WEB CAMERA

```
import os
import cv2
import numpy as np
import tensorflow as tf
from fastapi import FastAPI, File, UploadFile
from fastapi.responses import JSONResponse
from fastapi.middleware.cors import CORSMiddleware
from tensorflow.keras.preprocessing import image
from tensorflow.keras.models import load_model
import uvicorn

# ------------------------------
# Load the Model
# ------------------------------
model_path = "best_model.h5"
model = load_model(model_path)

class_names = ['ants', 'bees', 'beetle', 'catterpillar', 'earthworms', 'earwig',
               'grasshopper', 'moth', 'slug', 'snail', 'wasp', 'weevil']
img_size = (224, 224)

# ------------------------------
# FastAPI Setup
# ------------------------------
app = FastAPI()
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
```

```
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# ----------------------------
# Helper Functions
# ----------------------------
def preprocess_image(file_bytes):
    img = image.load_img(file_bytes, target_size=img_size)
    img_array = image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)
    return tf.keras.applications.efficientnet.preprocess_input(img_array)

def predict(image_tensor):
    predictions = model.predict(image_tensor, verbose=0)
    predicted_index = int(tf.argmax(predictions[0]))
    confidence = float(predictions[0][predicted_index])
    label = class_names[predicted_index]

    # Check if confidence is above 30% for an insect class
    insect_classes = ['ants', 'bees', 'beetle', 'catterpillar', 'earwig', 'grasshopper', 'moth', 'slug', 'snail',
'wasp', 'weevil']
    flag = label in insect_classes and confidence > 0.30

    return label, confidence, flag

# ----------------------------
# Endpoints
# ----------------------------
@app.post("/predict/image")
```

```
async def predict_image(file: UploadFile = File(...)):
    contents = await file.read()
    with open("temp.jpg", "wb") as f:
        f.write(contents)

    tensor = preprocess_image("temp.jpg")
    label, confidence, flag = predict(tensor)
    os.remove("temp.jpg")
    return JSONResponse({"prediction": label, "confidence": round(confidence, 2), "flag": flag})

@app.post("/predict/video")
async def predict_video(file: UploadFile = File(...)):
    contents = await file.read()
    with open("temp_video.mp4", "wb") as f:
        f.write(contents)
    cap = cv2.VideoCapture("temp_video.mp4")
    frame_count = 0
    results = []
    while cap.isOpened():
        ret, frame = cap.read()
        if not ret or frame_count >= 10:
            break
        frame_resized = cv2.resize(frame, img_size)
        input_tensor =
tf.expand_dims(tf.keras.applications.efficientnet.preprocess_input(frame_resized.astype("float32")),
0)
        label, confidence, flag = predict(input_tensor)
        results.append({"frame": frame_count, "prediction": label, "confidence": round(confidence, 2),
"flag": flag})
        frame_count += 1
    cap.release()
    os.remove("temp_video.mp4")
```

```
    return JSONResponse({"predictions": results})


@app.get("/predict/webcam")
def predict_webcam():
    cap = cv2.VideoCapture(0)
    predictions = []
    frame_count = 0

    while cap.isOpened() and frame_count < 10:
        ret, frame = cap.read()
        if not ret:
            break
        frame_resized = cv2.resize(frame, img_size)
        input_tensor =
tf.expand_dims(tf.keras.applications.efficientnet.preprocess_input(frame_resized.astype("float32")),
0)
        label, confidence, flag = predict(input_tensor)
        predictions.append({"frame": frame_count, "prediction": label, "confidence": round(confidence,
2), "flag": flag})
        frame_count += 1

    cap.release()
    return JSONResponse({"predictions": predictions})


if __name__ == "__main__":
    uvicorn.run("main:app", host="0.0.0.0", port=3665, reload=True)
```

```
import cv2
import tensorflow as tf
import numpy as np

# ------------------------------
# Load the Model
# ------------------------------
model_path = "best_model.h5"
model = tf.keras.models.load_model(model_path)
print("[INFO] Model loaded successfully.")

# ------------------------------
# Class Names (adjust if needed)
# ------------------------------
class_names = ['ants', 'bees', 'beetle', 'catterpillar', 'earthworms', 'earwig',
               'grasshopper', 'moth', 'slug', 'snail', 'wasp', 'weevil']

# ------------------------------
# Video Source
# ------------------------------
# Use 0 for webcam or provide path to a video file like "video.mp4"
video_source = 0 # Change to "your_video.mp4" if testing with file

cap = cv2.VideoCapture(video_source)

img_size = (224, 224)

# ------------------------------
```

```
# Frame Preprocessing
# ----------------------------
def preprocess_frame(frame):
    frame_resized = cv2.resize(frame, img_size)
    img_array = tf.keras.applications.efficientnet.preprocess_input(frame_resized.astype("float32"))
    return tf.expand_dims(img_array, 0)

# ----------------------------
# Video Loop
# ----------------------------
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
        break

    # Preprocess and predict
    input_tensor = preprocess_frame(frame)
    predictions = model.predict(input_tensor, verbose=0)
    predicted_index = int(tf.argmax(predictions[0]))
    confidence = predictions[0][predicted_index]
    predicted_class = class_names[predicted_index]

    # Display prediction
    label = f"{predicted_class} ({confidence:.2f})"
    cv2.putText(frame, label, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)
    cv2.imshow("Real-Time Insect Prediction", frame)
    # Press 'q' to quit
    if cv2.waitKey(1) & 0xFF == ord("q"):
        break
cap.release()
cv2.destroyAllWindows()
```

SOIL MOISTURE LEVEL

```
import eventlet
eventlet.monkey_patch()

from flask import Flask, render_template, request, jsonify
from flask_socketio import SocketIO, emit
import os
import random
import json
from werkzeug.utils import secure_filename
import time
from threading import Thread
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'static/uploads'
app.config['SECRET_KEY'] = 'secret!'
socketio = SocketIO(app)

# Ensure upload folder exists
os.makedirs(app.config['UPLOAD_FOLDER'], exist_ok=True)

# JSON database setup
DATABASE = 'database.json'
moisture_thread = None

def init_db():
    if not os.path.exists(DATABASE):
        with open(DATABASE, 'w') as db_file:
```

```
            json.dump({'equipment': [], 'orders': []}, db_file)

def read_db():
    with open(DATABASE, 'r') as db_file:
        return json.load(db_file)

def write_db(data):
    with open(DATABASE, 'w') as db_file:
        json.dump(data, db_file, indent=4)

init_db()

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

@app.route('/api/equipment', methods=['GET', 'POST'])
def equipment_api():
    if request.method == 'POST':
        name = request.form['name']
        cost_per_day = float(request.form['cost_per_day'])
        image = request.files['image']
        filename = secure_filename(image.filename)
        image.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

        db = read_db()
        db['equipment'].append({
            'id': len(db['equipment']) + 1,
            'name': name,
            'image': filename,
            'cost_per_day': cost_per_day
```

```
        })
        write_db(db)
        return jsonify({'message': 'Equipment added successfully!'}), 201
    else:
        db = read_db()
        return jsonify({'equipment': db['equipment']})

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

@app.route('/orders', methods=['POST'])
def orders():
    data = request.json
    equipment_id = data['equipment_id']
    quantity = data['quantity']

    db = read_db()
    db['orders'].append({
        'id': len(db['orders']) + 1,
        'equipment_id': equipment_id,
        'quantity': quantity
    })
    write_db(db)
    return jsonify({'message': 'Order placed successfully!'}), 201

@app.route('/api/cart', methods=['GET', 'POST', 'DELETE'])
def cart():
    db = read_db()
    if request.method == 'POST':
        item = request.json
```

```
        db.setdefault('cart', []).append(item)
        write_db(db)
        return jsonify({'message': 'Item added to cart!'}), 201
    elif request.method == 'DELETE':
        db['cart'] = []
        write_db(db)
        return jsonify({'message': 'Cart cleared!'}), 200
    else:
        return jsonify({'cart': db.get('cart', [])})

@app.route('/api/orders', methods=['GET', 'POST'])
def orders_api():
    db = read_db()
    if request.method == 'POST':
        order = {
            'id': len(db['orders']) + 1,
            'items': db.get('cart', []),
            'total': sum(item['cost_per_day'] * item['quantity'] for item in db.get('cart', []))
        }
        db['orders'].append(order)
        db['cart'] = []
        write_db(db)
        return jsonify({'message': 'Order placed successfully!', 'order': order}), 201
    else:
        return jsonify({'orders': db['orders']})

sensors = [
    {"sensor_id": i, "level": 0, "lat": random.uniform(13.154096, 13.154096 + 0.001), "lng":
random.uniform(79.778280, 79.778280 + 0.001)}
    for i in range(1, 11)
]
```

```
def generate_soil_moisture():
    with app.app_context():
        while True:
            for sensor in sensors:
                sensor["level"] = random.randint(0, 100)
            socketio.emit('moisture_update', {"sensors": sensors})
            time.sleep(10)

@app.route('/api/sensors', methods=['GET'])
def get_sensors():
    return jsonify({"sensors": sensors})

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

@socketio.on('connect')
def handle_connect():
    global moisture_thread
    if moisture_thread is None:
        moisture_thread = socketio.start_background_task(generate_soil_moisture)

if __name__ == '__main__':
        socketio.run(app, host='0.0.0.0', port=3045, debug=True)
```