Arduino Code

// Pin Definitions

#define PIR1\_PIN 12

#define PIR2\_PIN 13

#define TRIG\_PIN 5

#define ECHO\_PIN 18

#define BUZZER\_PIN 27

// Motion detection variables

unsigned long lastMotionTime = 0;

const int motionGap = 3000; // 3 seconds allowed between PIR1 and PIR2 detection

bool pir1Triggered = false;

bool pir2Triggered = false;

// Ultrasonic

float lastDistance = 0;

void setup() {

Serial.begin(115200);

pinMode(PIR1\_PIN, INPUT);

pinMode(PIR2\_PIN, INPUT);

pinMode(TRIG\_PIN, OUTPUT);

pinMode(ECHO\_PIN, INPUT);

pinMode(BUZZER\_PIN, OUTPUT);

digitalWrite(BUZZER\_PIN, LOW);

Serial.println("ESP32 Ready - Waiting for motion...");

}

void loop() {

bool motion1 = digitalRead(PIR1\_PIN);

bool motion2 = digitalRead(PIR2\_PIN);

unsigned long currentTime = millis();

if (motion1 && !pir1Triggered) {

pir1Triggered = true;

lastMotionTime = currentTime;

Serial.println("PIR1 Triggered");

}

if (motion2 && pir1Triggered && (currentTime - lastMotionTime <= motionGap)) {

pir2Triggered = true;

Serial.println("PIR2 Triggered");

}

if (pir1Triggered && pir2Triggered) {

float distance = getUltrasonicDistance();

Serial.print("Distance: ");

Serial.println(distance);

if (isMovementApproaching(distance)) {

Serial.println("APPROACHING OBJECT DETECTED");

digitalWrite(BUZZER\_PIN, HIGH);

delay(500);

digitalWrite(BUZZER\_PIN, LOW);

delay(500);

digitalWrite(BUZZER\_PIN, HIGH);

delay(500);

digitalWrite(BUZZER\_PIN, LOW);

delay(500);

digitalWrite(BUZZER\_PIN, HIGH);

delay(500);

digitalWrite(BUZZER\_PIN, LOW);

delay(500);

digitalWrite(BUZZER\_PIN, HIGH);

delay(500);

digitalWrite(BUZZER\_PIN, LOW);

delay(500);

digitalWrite(BUZZER\_PIN, HIGH);

delay(500);

digitalWrite(BUZZER\_PIN, LOW);

delay(500);

// Notify laptop to run Python CV script

Serial.println("RUN\_CV");

// Wait for Python response

while (Serial.available() == 0) {

delay(10);

}

String response = Serial.readStringUntil('\n');

response.trim();

if (response == "animal") {

Serial.println("Animal Confirmed");

triggerBuzzer(200);

} else {

Serial.println("Not an animal");

}

}

pir1Triggered = false;

pir2Triggered = false;

}

if (pir1Triggered && (currentTime - lastMotionTime > motionGap)) {

pir1Triggered = false;

pir2Triggered = false;

Serial.println("Motion timeout - reset");

}

delay(300);

}

float getUltrasonicDistance() {

digitalWrite(TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(10);

digitalWrite(TRIG\_PIN, LOW);

long duration = pulseIn(ECHO\_PIN, HIGH, 20000);

if (duration == 0) return -1;

return (duration \* 0.0343) / 2;

}

bool isMovementApproaching(float currentDistance) {

if (currentDistance < 0) return false;

bool approaching = false;

if (lastDistance > 0 && currentDistance < (lastDistance - 5)) {

approaching = true;

}

lastDistance = currentDistance;

return approaching;

}

void triggerBuzzer(int durationMs) {

digitalWrite(BUZZER\_PIN, HIGH);

delay(durationMs);

digitalWrite(BUZZER\_PIN, LOW);

}

CV code:

CV handler:

import serial

import time

import subprocess

# Replace with your actual COM port (check Arduino IDE or Device Manager)

ser = serial.Serial('COM7', 115200, timeout=1)

time.sleep(2) # wait for serial connection to stabilize

print("Listening for command...")

while True:

if ser.in\_waiting:

line = ser.readline().decode(errors='ignore').strip()

print("Received:", line)

if line == "RUN\_CV":

print("Running computer vision model...")

# Option 1: Call your actual Python script

result = subprocess.run(["python", "loadandrun.py"], capture\_output=True, text=True)

# Extract result from output (edit this if needed)

prediction = result.stdout.strip()

print("Model output:", prediction)

# Option 2 (Simplified for testing):

# prediction = "animal" # or "not\_animal"

# Send back to Arduino

ser.write((prediction + "\n").encode())

CV main:

import os

import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.models import Sequential

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

from tensorflow.keras.preprocessing.image import ImageDataGenerator

from tensorflow.keras.applications import MobileNetV2

from tensorflow.keras.applications.mobilenet\_v2 import preprocess\_input

from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

# Define constants

IMG\_SIZE = 224 # MobileNetV2 default input size

BATCH\_SIZE = 32

EPOCHS = 20

DATASET\_PATH = "big\_cats\_dataset" # Path to the downloaded dataset

MODEL\_PATH = "big\_cats\_model.h5"

# Function to load and prepare the dataset

def prepare\_dataset(dataset\_path):

# Data augmentation for training

train\_datagen = ImageDataGenerator(

preprocessing\_function=preprocess\_input,

rotation\_range=20,

width\_shift\_range=0.2,

height\_shift\_range=0.2,

shear\_range=0.2,

zoom\_range=0.2,

horizontal\_flip=True,

validation\_split=0.2

)

# Load training dataset

train\_generator = train\_datagen.flow\_from\_directory(

dataset\_path,

target\_size=(IMG\_SIZE, IMG\_SIZE),

batch\_size=BATCH\_SIZE,

class\_mode='categorical',

subset='training'

)

# Load validation dataset

validation\_generator = train\_datagen.flow\_from\_directory(

dataset\_path,

target\_size=(IMG\_SIZE, IMG\_SIZE),

batch\_size=BATCH\_SIZE,

class\_mode='categorical',

subset='validation'

)

return train\_generator, validation\_generator

# Function to create the model

def create\_model(num\_classes):

# Use MobileNetV2 as the base model for transfer learning

base\_model = MobileNetV2(

weights='imagenet',

include\_top=False,

input\_shape=(IMG\_SIZE, IMG\_SIZE, 3)

)

# Freeze the base model layers

base\_model.trainable = False

# Create the model

model = Sequential([

base\_model,

tf.keras.layers.GlobalAveragePooling2D(),

Dense(128, activation='relu'),

Dropout(0.5),

Dense(num\_classes, activation='softmax')

])

# Compile the model

model.compile(

optimizer=tf.keras.optimizers.Adam(learning\_rate=0.001),

loss='categorical\_crossentropy',

metrics=['accuracy']

)

return model

# Function to train the model

def train\_model(model, train\_generator, validation\_generator):

# Define callbacks

early\_stopping = EarlyStopping(

monitor='val\_loss',

patience=5,

restore\_best\_weights=True

)

model\_checkpoint = ModelCheckpoint(

MODEL\_PATH,

monitor='val\_accuracy',

save\_best\_only=True

)

# Train the model

history = model.fit(

train\_generator,

epochs=EPOCHS,

validation\_data=validation\_generator,

callbacks=[early\_stopping, model\_checkpoint]

)

return history, model

def detect\_from\_camera(model, class\_indices):

# Reverse the class indices dictionary to map indices to class names

class\_names = {v: k for k, v in class\_indices.items()}

# Start video capture

cap = cv2.VideoCapture(0) # 0 for default camera

# Higher confidence threshold as requested

CONFIDENCE\_THRESHOLD = 0.985 # Only detect if confidence is above 97%

# Flag to track whether an animal is currently being detected

animal\_detected = False

while True:

ret, frame = cap.read()

if not ret:

break

# Prepare the frame for prediction

resized\_frame = cv2.resize(frame, (IMG\_SIZE, IMG\_SIZE))

preprocessed\_frame = preprocess\_input(resized\_frame)

input\_data = np.expand\_dims(preprocessed\_frame, axis=0)

# Make prediction

prediction = model.predict(input\_data)

predicted\_class\_idx = np.argmax(prediction[0])

confidence = prediction[0][predicted\_class\_idx]

predicted\_class = class\_names[predicted\_class\_idx]

# Skip detection if the predicted class is "leopard"

if predicted\_class.lower() not in ["lion","tiger","cheetah"]:

text = "No Animal Detected"

color = (0, 0, 255) # Red for not detected

# Update terminal message if state changed

if animal\_detected:

print("No animals detected")

animal\_detected = False

# Only show label if confidence is high and it's not a leopard

elif confidence >= CONFIDENCE\_THRESHOLD:

text = f"{predicted\_class}: {confidence \* 100:.2f}%"

color = (0, 255, 0) # Green for detected

# Update terminal message if state changed

if not animal\_detected:

print(f"Animal detected: {predicted\_class} with confidence {confidence \* 100:.2f}%")

animal\_detected = True

else:

text = "No Animal Detected"

color = (0, 0, 255) # Red for not detected

# Update terminal message if state changed

if animal\_detected:

print("No animals detected")

animal\_detected = False

# Display the result on the frame

cv2.putText(frame, text, (10, 30), cv2.FONT\_HERSHEY\_SIMPLEX,

1, color, 2, cv2.LINE\_AA)

# Show the frame

cv2.imshow('Big Cats Detection', frame)

# Exit on 'q' key

if cv2.waitKey(1) & 0xFF == ord('q'):

break

cap.release()

cv2.destroyAllWindows()

# Main function

def main():

print("Loading and preparing the dataset...")

train\_generator, validation\_generator = prepare\_dataset(DATASET\_PATH)

num\_classes = len(train\_generator.class\_indices)

print(f"Found {num\_classes} classes: {train\_generator.class\_indices}")

print("Creating and training the model...")

model = create\_model(num\_classes)

history, trained\_model = train\_model(model, train\_generator, validation\_generator)

print("Starting live detection. Press 'q' to quit.")

detect\_from\_camera(trained\_model, train\_generator.class\_indices)

if \_\_name\_\_ == "\_\_main\_\_":

main()

User Dashboard:

Html code:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Wildlife Ranger Dashboard</title>

<link rel="stylesheet" href="style.css">

<script src="https://cdnjs.cloudflare.com/ajax/libs/Chart.js/3.9.1/chart.min.js"></script>

</head>

<body>

<div class="dashboard-container">

<header>

<div class="logo">

<h1>Wildlife Ranger Dashboard</h1>

</div>

<div class="user-info">

<span class="username">Park Ranger: Boopathi</span>

<span class="date-time" id="currentDateTime">Loading...</span>

</div>

</header>

<main>

<div class="dashboard-content">

<div class="stats-container">

<div class="stat-card">

<h3>Total Animals Detected</h3>

<div class="stat-value">249</div>

<div class="stat-trend positive">+12% from yesterday</div>

</div>

<div class="stat-card">

<h3>Species Identified</h3>

<div class="stat-value">16</div>

<div class="stat-trend neutral">Same as yesterday</div>

</div>

<div class="stat-card">

<h3>Endangered Species</h3>

<div class="stat-value">3</div>

<div class="stat-trend negative">-1 from yesterday</div>

</div>

<div class="stat-card">

<h3>Avg. Detection Distance</h3>

<div class="stat-value">125m</div>

<div class="stat-trend positive">+15m from yesterday</div>

</div>

</div>

<div class="analytics-section">

<h3 class="section-title">Analytics Overview</h3>

<div class="chart-grid">

<div class="chart-container">

<h3>Detections by Hour</h3>

<canvas id="hourlyDetectionsChart"></canvas>

</div>

<div class="chart-container">

<h3>Species Distribution</h3>

<canvas id="speciesDistributionChart"></canvas>

</div>

<div class="chart-container">

<h3>Movement Patterns</h3>

<canvas id="movementPatternsChart"></canvas>

</div>

<div class="chart-container">

<h3>Detection by Region</h3>

<canvas id="regionDetectionChart"></canvas>

</div>

</div>

</div>

</div>

<div class="recent-activity">

<h3>Recent Detections</h3>

<div class="activity-list">

<div class="activity-item">

<div class="activity-details">

<h4>Tiger (1)</h4>

<p>Detected at North Ridge Trail - 0.8 miles from visitor center</p>

<span class="activity-time">Today, 11:23 AM</span>

</div>

<div class="activity-actions">

<button class="btn-notify" data-animal="Whitetail Deer" data-location="North Ridge Trail" data-image="/api/placeholder/400/300">Send Alert</button>

</div>

</div>

<div class="activity-item">

<div class="activity-details">

<h4>Cheetah(1)</h4>

<p>Detected at Pine Creek Crossing - 1.2 miles from visitor center</p>

<span class="activity-time">Today, 10:17 AM</span>

</div>

<div class="activity-actions">

<button class="btn-notify" data-animal="Black Bear" data-location="Pine Creek Crossing" data-image="/api/placeholder/400/300">Send Alert</button>

</div>

</div>

<div class="activity-item">

<div class="activity-details">

<h4>Lion (1)</h4>

<p>Detected at South Meadow - 1.5 miles from visitor center</p>

<span class="activity-time">Today, 9:45 AM</span>

</div>

<div class="activity-actions">

<button class="btn-notify" data-animal="Coyote" data-location="South Meadow" data-image="/api/placeholder/400/300">Send Alert</button>

</div>

</div>

<div class="activity-item">

<div class="activity-details">

<h4>Tiger (2)</h4>

<p>Detected at Cedar Lake Trail - 2.3 miles from visitor center</p>

<span class="activity-time">Today, 8:12 AM</span>

</div>

<div class="activity-actions">

<button class="btn-notify" data-animal="Moose" data-location="Cedar Lake Trail" data-image="/api/placeholder/400/300">Send Alert</button>

</div>

</div>

</div>

</div>

</main>

<div id="notification-overlay" class="hidden">

<div class="notification-container">

<h2>WILDLIFE ALERT</h2>

<div class="notification-content">

<div class="notification-image">

<img id="notification-img" src="" alt="Wildlife Alert">

</div>

<div class="notification-details">

<h3 id="notification-title"></h3>

<p id="notification-location"></p>

<p id="notification-message"></p>

</div>

</div>

<div class="notification-actions">

<button id="notification-send">Send to All Citizens</button>

<button id="notification-cancel">Cancel</button>

</div>

</div>

</div>

<div id="alert-box" class="hidden">

<div class="alert-content">

<h2>NOTIFICATION SENT!</h2>

<p>Citizens have been alerted about the wildlife sighting.</p>

<button id="alert-close">Close</button>

</div>

</div>

</div>

<script src="script.js"></script>

</body>

</html>

CSS code:

/\* === BASE STYLES === \*/

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

}

:root {

--primary-color: #1d3557;

--secondary-color: #457b9d;

--accent-color: #e63946;

--background-color: #f1faee;

--card-color: #fff;

--text-color: #333;

--border-radius: 8px;

--box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);

}

body {

background-color: var(--background-color);

color: var(--text-color);

min-height: 100vh;

overflow-x: hidden;

}

.dashboard-container {

display: flex;

flex-direction: column;

min-height: 100vh;

width: 100%;

}

/\* === HEADER === \*/

header {

background-color: var(--primary-color);

color: white;

padding: 1rem 2rem;

display: flex;

justify-content: space-between;

align-items: center;

box-shadow: 0 2px 5px rgba(0, 0, 0, 0.2);

}

.logo h1 {

font-size: 1.5rem;

font-weight: 600;

}

.user-info {

display: flex;

flex-direction: column;

align-items: flex-end;

}

.username {

font-weight: 600;

}

.date-time {

font-size: 0.9rem;

opacity: 0.9;

}

/\* === MAIN CONTENT === \*/

main {

flex: 1;

padding: 1.5rem;

display: flex;

flex-direction: column;

gap: 1.5rem;

}

.dashboard-content {

display: flex;

flex-direction: column;

gap: 1.5rem;

}

.analytics-section {

background-color: var(--card-color);

border-radius: var(--border-radius);

padding: 1rem;

box-shadow: var(--box-shadow);

}

.section-title {

color: var(--primary-color);

font-size: 1.1rem;

margin-bottom: 1rem;

border-bottom: 1px solid #eee;

padding-bottom: 0.5rem;

}

/\* === STATS CARDS === \*/

.stats-container {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 1.5rem;

}

.stat-card {

background-color: var(--card-color);

border-radius: var(--border-radius);

padding: 1.25rem;

box-shadow: var(--box-shadow);

display: flex;

flex-direction: column;

gap: 0.75rem;

}

.stat-card h3 {

color: var(--secondary-color);

font-size: 1rem;

font-weight: 500;

}

.stat-value {

font-size: 2rem;

font-weight: 700;

color: var(--primary-color);

}

.stat-trend {

font-size: 0.85rem;

display: flex;

align-items: center;

}

.stat-trend.positive {

color: #38b000;

}

.stat-trend.negative {

color: var(--accent-color);

}

.stat-trend.neutral {

color: #777;

}

/\* === CHART GRID === \*/

.chart-grid {

display: grid;

grid-template-columns: repeat(4, 1fr);

gap: 1rem;

margin-bottom: 1rem;

}

.chart-container {

background-color: var(--card-color);

border-radius: var(--border-radius);

padding: 0.75rem;

box-shadow: var(--box-shadow);

height: 180px;

}

.chart-container h3 {

color: var(--secondary-color);

font-size: 0.85rem;

font-weight: 500;

margin-bottom: 0.5rem;

}

/\* === RECENT ACTIVITY === \*/

.recent-activity {

background-color: var(--card-color);

border-radius: var(--border-radius);

padding: 1.25rem;

box-shadow: var(--box-shadow);

}

.recent-activity h3 {

color: var(--secondary-color);

font-size: 1.1rem;

margin-bottom: 1rem;

}

.activity-list {

display: flex;

flex-direction: column;

gap: 1rem;

}

.activity-item {

display: flex;

gap: 1rem;

padding: 0.75rem;

border-radius: var(--border-radius);

background-color: rgba(240, 240, 240, 0.5);

}

.activity-image img {

width: 120px;

height: 90px;

object-fit: cover;

border-radius: 4px;

}

.activity-details {

flex: 1;

}

.activity-details h4 {

font-size: 1rem;

margin-bottom: 0.25rem;

}

.activity-details p {

font-size: 0.9rem;

color: #555;

margin-bottom: 0.5rem;

}

.activity-time {

font-size: 0.8rem;

color: #777;

}

.activity-actions {

display: flex;

align-items: center;

}

.btn-notify {

background-color: var(--accent-color);

color: white;

border: none;

padding: 0.5rem 1rem;

border-radius: 4px;

cursor: pointer;

font-weight: 500;

transition: background-color 0.2s;

}

.btn-notify:hover {

background-color: #d62828;

}

/\* === NOTIFICATION OVERLAY === \*/

#notification-overlay {

position: fixed;

top: 0;

left: 0;

width: 100%;

height: 100%;

background-color: rgba(0, 0, 0, 0.7);

display: flex;

justify-content: center;

align-items: center;

z-index: 100;

}

.notification-container {

background-color: white;

border-radius: var(--border-radius);

width: 600px;

max-width: 90%;

box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);

overflow: hidden;

}

.notification-container h2 {

background-color: var(--accent-color);

color: white;

padding: 1rem;

text-align: center;

font-size: 1.5rem;

}

.notification-content {

padding: 1.5rem;

display: flex;

gap: 1.5rem;

}

.notification-image img {

width: 250px;

height: 200px;

object-fit: cover;

border-radius: 4px;

}

.notification-details {

flex: 1;

}

.notification-details h3 {

font-size: 1.3rem;

margin-bottom: 0.75rem;

color: var(--primary-color);

}

.notification-details p {

margin-bottom: 0.5rem;

line-height: 1.5;

}

.notification-actions {

display: flex;

padding: 1rem;

border-top: 1px solid #eee;

gap: 1rem;

justify-content: flex-end;

}

.notification-actions button {

padding: 0.75rem 1.25rem;

border: none;

border-radius: 4px;

cursor: pointer;

font-weight: 500;

}

#notification-send {

background-color: var(--accent-color);

color: white;

}

#notification-send:hover {

background-color: #d62828;

}

#notification-cancel {

background-color: #eee;

color: #666;

}

/\* === ALERT BOX === \*/

#alert-box {

position: fixed;

top: 50%;

left: 50%;

transform: translate(-50%, -50%);

background-color: #d62828;

color: white;

border-radius: var(--border-radius);

padding: 2rem;

box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);

text-align: center;

z-index: 200;

width: 400px;

max-width: 90%;

}

.alert-content h2 {

font-size: 1.8rem;

margin-bottom: 1rem;

}

.alert-content p {

margin-bottom: 1.5rem;

font-size: 1.1rem;

}

#alert-close {

background-color: white;

color: var(--accent-color);

border: none;

padding: 0.75rem 2rem;

border-radius: 4px;

cursor: pointer;

font-weight: 600;

font-size: 1rem;

}

/\* === UTILITY CLASSES === \*/

.hidden {

display: none !important;

}

/\* === RESPONSIVE ADJUSTMENTS === \*/

@media (max-width: 1200px) {

.stats-container {

grid-template-columns: repeat(2, 1fr);

}

}

@media (max-width: 1100px) {

.chart-grid {

grid-template-columns: repeat(2, 1fr);

}

}

@media (max-width: 700px) {

.chart-grid {

grid-template-columns: 1fr;

}

}

@media (max-width: 700px) {

.stats-container {

grid-template-columns: 1fr;

}

.activity-item {

flex-direction: column;

}

.notification-content {

flex-direction: column;

}

.notification-image img {

width: 100%;

height: auto;

}

}

JS code:

// DOM Elements

const currentDateTimeEl = document.getElementById('currentDateTime');

const notificationOverlay = document.getElementById('notification-overlay');

const notificationTitle = document.getElementById('notification-title');

const notificationLocation = document.getElementById('notification-location');

const notificationMessage = document.getElementById('notification-message');

const notificationImg = document.getElementById('notification-img');

const notificationSendBtn = document.getElementById('notification-send');

const notificationCancelBtn = document.getElementById('notification-cancel');

const alertBox = document.getElementById('alert-box');

const alertCloseBtn = document.getElementById('alert-close');

const notifyButtons = document.querySelectorAll('.btn-notify');

// Chart Objects

let hourlyChart, speciesChart, movementChart, regionChart;

// Update current date and time

function updateDateTime() {

const now = new Date();

const options = {

weekday: 'long',

year: 'numeric',

month: 'long',

day: 'numeric',

hour: '2-digit',

minute: '2-digit'

};

currentDateTimeEl.textContent = now.toLocaleDateString('en-US', options);

}

// Initialize Charts

function initCharts() {

// Hourly Detections Chart

const hourlyCtx = document.getElementById('hourlyDetectionsChart').getContext('2d');

hourlyChart = new Chart(hourlyCtx, {

type: 'line',

data: {

labels: ['6AM', '7AM', '8AM', '9AM', '10AM', '11AM', '12PM', '1PM', '2PM', '3PM', '4PM', '5PM'],

datasets: [{

label: 'Animal Detections',

data: [5, 8, 15, 22, 18, 12, 14, 16, 19, 23, 17, 10],

backgroundColor: 'rgba(69, 123, 157, 0.2)',

borderColor: 'rgba(69, 123, 157, 1)',

borderWidth: 2,

tension: 0.4,

pointBackgroundColor: 'rgba(69, 123, 157, 1)'

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

y: {

beginAtZero: true,

grid: {

color: 'rgba(0, 0, 0, 0.05)'

}

},

x: {

grid: {

display: false

}

}

},

plugins: {

legend: {

display: false

}

}

}

});

// Species Distribution Chart

const speciesCtx = document.getElementById('speciesDistributionChart').getContext('2d');

speciesChart = new Chart(speciesCtx, {

type: 'doughnut',

data: {

labels: ['Deer', 'Bear', 'Coyote', 'Moose', 'Fox', 'Other'],

datasets: [{

label: 'Species Distribution',

data: [42, 11, 17, 8, 14, 8],

backgroundColor: [

'#457b9d',

'#e63946',

'#f1c453',

'#2a9d8f',

'#f4a261',

'#a8dadc'

],

borderWidth: 1

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

plugins: {

legend: {

position: 'right',

labels: {

boxWidth: 8,

font: {

size: 8

}

}

}

}

}

});

// Movement Patterns Chart

const movementCtx = document.getElementById('movementPatternsChart').getContext('2d');

movementChart = new Chart(movementCtx, {

type: 'bar',

data: {

labels: ['North Trail', 'South Meadow', 'East Ridge', 'West Valley', 'Lake Area'],

datasets: [{

label: 'Morning',

data: [25, 18, 15, 12, 30],

backgroundColor: 'rgba(42, 157, 143, 0.7)'

}, {

label: 'Evening',

data: [15, 22, 28, 19, 17],

backgroundColor: 'rgba(233, 196, 106, 0.7)'

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

y: {

beginAtZero: true,

grid: {

color: 'rgba(0, 0, 0, 0.05)'

}

},

x: {

grid: {

display: false

}

}

},

plugins: {

legend: {

position: 'top',

labels: {

boxWidth: 8,

font: {

size: 8

}

}

}

}

}

});

// Region Detection Chart

const regionCtx = document.getElementById('regionDetectionChart').getContext('2d');

regionChart = new Chart(regionCtx, {

type: 'radar',

data: {

labels: ['Distance (m)', 'Speed (km/h)', 'Group Size', 'Duration (min)', 'Activity Level'],

datasets: [{

label: 'Herbivores',

data: [150, 25, 8, 45, 60],

backgroundColor: 'rgba(69, 123, 157, 0.2)',

borderColor: 'rgba(69, 123, 157, 0.8)',

borderWidth: 2,

pointBackgroundColor: 'rgba(69, 123, 157, 1)'

}, {

label: 'Predators',

data: [100, 40, 2, 20, 85],

backgroundColor: 'rgba(230, 57, 70, 0.2)',

borderColor: 'rgba(230, 57, 70, 0.8)',

borderWidth: 2,

pointBackgroundColor: 'rgba(230, 57, 70, 1)'

}]

},

options: {

responsive: true,

maintainAspectRatio: false,

scales: {

r: {

beginAtZero: true,

ticks: {

display: false

}

}

}

}

});

}

// Notification System

function showNotification(animal, location, imageUrl) {

notificationTitle.textContent = animal + ' Detected';

notificationLocation.textContent = 'Location: ' + location;

notificationMessage.textContent = 'Wildlife has been detected in the park. Would you like to notify visitors in the area?';

notificationImg.src = imageUrl;

notificationImg.alt = animal;

notificationOverlay.classList.remove('hidden');

}

function hideNotification() {

notificationOverlay.classList.add('hidden');

}

function showAlert() {

hideNotification();

alertBox.classList.remove('hidden');

// Automatically hide alert after 3 seconds

setTimeout(() => {

alertBox.classList.add('hidden');

}, 5000);

}

function hideAlert() {

alertBox.classList.add('hidden');

}

// Event Listeners

function setupEventListeners() {

// Notification buttons

notifyButtons.forEach(button => {

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

const animal = button.getAttribute('data-animal');

const location = button.getAttribute('data-location');

const imageUrl = button.getAttribute('data-image');

showNotification(animal, location, imageUrl);

});

});

// Notification actions

notificationSendBtn.addEventListener('click', showAlert);

notificationCancelBtn.addEventListener('click', hideNotification);

alertCloseBtn.addEventListener('click', hideAlert);

}

// Resize charts when window is resized

function handleResize() {

if (hourlyChart) hourlyChart.resize();

if (speciesChart) speciesChart.resize();

if (movementChart) movementChart.resize();

if (regionChart) regionChart.resize();

}

// Initialize dashboard

function initDashboard() {

updateDateTime();

setInterval(updateDateTime, 60000); // Update time every minute

initCharts();

setupEventListeners();

window.addEventListener('resize', handleResize);

}

// Start dashboard when page is loaded

document.addEventListener('DOMContentLoaded', initDashboard);