nano* editor.

```
/*=============== GOOGLE FONTS ===============*/
@import url('https://fonts.googleapis.com/css2?
family=Roboto+Mono:wght@500&display=swap');

/*=============== VARIABLES CSS ===============*/
:root {
  /*========== Colors ==========*/
  --light: #F6FAFD;
  --dark: #122272;
```

```
  --pri-blue: #193FAF;
  --sec-blue: #17A5F8;
  --pri-green: #23C99D;
  --alert: #FE7F0E;

  /*========== Font and typography ==========*/
  --body-font: 'Poppins', sans-serif;
  --h1-font-size: 1.5rem;
  --medium-font-size: 0.973rem;
  --small-font-size: 0.813rem;
  --smaller-font-size: 0.75rem;
}

/*Responsive typography*/
@media screen and (min-width: 1024px) {
  :root {
    --h1-font-size: 1.6875rem;
    --medium-font-size: 1.125rem;
    --small-font-size: .875rem;
    --smaller-font-size: .813rem;
  }
}

/*=============== BASE ===============*/

* {
  box-sizing: border-box;
  padding: 0;
  margin: 0;
}

body {
  font-size: 1em;
  font-weight: 500;
  font-family: var(--body-font);
  background-color: var(--light);
}

img,
video {
  max-width: 100%;
  height: auto;
}

/*form :where(i, p) {
  /* color: var(--pri-blue);
} */

form i {
  font-size: 1em;
}

form button {
  font-size: 16px;
```

```
  border: none;
  font: var(--body-font);
  background-color: var(--first-color);
  cursor: pointer;
  color: #F6FAFD;
}

a {
  text-decoration: none;
  color: var(--sec-blue);
}

.main {
  padding: 0.5rem;
}

/*=============== GALLERY ===============*/

.gallery_container {
  /*width: 100%;*/
  margin-left: 10%;
  /*position: absolute;*/
  display: flex;
  width: 80%;
}

.gallery {
  flex: 1;
  border-radius: 1em;
  outline: 3px dashed var(--pri-blue);
  margin: 2em;
}

.gallery:first-child {
  margin-right: 3em;
}

.gallery img {
  width: 100%;
  height: 100%;
  border-radius: 1em;
}

/*=============== OPERATIONS ===============*/
.operations_wrapper {
  width: 90%;
  margin: 5% 5% 0% 5%;
  background-color: #fff;
  border-radius: 1em;
}

/*=============== HEADING ===============*/
.title {
  text-align: center;
```

```
}

.header {
  font-size: 2em;
  font-weight: 600;
  text-align: center;
  height: 5em;
  padding-top: 1em;
  color: #F6FAFD;
  margin-bottom: 5rem;
  box-shadow: 0 4px 8px 0 rgba(0, 0, 0, 0.2), 0 6px 20px 0 rgba(0, 0, 0, 0.1);
  border-bottom: 1rem;
  font-family: 'Roboto Mono', monospace;
  background: var(--pri-blue);
  background: linear-gradient(110deg, var(--dark) 38%, var(--pri-blue) 100%);
}

/*=============== BUTTONS ===============*/
.btn {

  padding: 2em;
  display: flex;
  justify-content: space-evenly;
  align-items: center;

}

.upload__button,
.download__button,
.live__button,
.inference__button {
  display: inline-flex;
  align-items: center;
  background-color: var(--pri-blue);
  color: #fff;
  border-radius: 0.5rem;
  padding: 0.5rem 1.5rem;
  cursor: pointer;
}

.upload__button:hover,
.download__button:hover,
.live__button:hover,
.inference__button:hover,
.email__sending:hover {
  background-color: var(--pri-blue);
  background: var(--pri-green);
  background: linear-gradient(144deg, var(--pri-green) 20%, var(--sec-blue) 100%);
  color: var(--light);
}

.upload__button:hover button,
.download__button:hover button,
.live__button:hover button,
```

```
.inference__button:hover button {
  color: var(--light);
}

.button__icon {
  margin-left: 0.25rem;
  transition: 0.3s;
  color: var(--light);
  font-size: var(--h1-font-size);
}

.download__button:hover .button__icon {
  transform: translateY(0.25rem);
}

.upload__button:hover .button__icon {
  transform: translateY(-0.25rem);
}

.inference__button:hover .button__icon {
  transform: rotate(1rad);
}

.ip_address-input,
.custom-file-input {
  margin-right: 1rem;
}

/*=============== TOGGLE SWITCH ===============*/
.email__sending {
  border-radius: 0.5rem;
  padding: 1rem 5rem 1rem 1rem;
  background-color: var(--pri-blue);
  display: inline-flex;
  align-items: center;
  /* margin: 2rem 5rem; */
}

.toggle__content {
  position: relative;
  margin-left: 2rem;
  bottom: 0.74rem;
}

.email__label {
  position: relative;
  left: 4rem;
  top: 0.85rem;
}

.toggle__label {
  cursor: pointer;
  padding-block: 0.5rem;
}
```

```
.toggle__check {
  display: none;
}

.toggle__rail {
  position: relative;
  width: 52px;
  height: 4px;
  background-color: var(--light);
  border-radius: 2rem;
}

.toggle__circle {
  display: block;
  width: 24px;
  height: 24px;
  background-color: var(--alert);
  /* box-shadow: inset 0 0 0 4px var(--dark); */
  border-radius: 50%;
  position: absolute;
  left: 0;
  top: 0;
  bottom: 0;
  margin: auto 0;
  transition: transform 0.4s, box-shadow 0.4s;
  z-index: 2;
}

.toggle__border {
  position: absolute;
  width: 32px;
  height: 32px;
  background-color: var(--light);
  border-radius: 50%;
  left: -4px;
  top: 0;
  bottom: 0;
  margin: auto 0;
  transition: transform 0.4s;
}

/*Toggle animation effects*/
.toggle__check:checked~.toggle__rail .toggle__circle {
  transform: translateX(28px);
  box-shadow: inset 0 0 0 12px var(--pri-green);
}

.toggle__check:checked~.toggle__rail .toggle__border {
  transform: translateX(28px);
}

/*=============== BREAKPOINTS ===============*/
/*For small devices*/
```

```
/*For large devices*/
```

Save the file by pressing `Ctrl+X` → `Y` → `Enter`

Move back to parent directory

```
cd ..
```

Again

```
cd ..
```

Step 4 - Creating Email Sending Script

Time to create a automated email sending script in *PPE-Violation-Detection/* directory.

```
nano send_mail.py
```

Copy and paste the following python script into your *nano* editor.

```
"""
This module sends emails with attachments to the participants
Reference - https://developers.google.com/gmail/api/quickstart/python

In order to run this module, you need to enable Gmail API and download
client_secrets.json file
"""

from email import encoders
from email.mime.base import MIMEBase
from email.mime.image import MIMEImage
from email.mime.multipart import MIMEMultipart
import mimetypes
import os
import time

from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
from email.mime.text import MIMEText
import base64
import cv2
```

```
# If modifying these scopes, delete the file token.json.
# We are using Gmail API to send emails
SCOPES = ['https://www.googleapis.com/auth/gmail.send']


def authentication():
    creds = None

    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
time.
    if os.path.exists('token.json'):
        # Load the credentials from the file
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)

    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        # Refresh the token if it has expired
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            # If there are no valid credentials available, let the user log in.
            flow = InstalledAppFlow.from_client_secrets_file(
                'client_secrets.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())
    return creds


def prepare_and_send_email(sender, recipient, subject, message_text, im0: bytes):
    """Prepares and send email with attachment to the participants

    Args:
        sender: Email address of the sender.
        recipient: Email address of the receiver.
        subject: The subject of the email message.
        message_text: The text of the email message.
        im0: The image to be attached

    Returns:
        None
    """
    # Get credentials
    creds = authentication()

    try:
        # Call the Gmail API
        service = build('gmail', 'v1', credentials=creds)

        # create message using a custom, function create_message()
        msg = create_message(sender, recipient, subject, message_text, im0)
```

```
        # send the message using a custom function send_message()
        send_message(service, 'me', msg)  # here 'me' is the user_id of the
authenticated user

    except HttpError as error:
        # TODO(developer) - Handle errors from gmail API.
        print(f'An error occurred: {error}')


def create_message(sender, to, subject, message_text, img_file):
    """Create a message for an email.

    Args:
        sender: Email address of the sender.
        to: Email address of the receiver.
        subject: The subject of the email message.
        message_text: The text of the email message.
        img_file: The image to be attached

    Returns:
        An object containing a base64url encoded email object.
    """
    # create a multipart email message with attachment
    message = MIMEMultipart()

    message['from'] = sender
    message['to'] = to
    message['subject'] = subject

    # create a directory to store the images that are attached to the email
    base_loc = '.\\static/violations\\'
    location = 'ABESIT'

    # get current date and time
    current_date_time = time.time()
    formatted_date_time = time.strftime("%H-%M-%S_%d-%m-%Y",
time.localtime(current_date_time))

    # if base_loc doesn't exist, create it
    if not os.path.exists(base_loc):
        os.makedirs(base_loc)

    file_name = base_loc + 'violation_' + str(location) + '_' +
str(formatted_date_time) + '.jpg'

    # convert img_file into jpeg format and save it in the file_name
    cv2.imencode('.jpg', img_file)[1].tofile(file_name)

    msg = MIMEText(message_text)
    message.attach(msg)

    content_type, encoding = mimetypes.guess_type(file_name)
    main_type, sub_type = content_type.split('/', 1)
```

```
    print(f'Attachment main_type = {main_type}, subtype= {sub_type}, and encoding
= {encoding}')

    # code to attach text, image, pdf and other files
    # if attachment is a text file
    if main_type == 'text':
        with open(file_name, 'r') as fp:
            msg = MIMEText(fp.read(), _subtype=sub_type)
        fp.close()
    # if attachment is an image file
    elif main_type == 'image':
        fp = open(file_name, 'rb')
        msg = MIMEImage(fp.read(), _subtype=sub_type)
        fp.close()
    # if attachment is a pdf file, then we need to set the main_type to
application and sub_type to octet-stream
    elif main_type == 'application' and sub_type == 'pdf' and encoding is None:  #
Reference - https://coderzcolumn.com/tutorials/python/mimetypes-guide-to-
determine-mime-type-of-file
        print("INSIDE PDF")
        main_type = 'application'
        sub_type = 'octet-stream'
        fp = open(file_name, 'rb')
        msg = MIMEBase(main_type, sub_type)
        msg.set_payload(fp.read())
        encoders.encode_base64(msg)
        fp.close()
    # if attachment is anything else
    else:
        fp = open(file_name, 'rb')
        msg = MIMEBase(main_type, sub_type)
        msg.set_payload(fp.read())
        fp.close()

    filename = os.path.basename(file_name)
    # add attachment to the message header
    msg.add_header('Content-Disposition', 'attachment', filename=filename)
    message.attach(msg)

    # convert the message into a string
    return {'raw':
base64.urlsafe_b64encode(message.as_string().encode()).decode()}


def send_message(service, user_id, message):
    """Send an email message.

    Args:
        service: Authorized Gmail API service instance.
        user_id: User's email address. The special value "me"
        can be used to indicate the authenticated user.
        message: Message to be sent.

    Returns:
```

```
        Sent Message.
    """
    try:
        message = (service.users().messages().send(userId=user_id, body=message)
                   .execute())
        print('Message Id: %s' % message['id'])
        return message
    except HttpError as error:
        print('An error occurred: %s' % error)


if __name__ == '__main__':
    # Uncomment the following lines to run the code locally
    # set sender and recipient accordingly
    # sender must be a gmail account using which you have enabled the gmail API
    '''prepare_and_send_email(sender='support.ai@giindia.com',
                              recipient='anubhavpatrick@gmail.com',
                              subject= 'Greeting from Global Infoventures',
                              message_text= 'Hello, this is a test email from Global
Infoventures',
                              im0= cv2.imread('test.jpg'))'''
    pass
```

Step 5 - Creating the Flask application

Now we create our flask application in *PPE-Violation-Detection/* directory.

```
nano app.py
```

Copy and paste the following python script into your *nano* editor.

```
import os.path
import cv2
import validators
from flask import Flask, render_template, request, Response
from send_mail import prepare_and_send_email

# Initialize the Flask application
app = Flask(__name__)
app.config["VIDEO_UPLOADS"] = "static/video"
app.config["ALLOWED_VIDEO_EXTENSIONS"] = ["MP4", "MOV", "AVI", "WMV", "WEBM"]

# Secret key for the session
app.config['SECRET_KEY'] = 'ppe_violation_detection'

# global variables
frames_buffer = []  # buffer to store frames from a stream
vid_path = app.config["VIDEO_UPLOADS"] + '/vid.mp4'  # path to uploaded/stored
video file
```

```
video_frames = cv2.VideoCapture(vid_path)  # video capture object


def allowed_video(filename):
    """
    A function to check if the uploaded file is a video

    Args:
        filename (str): name of the uploaded file

    Returns:
        bool: True if the file is a video, False otherwise
    """
    if "." not in filename:
        return False

    extension = filename.rsplit(".", 1)[1]

    if extension.upper() in app.config["ALLOWED_VIDEO_EXTENSIONS"]:
        return True
    else:
        return False


def generate_raw_frames():
    """
    A function to yield unprocessed frames from stored video file or ip cam stream

    Yields:
        bytes: a frame from the video file or ip cam stream
    """
    global video_frames

    while True:
        # Keep reading the frames from the video file or ip cam stream
        success, frame = video_frames.read()

        if success:
            # create a copy of the frame to store in the buffer
            frame_copy = frame.copy()

            # store the frame in the buffer for violation detection
            frames_buffer.append(frame_copy)

            # compress the frame and store it in the memory buffer
            _, buffer = cv2.imencode('.jpg', frame)
            # convert the buffer to bytes
            frame = buffer.tobytes()
            # yield the frame to the browser
            yield (b'--frame\r\n'
                   b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')


def generate_processed_frames(conf_=0.25):
```

```
    """
    A function to yield processed frames from stored video file or ip cam stream
after violation detection

    Args:
        conf_ (float, optional): confidence threshold for the detection. Defaults
to 0.25.

    Yields:
        bytes: a processed frame from the video file or ip cam stream
    """
    # to be implemented
    pass


@app.route('/video_raw')
def video_raw():
    """
    A function to handle the requests for the raw video stream

    Returns:
        Response: a response object containing the raw video stream
    """

    return Response(generate_raw_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')


@app.route('/video_processed')
def video_processed():
    """A function to handle the requests for the processed video stream after
violation detection

    Returns:
        Response: a response object containing the processed video stream
    """
    # default confidence threshold
    conf = 0.75
    return Response(generate_processed_frames(conf_=conf), mimetype='multipart/x-
mixed-replace; boundary=frame')


@app.route('/', methods=["GET", "POST"])
def index():
    """
    A function to handle the requests from the web page

    Returns:
        render_template: the index.html page (home page)
    """
    return render_template('index.html')


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

```
def submit_form():
    """
    A function to handle the requests from the HTML form on the web page

    Returns:
        str: a string containing the response message
    """
    # global variables
    # noinspection PyGlobalUndefined
    global vid_path, video_frames, frames_buffer

    # if the request is a POST request made by user interaction with the HTML form
    if request.method == "POST":
        # print(request.form)vid_ip_path.startswith('http://')

        # handle video upload request
        if request.files:
            video = request.files['video']

            # check if video file is uploaded or not
            if video.filename == '':
                # display a flash alert message on the web page
                return "That video must have a file name"

            # check if the uploaded file is a video
            elif not allowed_video(video.filename):
                # display a flash alert message on the web page
                return "Unsupported video. The video file must be in MP4, MOV,
AVI, WEBM or WMV format."
            else:
                # default video name
                filename = 'vid.mp4'
                # ensure video size is less than 200MB
                if video.content_length > 200 * 1024 * 1024:
                    return "Error! That video is too large"
                else:
                    # noinspection PyBroadException
                    try:
                        video.save(os.path.join(app.config["VIDEO_UPLOADS"],
filename))
                        return "That video is successfully uploaded"
                    except Exception as e:
                        print(e)
                        return "Error! The video could not be saved"

        # handle inference request for a video file
        elif 'inference_video_button' in request.form:
            video_frames = cv2.VideoCapture(vid_path)
            # clear the buffer of frames that may have been stored from a previous
inference
            frames_buffer.clear()
            # check if the video is opened
            if not video_frames.isOpened():
                return 'Error in opening video', 500
```

```
            else:
                frames_buffer.clear()
                return 'success'

        # handle inference request for a live stream via IP camera
        elif 'live_inference_button' in request.form:
            # to be implemented
            pass

        # handle email request# handle alert email request
        elif 'alert_email_checkbox' in request.form:
            email_checkbox_value = request.form['alert_email_checkbox']
            if email_checkbox_value == 'false':
                return "Alert email is disabled"
            else:
                alert_recipient = request.form['alert_email_textbox']
                # send email
                prepare_and_send_email(sender='hamza2019cs148@abesit.edu.in',
                                       recipient=alert_recipient,
                                       subject='Greeting from Global
Infoventures',
                                       message_text='Hello, this is a test email
from Global Infoventures',
                                       im0=cv2.imread('static/test.jpg'))
                return f"Alert email is sent at {alert_recipient} with the
attached image"

        # handle download request for the detections summary report
        elif 'download_button' in request.form:
            return Response(open('static/reports/detections_summary.txt',
'r').read(),
                            mimetype='text/plain',
                            headers={"Content-Disposition":
"attachment;filename=detections_summary.txt"})


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

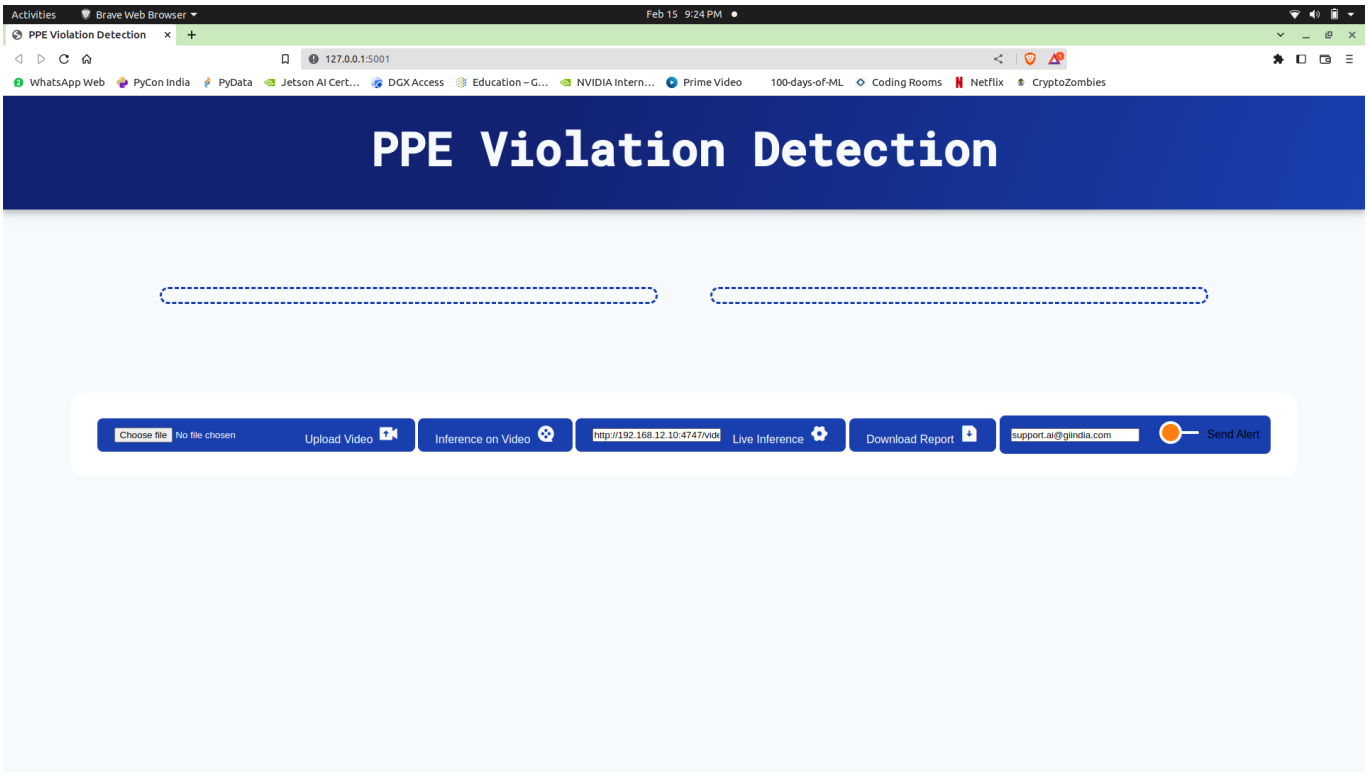
Save the file by pressing `Ctrl+X` → `Y` → `Enter`

Step 5 - Run Flask Application

Start the flask application by entering the following command in your terminal.

```
python -m flask --app app.py run
```

Your flask application should look something like this.


Activities
Brave Web Browser
Feb 15 9:24 PM
PPE Violation Detection
127.0.0.1:5001
WhatsApp Web
PyCon India
PyData
Jetson AI Cert...
DGX Access
Education – G...
NVIDIA Intern...
Prime Video
100-days-of-ML
Coding Rooms
Netflix
CryptoZombies
PPE Violation Detection
Choose file
No file chosen
Upload Video
Inference on Video
http://192.168.12.10:4747/vide
Live Inference
Download Report
support.ai@giindia.com
Send Alert