KRISHIAI

app.py

from flask import Flask, render\_template, request

import os, logging

from PIL import Image

import numpy as np

from tensorflow.keras.models import load\_model

# Setup

app = Flask(\_\_name\_\_)

app.config['UPLOAD\_FOLDER'] = 'uploads'

ALLOWED\_EXTENSIONS = {'png', 'jpg', 'jpeg'}

os.makedirs(app.config['UPLOAD\_FOLDER'], exist\_ok=True)

logging.basicConfig(level=logging.INFO)

# Load trained model

model = load\_model('models/plant\_model.h5')

class\_labels = {0: 'Healthy', 1: 'Leaf Blight', 2: 'Rust'}

# Utility

def allowed\_file(filename):

return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED\_EXTENSIONS

def preprocess\_image(file):

img = Image.open(file.stream).resize((224, 224))

img\_array = np.array(img) / 255.0

return np.expand\_dims(img\_array, axis=0)

# Home Page

@app.route('/')

def home():

return render\_template('index.html')

# Scan Page (GET + POST)

@app.route('/scan', methods=['GET', 'POST'])

def scan():

if request.method == 'POST':

image = request.files.get('image')

if image and allowed\_file(image.filename):

try:

img\_array = preprocess\_image(image)

prediction = model.predict(img\_array)

result = np.argmax(prediction)

label = class\_labels[result]

return render\_template('scan.html', prediction=label)

except Exception as e:

logging.error(f"Scan error: {e}")

return render\_template('scan.html', error="Failed to process image.")

return render\_template('scan.html', error="Invalid or missing image.")

# GET request: just show the scan page

return render\_template('scan.html')

# Run the app

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

=================================================================================index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>KrishiAI | Empowering Farmers with AI</title>

<!-- Link to static CSS and JS -->

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<link href="https://fonts.googleapis.com/css2?family=Roboto:wght@400;700&display=swap" rel="stylesheet">

<script defer src="{{ url\_for('static', filename='script.js') }}"></script>

</head>

<body>

<div class="progress-container">

<div class="progress-bar" id="progressBar"></div>

</div>

<header>

<nav>

<h1>🌾 KrishiAI</h1>

<ul>

<li><a href="#about">About</a></li>

<li><a href="#features">Features</a></li>

<li><a href="#team">Team</a></li>

<li><a href="#contact">Contact</a></li>

</ul>

</nav>

</header>

<section class="hero">

<h2>AI for Every Farmer</h2>

<p>KrishiAI is your intelligent companion in the field—identifying plants, optimizing yield, and making agriculture smarter and more accessible.</p>

<a href="#features" class="btn">Explore Features</a>

<a href="{{ url\_for('scan') }}" class="btn scan-link">Try KrishiAI</a>

</section>

<section id="about" class="intro">

<h2>🌿 About KrishiAI</h2>

<p>Born from a passion for technology and empathy for farmers, KrishiAI blends computer vision and machine learning to solve real agricultural challenges. Whether you're identifying crop diseases or planning your next harvest, KrishiAI is here to help.</p>

</section>

<section id="features" class="features">

<h2>🚀 Features</h2>

<ul>

<li><strong>Plant Identification:</strong> Snap a photo, get instant insights.</li>

<li><strong>Yield Prediction:</strong> AI-powered forecasting for smarter planning.</li>

<li><strong>Offline Access:</strong> Designed for rural connectivity constraints.</li>

<li><strong>Farmer-Friendly UI:</strong> Simple, animated, and multilingual.</li>

</ul>

</section>

<section id="team" class="team">

<h2>👨‍💻 Meet the Team</h2>

<div class="member">

<h3>Harshit</h3>

<p>Lead Developer & Visionary. BTech CSE student at Manipal University Jaipur. Passionate about making tech approachable for farmers and blending AI with empathy.</p>

</div>

<div class="member">

<h3>Simrat</h3>

<p>UI/UX Designer. Brings intuitive design and accessibility to KrishiAI, ensuring farmers can interact with ease and confidence.</p>

</div>

<div class="member">

<h3>Bhumi</h3>

<p>Data Analyst & Researcher. Focused on agricultural datasets and model accuracy, helping KrishiAI deliver reliable insights to farmers.</p>

</div>

<div class="member">

<h3>Shams</h3>

<p>Backend Engineer. Integrates AI models and optimizes performance, making sure KrishiAI runs smoothly even in low-connectivity zones.</p>

</div>

</section>

<section id="contact" class="contact">

<h2>📬 Contact Us</h2>

<p>Have feedback or want to collaborate? Reach out at <a href="mailto:krishiai@support.com">krishiai@support.com</a></p>

</section>

<footer>

<p>&copy; 2025 KrishiAI. Built with ❤️ by Harshit and team.</p>

</footer>

</body>

</html>

=================================================================================scan.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<meta name="viewport" content="width=device-width, initial-scale=1.0" />

<title>KrishiAI | Scan Plant</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

<script defer src="{{ url\_for('static', filename='script.js') }}"></script>

</head>

<body>

<div class="progress-container">

<div id="progressBar" class="progress-bar"></div>

</div>

<header>

<nav>

<h1>🌾 KrishiAI</h1>

<ul>

<li><a href="{{ url\_for('home') }}">Home</a></li>

<li><a href="{{ url\_for('scan') }}">Scan</a></li> <!-- Fixed link to match route -->

</ul>

</nav>

</header>

<section class="scan-section">

<h2>📷 Scan Your Plant</h2>

<p>Use your camera or upload a photo to identify plant species and detect crop health.</p>

<!-- Live Camera Capture -->

<div class="camera-box">

<video id="camera" autoplay playsinline></video>

<canvas id="snapshot" style="display:none;"></canvas>

<button id="captureBtn" class="btn">📸 Capture</button>

</div>

<!-- Image Upload Form -->

<form action="{{ url\_for('scan') }}" method="POST" enctype="multipart/form-data">

<label for="image">Upload a plant image:</label>

<input type="file" name="image" accept="image/\*" required>

<button type="submit" class="btn">Scan Image</button>

</form>

</section>

<!-- Scanning Overlay -->

<div id="scanningOverlay" class="scanning-overlay">

<div class="loader-text">Scanning<span class="dots"></span></div>

</div>

<!-- Result Display -->

<div id="resultContainer">

{% if prediction %}

<div class="result-card">

<h3>🧪 Prediction: {{ prediction }}</h3>

</div>

{% elif error %}

<div class="error-card">

<p>{{ error }}</p>

</div>

{% endif %}

</div>

<footer>

<p>&copy; 2025 KrishiAI. Built with ❤️ by Harshit and team.</p>

</footer>

</body>

</html>

=================================================================================script.js

window.addEventListener('scroll', () => {

const scrollTop = window.scrollY;

const docHeight = document.body.scrollHeight - window.innerHeight;

const scrollPercent = (scrollTop / docHeight) \* 100;

const progressBar = document.getElementById('progressBar');

if (progressBar) {

progressBar.style.width = scrollPercent + '%';

}

});

document.addEventListener('DOMContentLoaded', () => {

if (window.location.pathname.includes('scan')) {

const video = document.getElementById('camera');

const canvas = document.getElementById('snapshot');

const captureBtn = document.getElementById('captureBtn');

const overlay = document.getElementById('scanningOverlay');

if (navigator.mediaDevices && navigator.mediaDevices.getUserMedia) {

navigator.mediaDevices.getUserMedia({ video: true })

.then(stream => {

video.srcObject = stream;

})

.catch(err => {

console.error("Camera access failed:", err);

});

}

captureBtn.addEventListener('click', () => {

canvas.style.display = 'block';

canvas.width = video.videoWidth;

canvas.height = video.videoHeight;

canvas.getContext('2d').drawImage(video, 0, 0);

if (overlay) {

overlay.style.display = 'flex';

setTimeout(() => {

overlay.style.display = 'none';

}, 3000);

}

});

}

const scanForm = document.getElementById('scanForm');

if (scanForm) {

scanForm.addEventListener('submit', async (e) => {

e.preventDefault();

const formData = new FormData(e.target);

const res = await fetch('/scan', {

method: 'POST',

body: formData

});

const data = await res.json();

displayResult(data);

// Optionally call showResultCard(data) if you want to show a card

// showResultCard(data);

});

}

});

function displayResult(data) {

const container = document.getElementById('resultContainer');

container.innerHTML = `

<h2>${data.name}</h2>

<p><strong>Uses:</strong> ${data.uses}</p>

<p><strong>How to Grow:</strong> ${data.growth}</p>

<p><strong>Care Guide:</strong> ${data.care}</p>

<p><strong>Reminders:</strong></p>

<ul>

<li>💧 Water: ${data.reminders.water}</li>

<li>🌱 Fertilize: ${data.reminders.fertilize}</li>

<li>✂️ Prune: ${data.reminders.prune}</li>

</ul>

`;

}

// Optional: Result card function (call it if you add a #result-card element in your HTML)

function showResultCard(data) {

const card = document.getElementById("result-card");

if (card) {

card.classList.add("fade-in");

card.innerHTML = `

<h3>${data.label}</h3>

<p>Confidence: ${data.confidence}%</p>

<p>Suggestion: ${data.suggestion}</p>

`;

}

}

=================================================================================style.css

/\* Base styles \*/

body {

font-family: 'Roboto', sans-serif;

margin: 0;

padding: 0;

line-height: 1.6;

background-color: #f9f9f9;

color: #333;

}

header {

background-color: #4caf50;

color: white;

padding: 1rem;

position: sticky;

top: 0;

z-index: 1000;

}

nav {

display: flex;

justify-content: space-between;

align-items: center;

}

nav ul {

list-style: none;

display: flex;

gap: 1rem;

}

nav a {

color: white;

text-decoration: none;

font-weight: bold;

}

.hero {

text-align: center;

padding: 4rem 2rem;

background: linear-gradient(to right, #81c784, #aed581);

}

.hero h2 {

font-size: 2.5rem;

margin-bottom: 1rem;

}

.hero .btn {

display: inline-block;

margin-top: 1rem;

padding: 0.75rem 1.5rem;

background-color: #388e3c;

color: white;

text-decoration: none;

border-radius: 5px;

}

section {

padding: 2rem;

}

.features ul {

padding-left: 1.5rem;

}

.team .member {

background-color: #e8f5e9;

padding: 1rem;

border-radius: 8px;

margin-top: 1rem;

}

footer {

text-align: center;

padding: 1rem;

background-color: #4caf50;

color: white;

}

/\* Progress Bar Styling \*/

.progress-container {

width: 100%;

height: 8px;

background-color: #e0e0e0;

position: sticky;

top: 0;

z-index: 999;

}

.progress-bar {

height: 100%;

width: 0%;

background: linear-gradient(to right, #4caf50, #81c784);

transition: width 0.25s ease-in-out;

}

/\* Scanning Overlay \*/

.scanning-overlay {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background: rgba(76, 175, 80, 0.8); /\* semi-transparent green \*/

display: flex;

justify-content: center;

align-items: center;

z-index: 1001;

display: none;

}

.loader-text {

font-size: 2rem;

color: white;

font-weight: bold;

animation: fadeIn 0.5s ease-in-out;

}

.dots::after {

content: '';

animation: dots 1.5s steps(3, end) infinite;

}

@keyframes dots {

0% { content: ''; }

33% { content: '.'; }

66% { content: '..'; }

100% { content: '...'; }

}

@keyframes fadeIn {

from { opacity: 0; }

to { opacity: 1; }

}

.scan-section {

padding: 3rem 2rem;

text-align: center;

background-color: #e8f5e9;

}

.scan-section h2 {

font-size: 2rem;

color: #388e3c;

margin-bottom: 0.5rem;

}

.scan-section p {

font-size: 1rem;

color: #555;

margin-bottom: 2rem;

}

.camera-box {

display: flex;

flex-direction: column;

align-items: center;

gap: 1rem;

}

video {

width: 100%;

max-width: 400px;

border: 2px solid #4caf50;

border-radius: 8px;

}

canvas {

max-width: 400px;

border: 2px dashed #81c784;

border-radius: 8px;

}

.btn {

padding: 0.75rem 1.5rem;

background-color: #388e3c;

color: white;

border: none;

border-radius: 5px;

font-weight: bold;

cursor: pointer;

}

.connect-btn {

padding: 10px 20px;

background-color: #4CAF50;

color: white;

border: none;

border-radius: 6px;

text-decoration: none;

font-size: 16px;

cursor: pointer;

transition: background-color 0.3s ease;

}

.connect-btn:hover {

background-color: #45a049;

}

.fade-in {

animation: fadeIn 0.6s ease-out forwards;

}

@keyframes fadeIn {

from { opacity: 0; transform: translateY(20px); }

to { opacity: 1; transform: translateY(0); }

}

=================================================================================train\_model.py

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# Image preprocessing

datagen = ImageDataGenerator(rescale=1./255, validation\_split=0.2)

train\_data = datagen.flow\_from\_directory(

'dataset/',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='training'

)

val\_data = datagen.flow\_from\_directory(

'dataset/',

target\_size=(224, 224),

batch\_size=32,

class\_mode='categorical',

subset='validation'

)

# Model architecture

model = Sequential([

Conv2D(32, (3,3), activation='relu', input\_shape=(224,224,3)),

MaxPooling2D(2,2),

Conv2D(64, (3,3), activation='relu'),

MaxPooling2D(2,2),

Flatten(),

Dense(128, activation='relu'),

Dense(train\_data.num\_classes, activation='softmax')

])

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Train it

model.fit(train\_data, validation\_data=val\_data, epochs=10)

import os

# Create the models folder if it doesn't exist

os.makedirs('models', exist\_ok=True)

# Save the trained model

model.save('models/plant\_model.h5')

preprocess.py

from tensorflow.keras.preprocessing.image import load\_img, img\_to\_array import numpy as np

**Define your class labels (adjust as per your model)**

CLASS\_NAMES = ['Healthy', 'Leaf Spot', 'Blight', 'Rust']

def preprocess(file): """ Preprocess uploaded image for model prediction. - Resize to match model input - Normalize pixel values - Expand dimensions for batch input """ img = load\_img(file, target\_size=(224, 224)) # adjust if your model uses a different size img\_array = img\_to\_array(img) / 255.0 # normalize to [0, 1] return np.expand\_dims(img\_array, axis=0) # shape: (1, 224, 224, 3)

def decode\_prediction(pred): """ Convert model output to human-readable label and confidence. """ class\_index = np.argmax(pred) confidence = float(np.max(pred)) label = CLASS\_NAMES[class\_index] return { 'label': label, 'confidence': round(confidence \* 100, 2) # percentage format }