Hand Gesture Recognition Application

Developer's Guide

BOUDHINA MOHAMED

Introduction

Welcome to the developer's guide for body's Gesture Recognition Application. This application captures hand,face,body gestures, trains a classification model, and performs real-time predictions. It is built using Flask and utilizes Mediapipe for gesture detection.

Prerequisites:

INSTALL Python 3.x

Installation Instructions:

1. Download the application repository:

url_web_app_hand: https://github.com/mohamedtns/handmarker_git

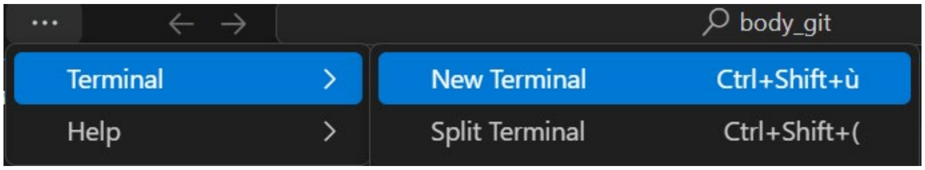
url_web_app_face: https://github.com/mohamedtns/face_git

url_web_app_body: https://github.com/mohamedtns/body_git

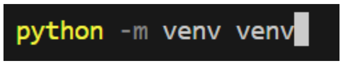
2.Open VSCODE

3.Create virtual environnement

a)open terminal in VSCODE



b)write: python -m venv venv



```
python -m venv venv
```

c) write: .\venv\Scripts\activate

```
d)view-> command palette->python:select interpreter->enter interpter
paths->find->venv->sCripts->python.exe
```

4. Install the required libraries:

Write in the terminal:

pip install flask

pip install pip install opencv-python-headless

pip install mediapipe

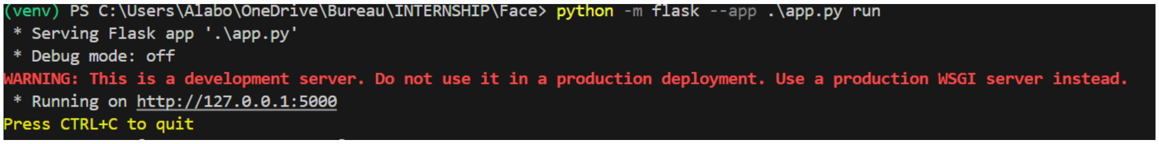
pip install pandas

pip install scikit-learn

pip install joblib

5) write: python -m flask --app .\app.py run

6) click on http://127.0.0.1:5000



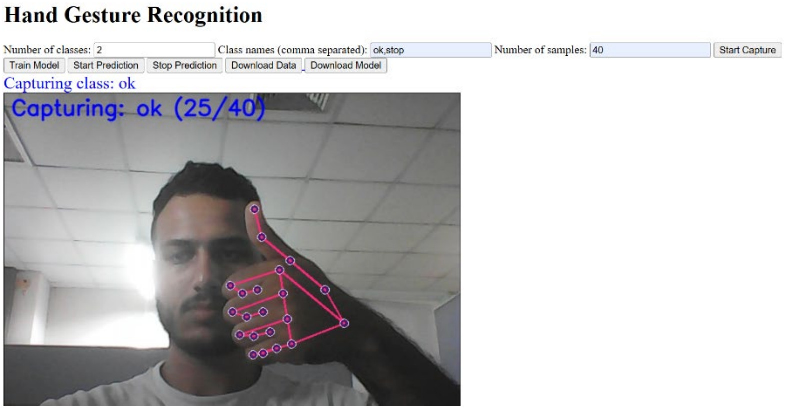
```
(venv) PS C:\Users\Alabo\OneDrive\Bureau\INTERNSHIP\Face> python -m flask --app .\app.py run
 * Serving Flask app '.\app.py'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

Using the application:

Gesture Capture:

- Fill out the form on the home page with the number of classes, class names (comma-separated), and the number of samples.

- Click "Start Capture" to begin capturing gestures.



Hand Gesture Recognition

Number of classes: 2 Class names (comma separated): ok,stop Number of samples: 40

Capturing class: ok

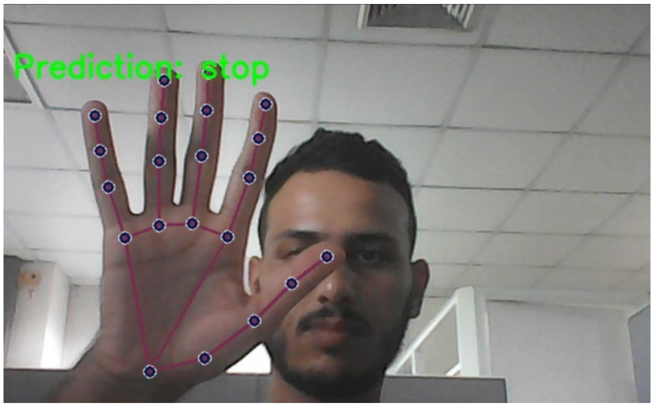
Capturing: ok (25/40)

Training the Model:

- Click the "Train Model" button to train the model with the captured data.

Gesture Prediction:

- Click the "Start Prediction" button to start gesture prediction.



- Click the "Stop Prediction" button to stop the prediction.

Downloading Data and Model:

- Click the "Download Data" button to download the captured data.

- Click the "Download Model" button to download the trained model.

Library:

1. Flask

- Flask is a lightweight WSGI web application framework in Python.

It is designed with simplicity and flexibility in mind.

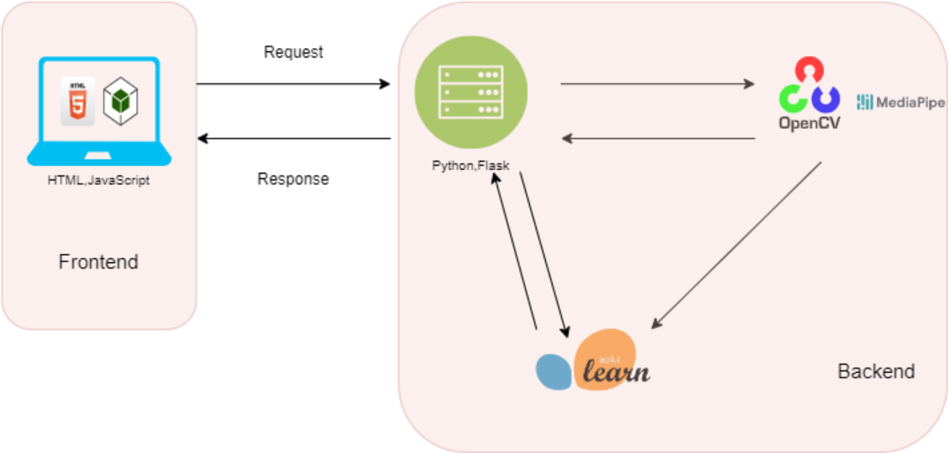
- In this application, Flask is used to create the web server that serves the web interface and handles API requests.

2. OpenCV

- OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library.

- In this application, OpenCV is used for capturing video frames from the webcam.

Explanation:



Summary in English

The hand gesture recognition web application integrates multiple technologies in a sequential process to deliver its functionality. The process begins with the Frontend (HTML, JavaScript), where the user interacts with the interface to initiate actions like data capture, model training, and predictions. These requests are sent to the Backend (Flask), which manages the routes and endpoints, orchestrating the overall workflow.

Once the backend receives a request, it employs Computer Vision technologies (OpenCV, Mediapipe) to capture and process real-time video, detecting hand landmarks necessary for gesture recognition. This processed data is then passed back to the backend, which uses Machine Learning (Scikit-learn) to train a gesture classification model on the captured data or to make real-time predictions based on the model. The results of these predictions or confirmations of data capture and model training are sent back to the frontend for user feedback, completing the interactive loop.

This diagram and summary outline the interconnectedness and order of operations among the technologies, ensuring a clear understanding of how the application functions end-to-end.

API Endpoints

- GET /: Home page of the application.

- GET /video_feed: Live video feed.

- POST /start_capture: Start capturing gestures.

- POST /train_model: Train the model.

- POST /start_prediction: Start predictions.

- POST /stop_prediction: Stop predictions.

- GET /download_data: Download captured data.

- GET /download_model: Download the trained model.

FAQ

Common Issues:

- Camera not connecting: Ensure the camera is properly connected and the drivers are up to date.

- Model not training correctly: Make sure enough data is captured for each gesture class.

Frequently Asked Questions:

- How to add new gesture classes?

  Add the new class names in the form on the home page and start capturing.

Conclusion

Thank you for using our body's Gesture Recognition Application.

For further information or assistance, please contact me here: mohamedboudhina01@hotmail.com

Python code:

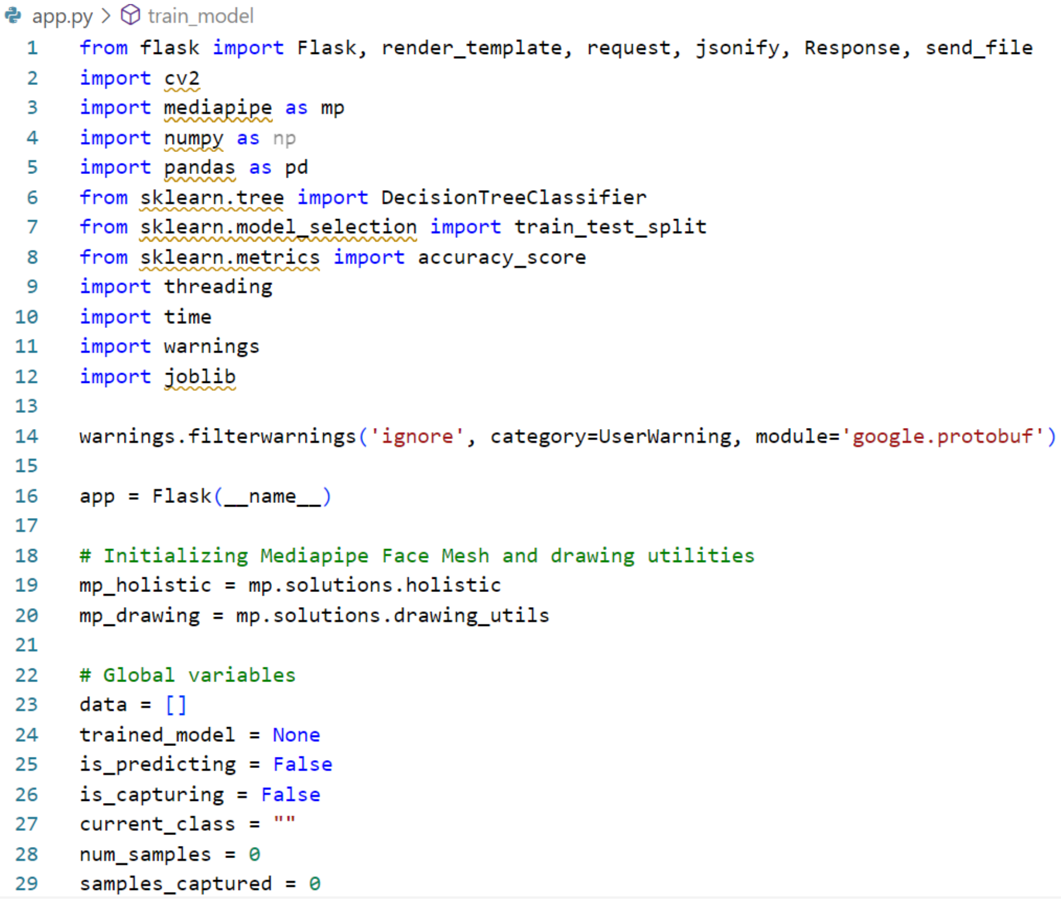
app.py:

- Contains the main application code, handling routes and functionalities for capturing, training, and predicting gestures.

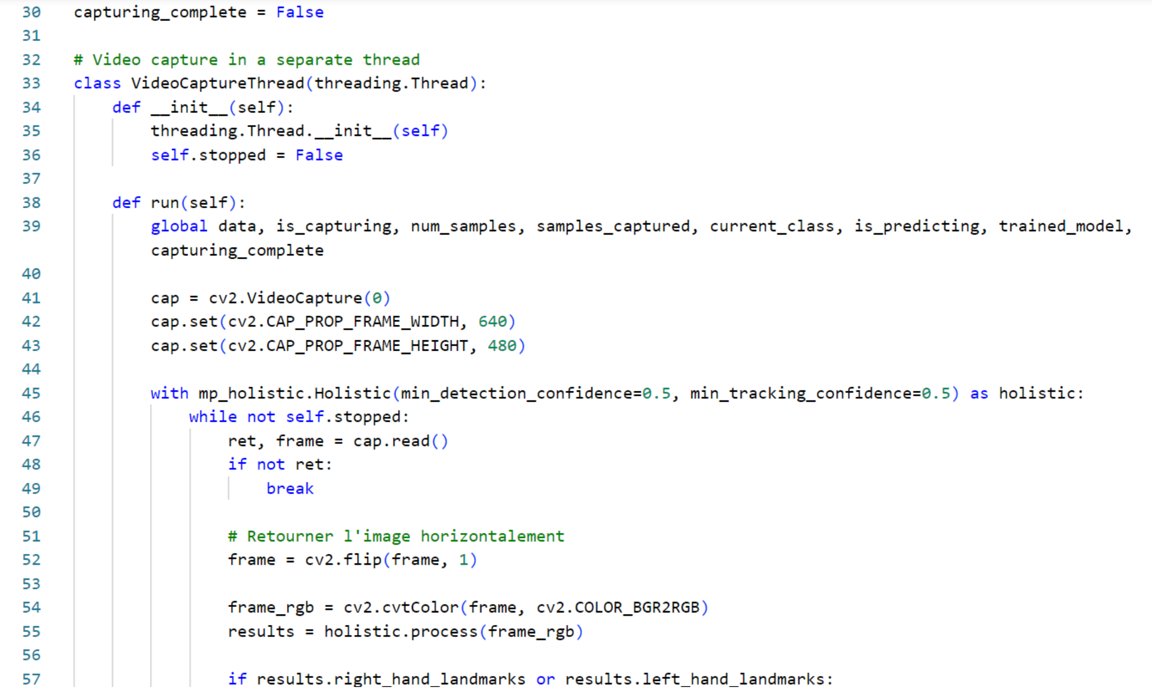
- Key functions:

  - VideoCaptureThread: Handles video capture in a separate thread.

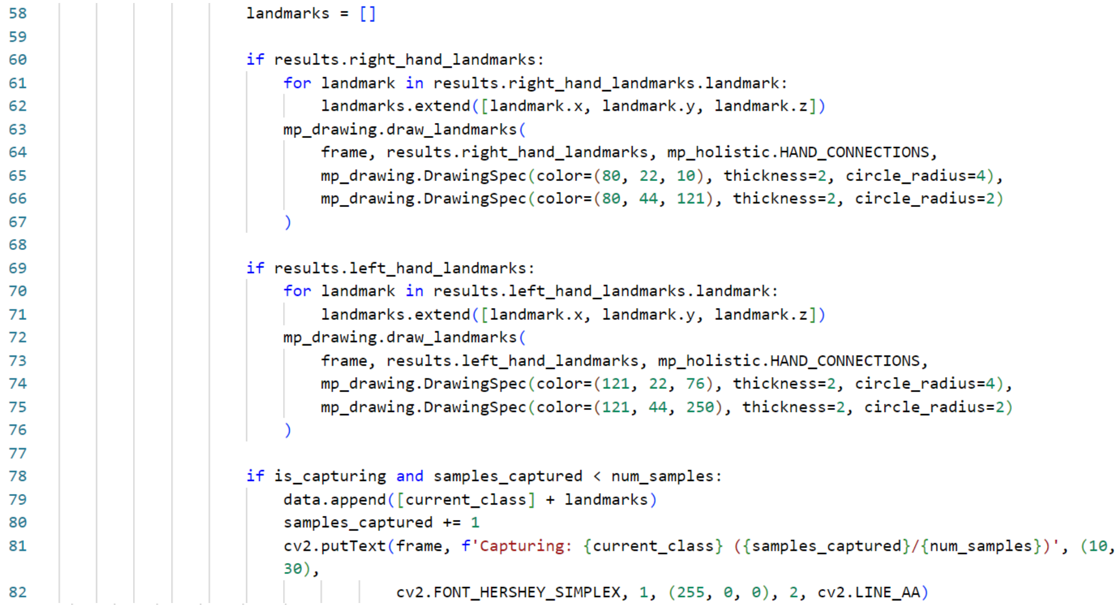
  - Routes for capturing gestures, training the model, starting and stopping predictions, and downloading data and models.



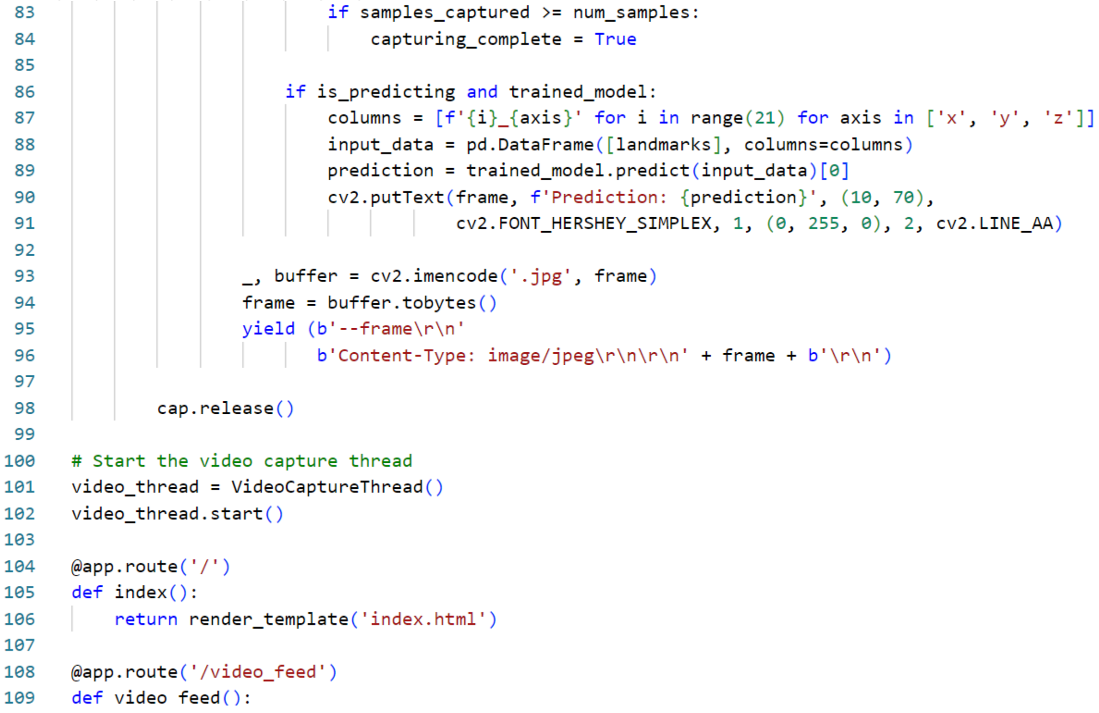
```python
from flask import Flask, render_template, request, jsonify, Response, send_file
import cv2
import mediapipe as mp
import numpy as np
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import threading
import time
import warnings
import joblib

warnings.filterwarnings('ignore', category=UserWarning, module='google.protobuf')

app = Flask(__name__)

# Initializing Mediapipe Face Mesh and drawing utilities
mp_holistic = mp.solutions.holistic
mp_drawing = mp.solutions.drawing_utils

# Global variables
data = []
trained_model = None
is_predicting = False
is_capturing = False
current_class = ""
num_samples = 0
samples_captured = 0
```



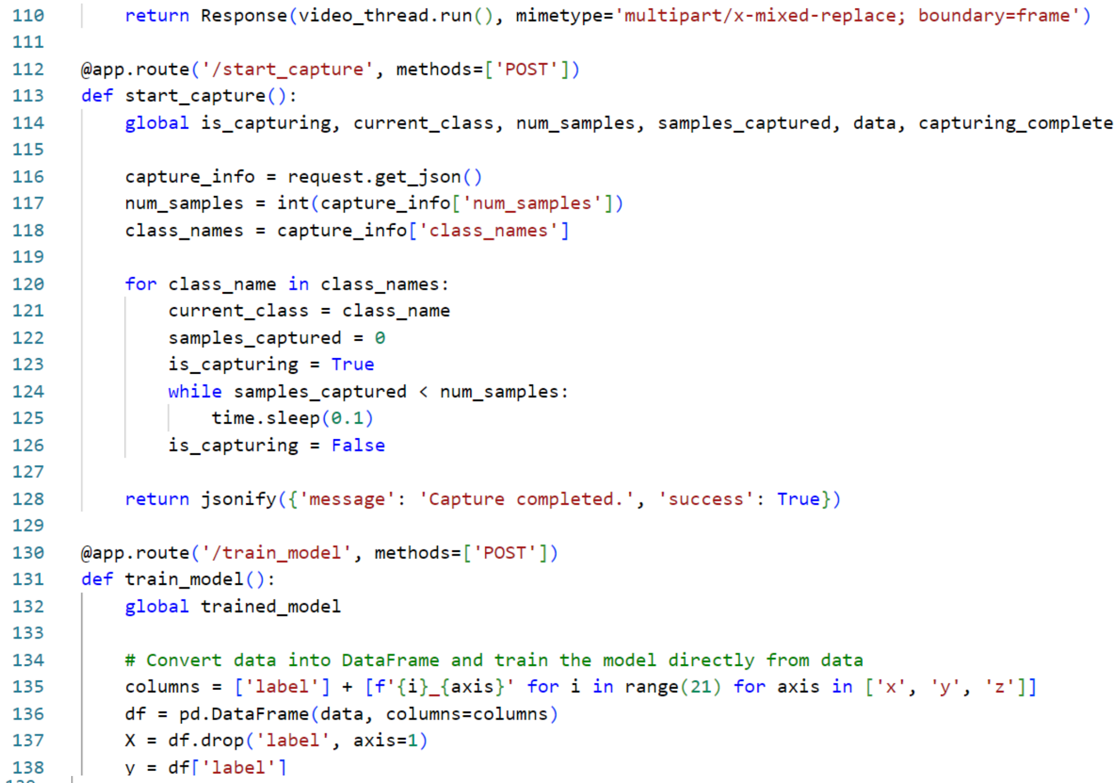
```python
capturing_complete = False

# Video capture in a separate thread
class VideoCaptureThread(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)
        self.stopped = False

    def run(self):
        global data, is_capturing, num_samples, samples_captured, current_class, is_predicting, trained_model, capturing_complete

        cap = cv2.VideoCapture(0)
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 480)

        with mp_holistic.Holistic(min_detection_confidence=0.5, min_tracking_confidence=0.5) as holistic:
            while not self.stopped:
                ret, frame = cap.read()
                if not ret:
                    break

                # Retourner l'image horizontalement
                frame = cv2.flip(frame, 1)

                frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                results = holistic.process(frame_rgb)

                if results.right_hand_landmarks or results.left_hand_landmarks:
```



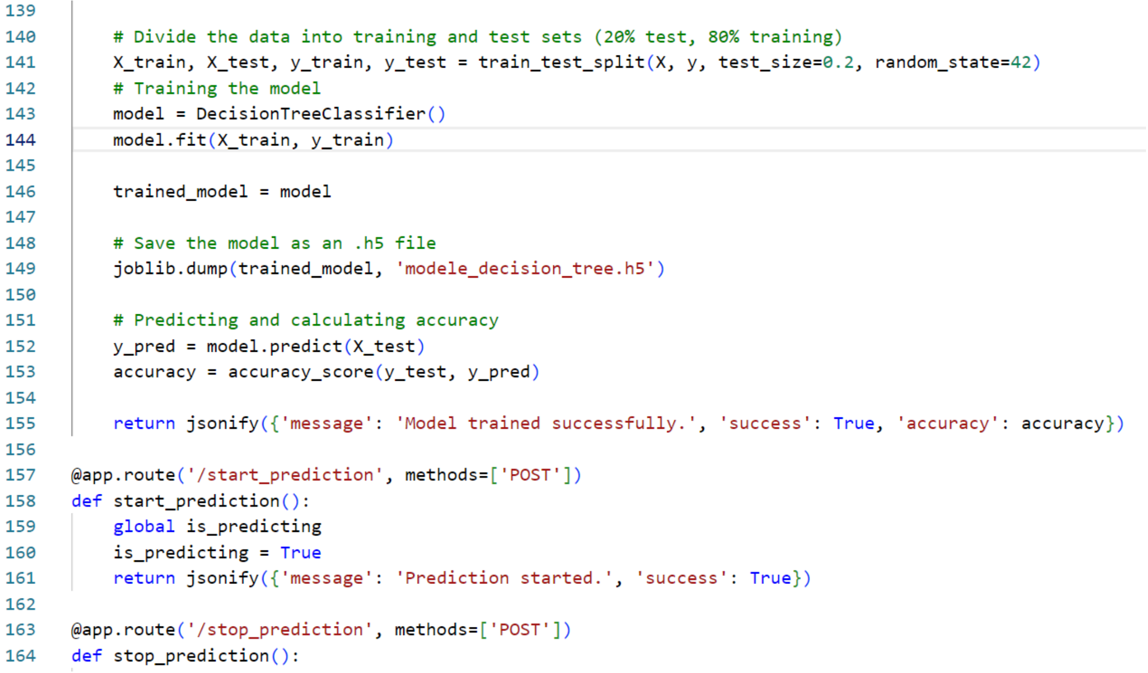
```
                    landmarks = []

                    if results.right_hand_landmarks:
                        for landmark in results.right_hand_landmarks.landmark:
                            landmarks.extend([landmark.x, landmark.y, landmark.z])
                        mp_drawing.draw_landmarks(
                            frame, results.right_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(80, 22, 10), thickness=2, circle_radius=4),
                            mp_drawing.DrawingSpec(color=(80, 44, 121), thickness=2, circle_radius=2)
                        )

                    if results.left_hand_landmarks:
                        for landmark in results.left_hand_landmarks.landmark:
                            landmarks.extend([landmark.x, landmark.y, landmark.z])
                        mp_drawing.draw_landmarks(
                            frame, results.left_hand_landmarks, mp_holistic.HAND_CONNECTIONS,
                            mp_drawing.DrawingSpec(color=(121, 22, 76), thickness=2, circle_radius=4),
                            mp_drawing.DrawingSpec(color=(121, 44, 250), thickness=2, circle_radius=2)
                        )

                    if is_capturing and samples_captured < num_samples:
                        data.append([current_class] + landmarks)
                        samples_captured += 1
                        cv2.putText(frame, f'Capturing: {current_class} ({samples_captured}/{num_samples})', (10, 30),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 0, 0), 2, cv2.LINE_AA)
```



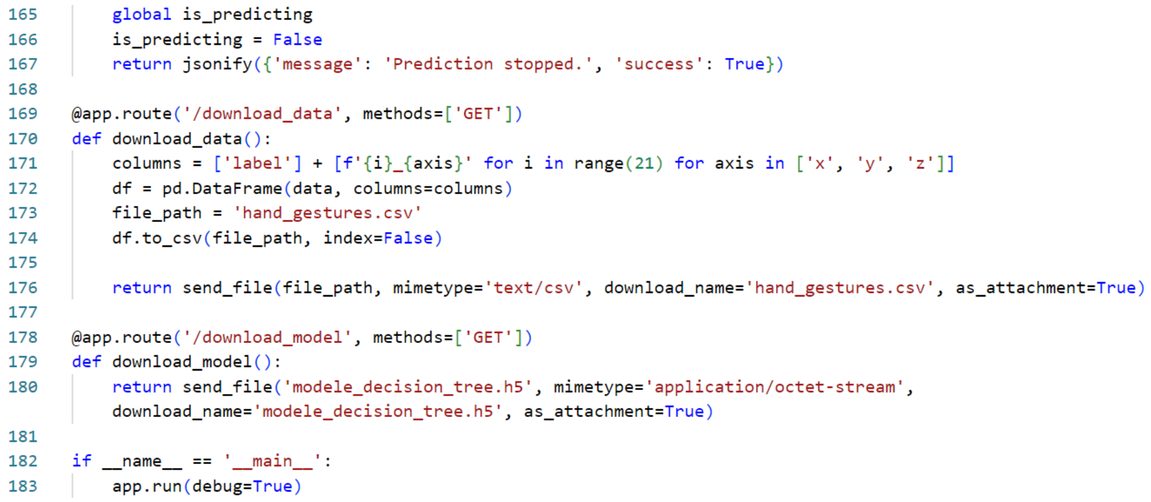
```
                        if samples_captured >= num_samples:
                            capturing_complete = True

                    if is_predicting and trained_model:
                        columns = [f'{i}_{axis}' for i in range(21) for axis in ['x', 'y', 'z']]
                        input_data = pd.DataFrame([landmarks], columns=columns)
                        prediction = trained_model.predict(input_data)[0]
                        cv2.putText(frame, f'Prediction: {prediction}', (10, 70),
                                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2, cv2.LINE_AA)

                _, buffer = cv2.imencode('.jpg', frame)
                frame = buffer.tobytes()
                yield (b'--frame\r\n'
                       b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')

        cap.release()

# Start the video capture thread
video_thread = VideoCaptureThread()
video_thread.start()

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

@app.route('/video_feed')
def video feed():
```



```
    return Response(video_thread.run(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/start_capture', methods=['POST'])
def start_capture():
    global is_capturing, current_class, num_samples, samples_captured, data, capturing_complete

    capture_info = request.get_json()
    num_samples = int(capture_info['num_samples'])
    class_names = capture_info['class_names']

    for class_name in class_names:
        current_class = class_name
        samples_captured = 0
        is_capturing = True
        while samples_captured < num_samples:
            time.sleep(0.1)
        is_capturing = False

    return jsonify({'message': 'Capture completed.', 'success': True})

@app.route('/train_model', methods=['POST'])
def train_model():
    global trained_model

    # Convert data into DataFrame and train the model directly from data
    columns = ['label'] + [f'{i}_{axis}' for i in range(21) for axis in ['x', 'y', 'z']]
    df = pd.DataFrame(data, columns=columns)
    X = df.drop('label', axis=1)
    y = df['label']
```

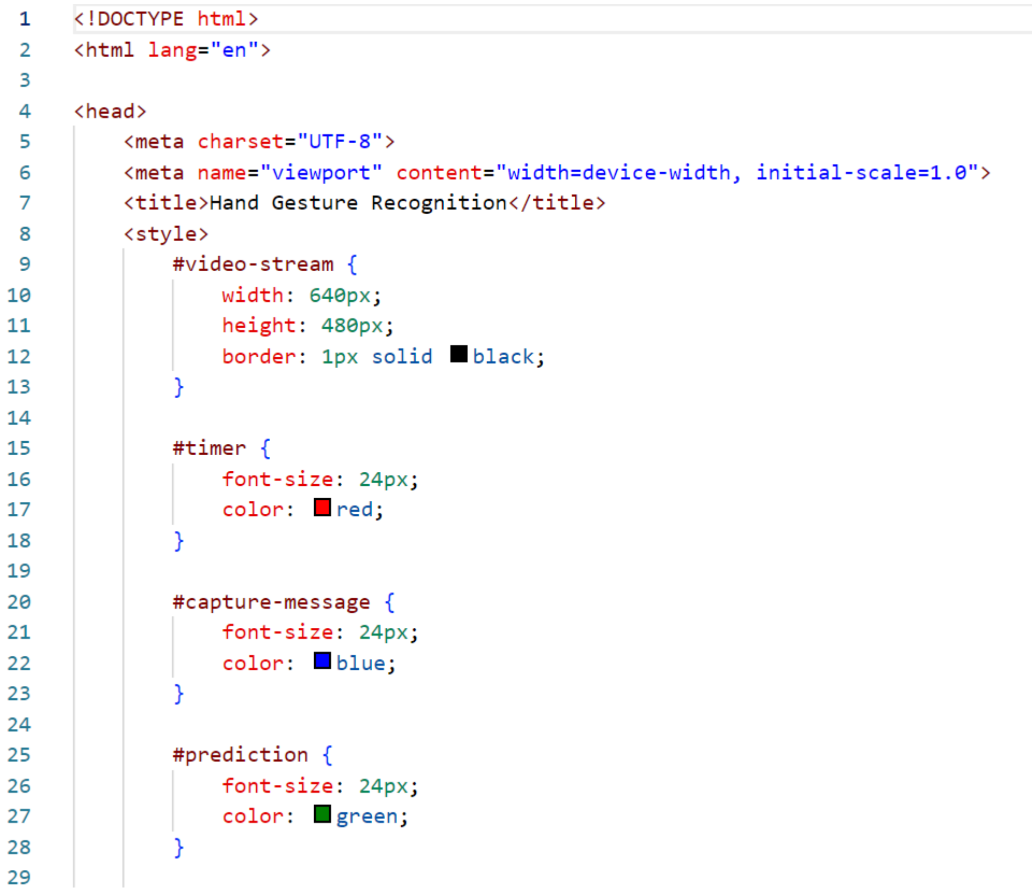


```

    # Divide the data into training and test sets (20% test, 80% training)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    # Training the model
    model = DecisionTreeClassifier()
    model.fit(X_train, y_train)

    trained_model = model

    # Save the model as an .h5 file
    joblib.dump(trained_model, 'modele_decision_tree.h5')

    # Predicting and calculating accuracy
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)

    return jsonify({'message': 'Model trained successfully.', 'success': True, 'accuracy': accuracy})

@app.route('/start_prediction', methods=['POST'])
def start_prediction():
    global is_predicting
    is_predicting = True
    return jsonify({'message': 'Prediction started.', 'success': True})

@app.route('/stop_prediction', methods=['POST'])
def stop_prediction():
```



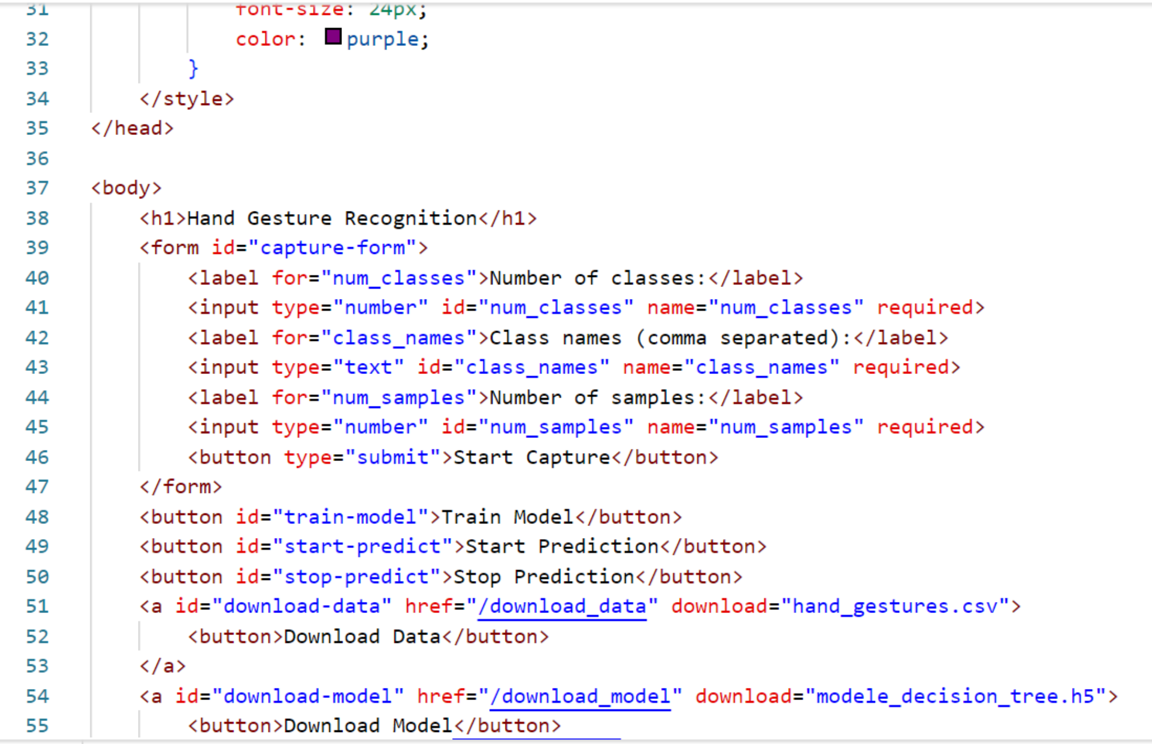
```python
    global is_predicting
    is_predicting = False
    return jsonify({'message': 'Prediction stopped.', 'success': True})

@app.route('/download_data', methods=['GET'])
def download_data():
    columns = ['label'] + [f'{i}_{axis}' for i in range(21) for axis in ['x', 'y', 'z']]
    df = pd.DataFrame(data, columns=columns)
    file_path = 'hand_gestures.csv'
    df.to_csv(file_path, index=False)

    return send_file(file_path, mimetype='text/csv', download_name='hand_gestures.csv', as_attachment=True)

@app.route('/download_model', methods=['GET'])
def download_model():
    return send_file('modele_decision_tree.h5', mimetype='application/octet-stream',
    download_name='modele_decision_tree.h5', as_attachment=True)

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

HTML CODE:

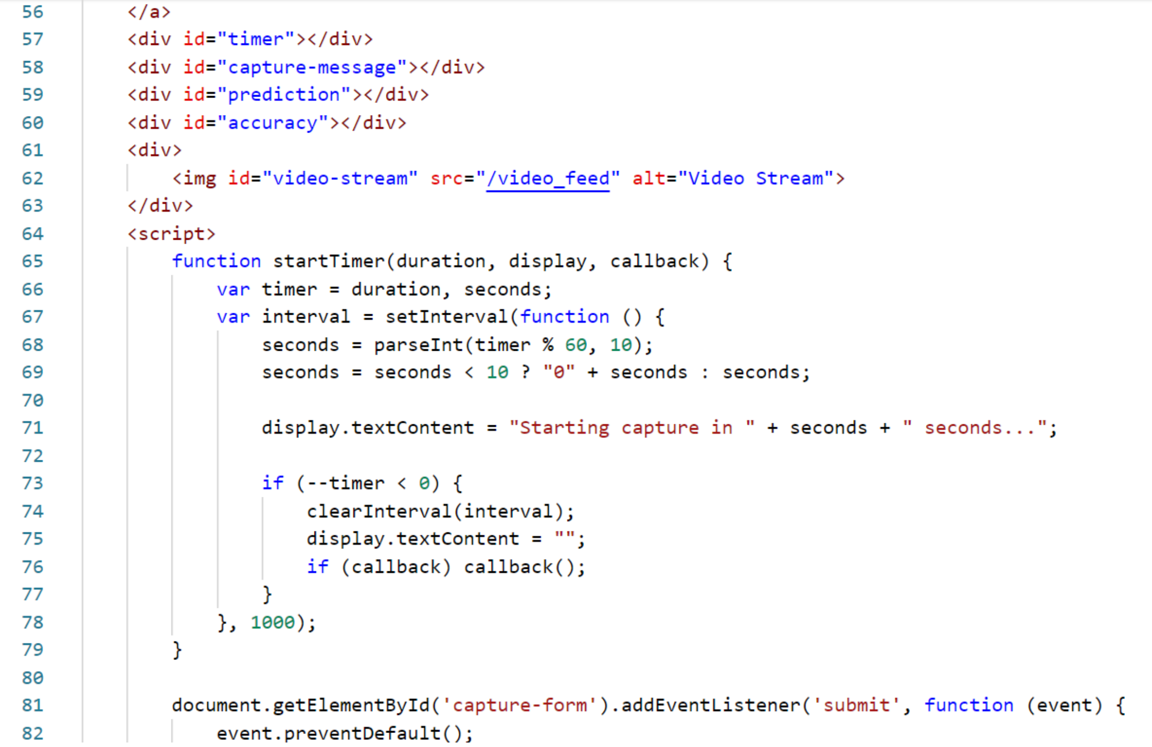


```html
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Hand Gesture Recognition</title>
    <style>
        #video-stream {
            width: 640px;
            height: 480px;
            border: 1px solid ⬛black;
        }

        #timer {
            font-size: 24px;
            color: 🟥red;
        }

        #capture-message {
            font-size: 24px;
            color: 🟦blue;
        }

        #prediction {
            font-size: 24px;
            color: 🟩green;
        }

```

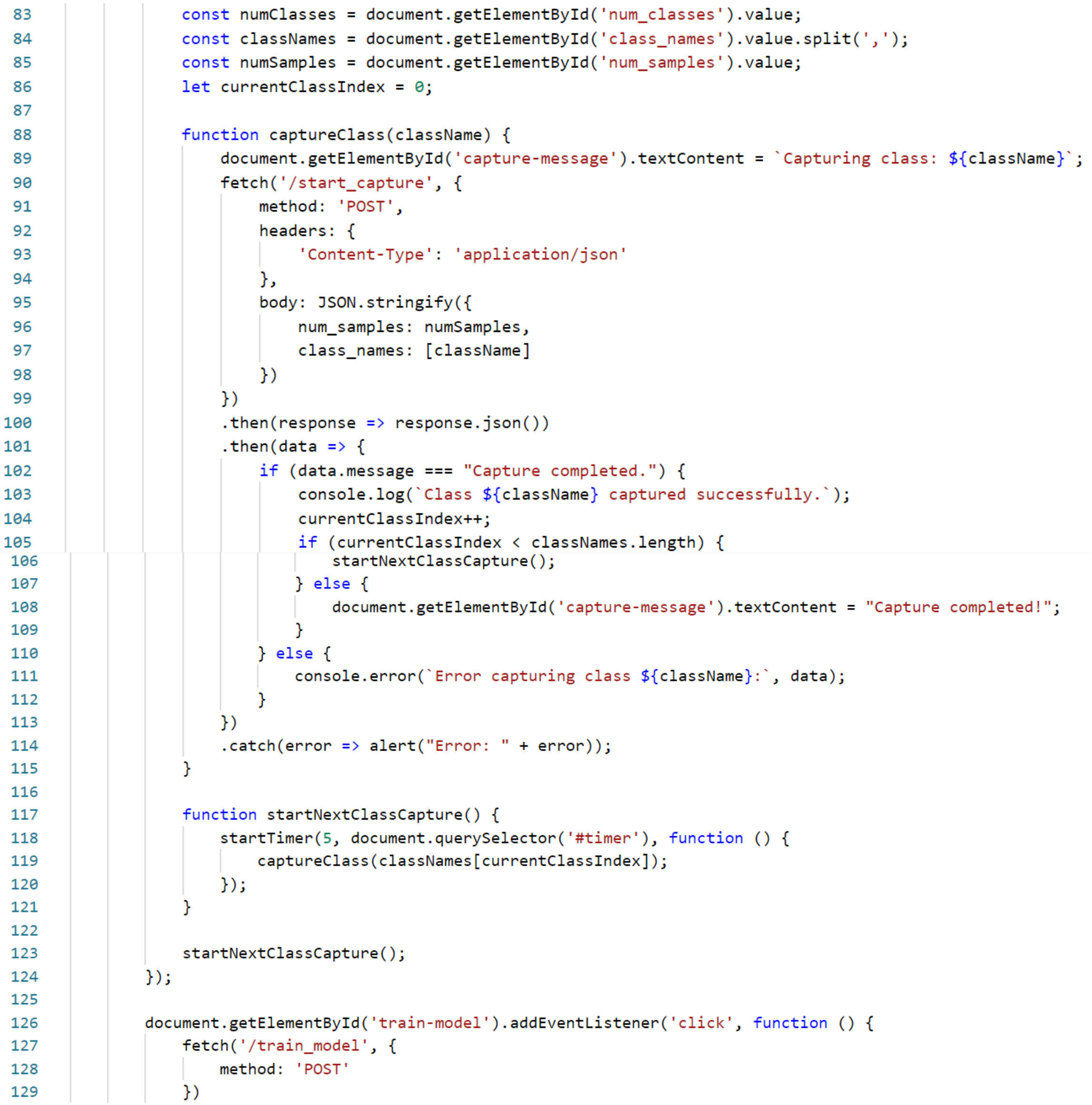
```
        #accuracy {
```



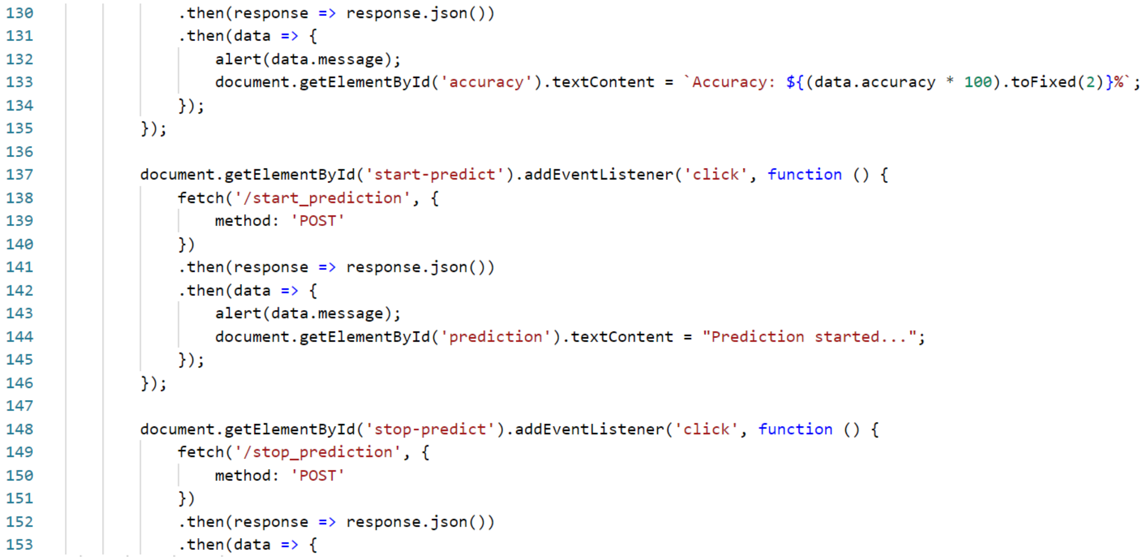
```
            font-size: 24px;
            color: purple;
        }
    </style>
</head>

<body>
    <h1>Hand Gesture Recognition</h1>
    <form id="capture-form">
        <label for="num_classes">Number of classes:</label>
        <input type="number" id="num_classes" name="num_classes" required>
        <label for="class_names">Class names (comma separated):</label>
        <input type="text" id="class_names" name="class_names" required>
        <label for="num_samples">Number of samples:</label>
        <input type="number" id="num_samples" name="num_samples" required>
        <button type="submit">Start Capture</button>
    </form>
    <button id="train-model">Train Model</button>
    <button id="start-predict">Start Prediction</button>
    <button id="stop-predict">Stop Prediction</button>
    <a id="download-data" href="/download_data" download="hand_gestures.csv">
        <button>Download Data</button>
    </a>
    <a id="download-model" href="/download_model" download="modele_decision_tree.h5">
        <button>Download Model</button>
```



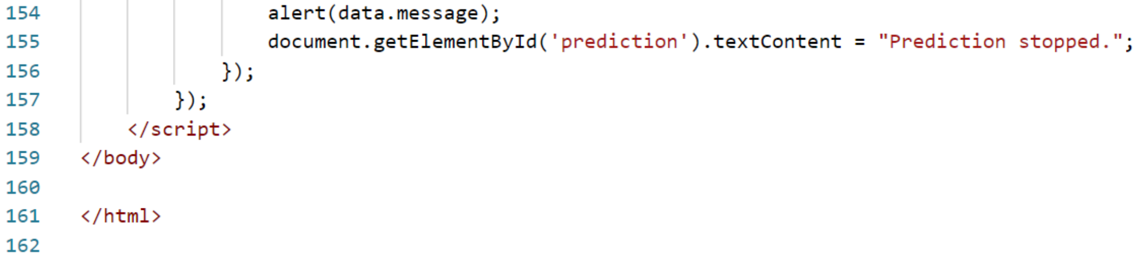
```
    </a>
    <div id="timer"></div>
    <div id="capture-message"></div>
    <div id="prediction"></div>
    <div id="accuracy"></div>
    <div>
        <img id="video-stream" src="/video_feed" alt="Video Stream">
    </div>
    <script>
        function startTimer(duration, display, callback) {
            var timer = duration, seconds;
            var interval = setInterval(function () {
                seconds = parseInt(timer % 60, 10);
                seconds = seconds < 10 ? "0" + seconds : seconds;

                display.textContent = "Starting capture in " + seconds + " seconds...";

                if (--timer < 0) {
                    clearInterval(interval);
                    display.textContent = "";
                    if (callback) callback();
                }
            }, 1000);
        }

        document.getElementById('capture-form').addEventListener('submit', function (event) {
            event.preventDefault();
```



```
            const numClasses = document.getElementById('num_classes').value;
            const classNames = document.getElementById('class_names').value.split(',');
            const numSamples = document.getElementById('num_samples').value;
            let currentClassIndex = 0;

            function captureClass(className) {
                document.getElementById('capture-message').textContent = `Capturing class: ${className}`;
                fetch('/start_capture', {
                    method: 'POST',
                    headers: {
                        'Content-Type': 'application/json'
                    },
                    body: JSON.stringify({
                        num_samples: numSamples,
                        class_names: [className]
                    })
                })
                .then(response => response.json())
                .then(data => {
                    if (data.message === "Capture completed.") {
                        console.log(`Class ${className} captured successfully.`);
                        currentClassIndex++;
                        if (currentClassIndex < classNames.length) {
                            startNextClassCapture();
                        } else {
                            document.getElementById('capture-message').textContent = "Capture completed!";
                        }
                    } else {
                        console.error(`Error capturing class ${className}:`, data);
                    }
                })
                .catch(error => alert("Error: " + error));
            }

            function startNextClassCapture() {
                startTimer(5, document.querySelector('#timer'), function () {
                    captureClass(classNames[currentClassIndex]);
                });
            }

            startNextClassCapture();
        });

        document.getElementById('train-model').addEventListener('click', function () {
            fetch('/train_model', {
                method: 'POST'
            })
```



```
            .then(response => response.json())
            .then(data => {
                alert(data.message);
                document.getElementById('accuracy').textContent = `Accuracy: ${(data.accuracy * 100).toFixed(2)}%`;
            });
        });

        document.getElementById('start-predict').addEventListener('click', function () {
            fetch('/start_prediction', {
                method: 'POST'
            })
            .then(response => response.json())
            .then(data => {
                alert(data.message);
                document.getElementById('prediction').textContent = "Prediction started...";
            });
        });

        document.getElementById('stop-predict').addEventListener('click', function () {
            fetch('/stop_prediction', {
                method: 'POST'
            })
            .then(response => response.json())
            .then(data => {
```



```
                alert(data.message);
                document.getElementById('prediction').textContent = "Prediction stopped.";
            });
        });
    </script>
</body>

</html>
```