

Sprint 4

app.py

```
# Import Necessary libraries
from datetime import timedelta
from flask import Flask, render_template, request, redirect, session,
flash, Response
from flask_sqlalchemy import SQLAlchemy
from flask_session import Session
from werkzeug.utils import secure_filename
import os
import cv2
from tensorflow.keras.preprocessing import image
import numpy as np
from keras.models import load_model
from twilio.rest import Client
from pygame import mixer

app = Flask(__name__)

UPLOAD_FOLDER = os.path.join('static', 'uploads')
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER
app.secret_key = 'admin'

# Database connection
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///user.db'
app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
db = SQLAlchemy(app)

# user cache

app.permanent_session_lifetime = timedelta(minutes=5)
app.config['SESSION_TYPE'] = 'filesystem'
Session(app)

with app.app_context():
    db.create_all()

# global camera object
camera = None

# Database Model
class Registration(db.Model):
    _id = db.Column('id', db.Integer, primary_key=True)
    name = db.Column(db.String(25), nullable=False)
    email = db.Column(db.String(120), nullable=False)
    password = db.Column(db.String(16), nullable=False)

    def __repr__(self):
        return f"Name : {self.name} Mail :{self.email}"

# Home Page
@app.route('/', methods=['GET'])
def home():
    if camera:
```

```

        camera.release()
    if session.get('name'):
        return render_template('home.html', link_image_video = True)
    else:
        return redirect('/signup')

# signup page
@app.route('/signup', methods=['POST', 'GET'])
def signup():
    if request.method == 'POST':
        name = request.form['uname']
        email = request.form['email']
        pwd = request.form['pwd']
        user = Registration.query.filter_by(name=name).first()
        if user is not None:
            return render_template('index.html', msg='Already Registered')
        user = Registration(name=name, email=email, password=pwd)
        db.session.add(user)
        db.session.commit()
        return render_template('index.html', msg='Successfully Registered')
    return render_template('index.html')

# login page
@app.route('/login', methods=['POST'])
def login():
    if request.method == 'POST':
        email = request.form['email']
        pwd = request.form['pwd']
        user = Registration.query.filter_by(email=email).first()
        if user is None:
            return render_template('index.html', msg="Invalid user!")
        if user.password != pwd:
            return render_template('index.html', msg="Invalid Password!")
        session['name'] = user.name
        return redirect('/')
    return redirect('/signup')

# Logout Page
@app.route('/logout')
def logout():
    session['name'] = None
    return redirect('/')

# About Page
@app.route('/about')
def about():
    if camera:
        camera.release()
    return render_template('about.html')

def detect(pred):
    li = ['Not Fire', 'Fire']
    return li[pred]

model = load_model(r'./static/model/ForestDetectionModel.h5')

```

```

def image_prediction(img_file):
    img = image.load_img(img_file, target_size=(150, 150))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)
    pred = model.predict(x)
    pred = detect(int(pred))
    return pred

def video_prediction(video_file):
    cap = cv2.VideoCapture(video_file)
    WIDTH = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
    HEIGHT = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
    FPS = 1
    fourcc = 0x00000021
    output = cv2.VideoWriter("./static/uploads/videos/output.mp4", fourcc,
FPS, (WIDTH, HEIGHT))
    def getFrame(sec):
        cap.set(cv2.CAP_PROP_POS_MSEC, sec * 1000)
        hasFrames, frame = cap.read()
        if hasFrames:
            cv2.imwrite('./static/uploads/images/pic.jpg', frame)
            img = image.load_img('./static/uploads/images/pic.jpg',
target_size=(150, 150))
            img = image.img_to_array(img)
            img = np.expand_dims(img, axis=0)
            pred = model.predict(img)
            if pred[0] == 1:
                cv2.circle(frame, (480, 55), 10, (0, 0, 255), -1)
                cv2.putText(frame, 'Fire Alert', (500, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2, cv2.LINE_AA)
                output.write(frame)
            return hasFrames
        sec = 0
        frameRate = 1
        count = 1
        success = getFrame(sec)
        while success:
            count = count + 1
            sec = sec + frameRate
            sec = round(sec, 2)
            success = getFrame(sec)
        output.release()
        cap.release()
        return 1

# Check the image and video extensions
def allowed_file(filename):
    image_extensions = {'jpg', 'jpeg', 'png', 'gif', 'webp'}
    video_extensions = {'webm', 'mp4', 'mov', 'avi', 'mkv'}
    ex = filename.rsplit('.', 1)[1].lower()
    if ex in image_extensions:
        return 'images'
    if ex in video_extensions:
        return 'videos'
    return None

```

```

# Predict Forest Fire using images and videos
@app.route('/predict', methods=['POST', 'GET'])
def predict():
    if request.method == 'POST':
        file = request.files['Pred_file']
        if not file:
            flash('No File Selected', 'image_video')
            return redirect('/')
        if not allowed_file(file.filename):
            flash('Invalid File Format', 'image_video')
            return redirect('/')
        if file and allowed_file(file.filename) == 'images':
            img_filename = secure_filename(file.filename)
            path = os.path.join(app.config['UPLOAD_FOLDER'], 'images',
img_filename)
            file.save(path)
            imageprediction = image_prediction(path)
            return render_template('home.html', img=img_filename,
img_prediction=imageprediction, link_image_video = True)
        if file and allowed_file(file.filename) == 'videos':
            vdo_filename = secure_filename(file.filename)
            path = os.path.join(app.config['UPLOAD_FOLDER'], 'videos',
vdo_filename)
            file.save(path)
            videoprediction = video_prediction(path)
            return render_template('home.html',
video_prediction=videoprediction, link_image_video = True)
        return redirect('/')

# send a message to alert fire
def alert_message():
    account_sid = 'AC68305e6170b2901e2d784239fb9fca5c'
    auth_token = '55e44f3c92c8e9bc70f77f5345afe7b5'
    client = Client(account_sid, auth_token)

    message = client.messages.create(
        body='Fire Alert Please Be Safe',
        from_='+14246881439',
        to='+919025937756',
    )
    return True

# Generate and predict using each frames in realtime camera
def gen_frames():
    send = False
    while True:
        success, frame = camera.read()
        if success:
            try:
                cv2.imwrite('./static/uploads/videos/pic.jpg', frame)
                img = image.load_img('./static/uploads/videos/pic.jpg',
target_size=(150, 150))
                img = image.img_to_array(img)
                img = np.expand_dims(img, axis=0)
                pred = model.predict(img)
                if pred[0][0] == 1.0:
                    cv2.circle(frame, (480, 55), 10, (0, 0, 255), -1)
                    cv2.putText(frame, 'Fire Alert', (500, 60),
cv2.FONT_HERSHEY_SIMPLEX, 0.8, (0, 0, 255), 2, cv2.LINE_AA)

```

```

        mixer.init()
        sound = mixer.Sound('./static/sound/alert1.ogg')
        sound.play()
        if not send:
            alert_messge()
            send = True
    ret, 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')

except:
    pass

# render to cv camera to html file
@app.route('/video_feed')
def video_feed():
    return Response(gen_frames(), mimetype='multipart/x-mixed-replace;
boundary=frame')

# Realtime camera page
@app.route('/camera_pred')
def camera_pred():
    global camera
    camera = cv2.VideoCapture(0)
    return render_template('/home.html', camera_start = True,
camera_prediction=True)

# camera stop method
@app.route('/camera_stop')
def camera_stop():
    camera.release()
    return render_template('/home.html', camera_start = False,
camera_prediction=True)

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

```

about.html

```

<!DOCTYPE html>
<html>
<head>
    <title>Forest Fire Detection</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static',
filename = 'css/about.css') }}">
    <link
href="https://fonts.googleapis.com/css2?family=Jost:wght@500&display=swap"
rel="stylesheet">
</head>
<body>
    <div id="header">

```

```

<div class="title">
  <h1>Emerging Methods for Early Detection of Forest Fires</h1>
</div>
<div class="sections">
  <a href="{{url_for('home')}}">Image and video</a>
  <a href="{{url_for('camera_pred')}}">Camera</a>
  <a href="{{url_for('about')}}">About</a>
  <a href="{{url_for('logout')}}">Logout</a>
</div>
</div>

<div id="content">
  <p>
    Forest fires are a major environmental issue, creating economic
    and ecological damage
    while endangering human lives. There are typically about 100,000
    wildfires in the United
    States every year. Over 9 million acres of land have been
    destroyed due to treacherous
    wildfires. It is difficult to predict and detect Forest Fire in a
    sparsely populated forest
    area and it is more difficult if the prediction is done using
    ground-based methods like
    Camera or Video-Based approach. Satellites can be an important
    source of data prior to
    and also during the Fire due to its reliability and efficiency.
    The various real-time
    forest fire detection and prediction approaches, with the goal of
    informing the local
    fire authorities.
  </p>
</div>

<div class="team_details">
  <h2>Project Team Members</h2>
  <h3>Santhana selvam chandru (Leader)</h3>
  <h3>Harish Ragavendar</h3>
  <h3>Vikneswar</h3>
  <h3>Suresh Raja Perumal</h3>
</div>

</body>
</html>

```

about.css

```

body{
  margin: 0;
  padding: 0;
  min-height: 100vh;
  font-family: 'Jost', sans-serif;
  color: #fff;
  background: linear-gradient(to bottom, #0f0c29, #302b63, #24243e);
}
#header{
  display: flex;
  position: relative;
  flex-direction: row;

```

```
        margin : 0.5em;
    }
    #header .title{
        text-align: left;
    }
    #header h1{
        font-size: 1.5em;
    }

    #header .sections{
        position: absolute;
        top:1em;
        right:0;
        font-size: 1em;
    }
    #header .sections a{
        margin-right : 1.5em;
        text-decoration: none;
        color: #fff;
    }

    #content p{
        text-align:center;
        font-size: 1em;
        text-align: justify;
        margin: 0.3em;
        margin-left:3em;
        margin-right:3em;
        padding: 0.5em;
        line-height: 2;
    }

    .team_details{
        text-align: center;
        margin-top: 1em;
    }
}
```