Dashboard.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Dashboard - Sound Attendance System</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.11.0/font/bootstrap-icons.css">

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

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container">

<a class="navbar-brand" href="/">Sound Attendance System</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav ms-auto">

<li class="nav-item">

<a class="nav-link" href="/">Home</a>

</li>

<li class="nav-item">

<a class="nav-link active" href="/dashboard">Dashboard</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/reports">Reports</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container mt-5">

<div class="row">

<div class="col-md-12">

<div class="card shadow">

<div class="card-header bg-primary text-white d-flex justify-content-between align-items-center">

<h3 class="mb-0">Attendance Dashboard</h3>

<div>

<span id="lastUpdated" class="badge bg-light text-dark me-2">Last updated: Never</span>

<button id="refreshButton" class="btn btn-sm btn-light" onclick="refreshAttendanceData()">

<i class="bi bi-arrow-clockwise"></i> Refresh

</button>

</div>

</div>

<div class="card-body">

<!-- Auto-refresh toggle -->

<div class="form-check form-switch mb-3">

<input class="form-check-input" type="checkbox" id="autoRefreshToggle" checked>

<label class="form-check-label" for="autoRefreshToggle">Auto-refresh data (every 30

seconds)</label>

</div>

<div class="row mb-4">

<div class="col-md-4">

<div class="card bg-primary text-white">

<div class="card-body">

<h5 class="card-title">Total Students</h5>

<h2 id="totalStudents">0</h2>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card bg-success text-white">

<div class="card-body">

<h5 class="card-title">Present Today</h5>

<h2 id="presentToday">0</h2>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card bg-danger text-white">

<div class="card-body">

<h5 class="card-title">Absent Today</h5>

<h2 id="absentToday">0</h2>

</div>

</div>

</div>

</div>

<div class="row">

<div class="col-md-12">

<div class="card">

<div class="card-header bg-light">

<div class="d-flex justify-content-between align-items-center">

<h5 class="mb-0">Attendance Records</h5>

<div>

<input type="date" id="attendanceDate" class="form-control" value="">

</div>

</div>

</div>

<div class="card-body">

<div class="table-responsive">

<table class="table table-striped">

<thead>

<tr>

<th>Student ID</th>

<th>Name</th>

<th>Status</th>

<th>Actions</th>

</tr>

</thead>

<tbody id="attendanceTable">

<!-- Attendance data will be loaded here -->

</tbody>

</table>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

<script>

let autoRefreshInterval;

const REFRESH\_INTERVAL = 30000; // 30 seconds

document.addEventListener('DOMContentLoaded', function () {

// Set today's date as default

const today = new Date();

const formattedDate = today.toISOString().split('T')[0];

document.getElementById('attendanceDate').value = formattedDate;

// Load attendance data for today

loadAttendanceData(formattedDate);

// Add event listener for date change

document.getElementById('attendanceDate').addEventListener('change', function () {

loadAttendanceData(this.value);

});

// Add event listener for auto-refresh toggle

document.getElementById('autoRefreshToggle').addEventListener('change', function () {

if (this.checked) {

startAutoRefresh();

} else {

stopAutoRefresh();

}

});

// Start auto-refresh by default

startAutoRefresh();

});

function startAutoRefresh() {

// Clear any existing interval

if (autoRefreshInterval) {

clearInterval(autoRefreshInterval);

}

// Set new interval

autoRefreshInterval = setInterval(refreshAttendanceData, REFRESH\_INTERVAL);

console.log("Auto-refresh started");

}

function stopAutoRefresh() {

if (autoRefreshInterval) {

clearInterval(autoRefreshInterval);

autoRefreshInterval = null;

console.log("Auto-refresh stopped");

}

}

function refreshAttendanceData() {

const date = document.getElementById('attendanceDate').value;

loadAttendanceData(date);

console.log("Attendance data refreshed");

}

function loadAttendanceData(date) {

fetch(`/get\_attendance?date=${date}`)

.then(response => response.json())

.then(data => {

if (data.success) {

updateDashboard(data.data);

// Update last refreshed time

const now = new Date();

const timeString = now.toLocaleTimeString();

document.getElementById('lastUpdated').textContent = `Last updated: ${timeString}`;

} else {

console.error('Error loading attendance data:', data.error);

}

})

.catch(error => console.error('Error:', error));

}

function updateDashboard(attendanceData) {

const tableBody = document.getElementById('attendanceTable');

tableBody.innerHTML = '';

let presentCount = 0;

let absentCount = 0;

attendanceData.forEach(student => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${student.STUDENT\_ID}</td>

<td>${student.STUDENT\_NAME}</td>

<td>

<span class="badge ${student.STATUS === 'Present' ? 'bg-success' : 'bg-danger'}">

${student.STATUS}

</span>

</td>

<td>

<button class="btn btn-sm ${student.STATUS === 'Present' ? 'btn-danger' : 'btn-success'}"

onclick="toggleAttendance('${student.STUDENT\_ID}', ${student.STATUS === 'Present' ? 'false' : 'true'})">

${student.STATUS === 'Present' ? 'Mark Absent' : 'Mark Present'}

</button>

</td>

`;

tableBody.appendChild(row);

if (student.STATUS === 'Present') {

presentCount++;

} else {

absentCount++;

}

});

document.getElementById('totalStudents').textContent = attendanceData.length;

document.getElementById('presentToday').textContent = presentCount;

document.getElementById('absentToday').textContent = absentCount;

}

function toggleAttendance(studentId, present) {

fetch('/mark\_attendance', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

student\_id: studentId,

present: present

})

})

.then(response => response.json())

.then(data => {

if (data.success) {

// Reload attendance data

const date = document.getElementById('attendanceDate').value;

loadAttendanceData(date);

} else {

console.error('Error updating attendance:', data.error);

}

})

.catch(error => console.error('Error:', error));

}

</script>

</body>

</html>

Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Sound Attendance System</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

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

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container">

<a class="navbar-brand" href="/">Sound Attendance System</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav ms-auto">

<li class="nav-item">

<a class="nav-link active" href="/">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/dashboard">Dashboard</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/reports">Reports</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container mt-5">

<div class="row justify-content-center">

<div class="col-md-8">

<div class="card shadow">

<div class="card-header bg-primary text-white">

<h3 class="mb-0">Sound-Based Attendance</h3>

</div>

<div class="card-body">

<div class="alert alert-info">

<p>This system marks attendance based on sound recognition. Each student has a unique sound identifier.</p>

<p>Make a sound to mark your attendance!</p>

<p>Please read these lines to mark your attendance</p>

</div>

<div class="text-center mb-4">

<div id="status" class="alert alert-secondary">Waiting for sound...</div>

</div>

<div class="model-container mb-4">

<!-- Teachable Machine Model will be loaded here -->

<div id="teachable-machine-container">

<!-- This is where the Teachable Machine model will be inserted -->

</div>

</div>

<div class="text-center">

<button id="startButton" class="btn btn-primary btn-lg">Start Listening</button>

<button id="stopButton" class="btn btn-danger btn-lg ms-2" disabled>Stop</button>

</div>

<div class="mt-4">

<h4>Today's Attendance</h4>

<div class="table-responsive">

<table class="table table-striped">

<thead>

<tr>

<th>Student ID</th>

<th>Name</th>

<th>Status</th>

<th>Time</th>

</tr>

</thead>

<tbody id="attendanceList">

<!-- Attendance records will be inserted here -->

</tbody>

</table>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

<!-- Teachable Machine Library -->

<!-- Teachable Machine Libraries -->

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs@1.3.1/dist/tf.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow-models/speech-commands@0.4.0/dist/speech-commands.min.js"></script>

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

</body>

</html>

Reports.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

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

<title>Reports - Sound Attendance System</title>

<link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

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

</head>

<body>

<nav class="navbar navbar-expand-lg navbar-dark bg-primary">

<div class="container">

<a class="navbar-brand" href="/">Sound Attendance System</a>

<button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarNav">

<span class="navbar-toggler-icon"></span>

</button>

<div class="collapse navbar-collapse" id="navbarNav">

<ul class="navbar-nav ms-auto">

<li class="nav-item">

<a class="nav-link" href="/">Home</a>

</li>

<li class="nav-item">

<a class="nav-link" href="/dashboard">Dashboard</a>

</li>

<li class="nav-item">

<a class="nav-link active" href="/reports">Reports</a>

</li>

</ul>

</div>

</div>

</nav>

<div class="container mt-5">

<div class="row">

<div class="col-md-12">

<div class="card shadow">

<div class="card-header bg-primary text-white">

<h3 class="mb-0">Attendance Reports</h3>

</div>

<div class="card-body">

<div class="row mb-4">

<div class="col-md-6">

<div class="card">

<div class="card-header bg-light">

<h5 class="mb-0">Student Report</h5>

</div>

<div class="card-body">

<div class="mb-3">

<label for="studentSelect" class="form-label">Select Student</label>

<select id="studentSelect" class="form-select">

<option value="">Select a student</option>

<!-- Students will be loaded here -->

</select>

</div>

<div class="row">

<div class="col-md-6">

<div class="mb-3">

<label for="startDate" class="form-label">Start Date</label>

<input type="date" id="startDate" class="form-control">

</div>

</div>

<div class="col-md-6">

<div class="mb-3">

<label for="endDate" class="form-label">End Date</label>

<input type="date" id="endDate" class="form-control">

</div>

</div>

</div>

<button id="generateStudentReport" class="btn btn-primary">Generate Report</button>

</div>

</div>

</div>

<div class="col-md-6">

<div class="card">

<div class="card-header bg-light">

<h5 class="mb-0">Date Range Report</h5>

</div>

<div class="card-body">

<div class="row">

<div class="col-md-6">

<div class="mb-3">

<label for="rangeDateStart" class="form-label">Start Date</label>

<input type="date" id="rangeDateStart" class="form-control">

</div>

</div>

<div class="col-md-6">

<div class="mb-3">

<label for="rangeDateEnd" class="form-label">End Date</label>

<input type="date" id="rangeDateEnd" class="form-control">

</div>

</div>

</div>

<button id="generateRangeReport" class="btn btn-primary">Generate Report</button>

</div>

</div>

</div>

</div>

<div class="card">

<div class="card-header bg-light">

<h5 class="mb-0">Report Results</h5>

</div>

<div class="card-body">

<div class="table-responsive">

<table class="table table-striped">

<thead id="reportTableHead">

<!-- Table header will be dynamically generated -->

</thead>

<tbody id="reportTableBody">

<!-- Report data will be loaded here -->

</tbody>

</table>

</div>

<div class="mt-3">

<button id="exportReport" class="btn btn-success" disabled>Export to CSV</button>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

</div>

<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"></script>

<script>

document.addEventListener('DOMContentLoaded', function() {

// Set default dates (current month)

const today = new Date();

const firstDay = new Date(today.getFullYear(), today.getMonth(), 1);

const lastDay = new Date(today.getFullYear(), today.getMonth() + 1, 0);

const formatDate = date => date.toISOString().split('T')[0];

document.getElementById('startDate').value = formatDate(firstDay);

document.getElementById('endDate').value = formatDate(today);

document.getElementById('rangeDateStart').value = formatDate(firstDay);

document.getElementById('rangeDateEnd').value = formatDate(today);

// Load students

loadStudents();

// Add event listeners

document.getElementById('generateStudentReport').addEventListener('click', generateStudentReport);

document.getElementById('generateRangeReport').addEventListener('click', generateRangeReport);

document.getElementById('exportReport').addEventListener('click', exportReportToCSV);

});

function loadStudents() {

fetch('/get\_students')

.then(response => response.json())

.then(data => {

if (data.success) {

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

data.data.forEach(student => {

const option = document.createElement('option');

option.value = student.STUDENT\_ID;

option.textContent = `${student.STUDENT\_NAME} (${student.STUDENT\_ID})`;

studentSelect.appendChild(option);

});

} else {

console.error('Error loading students:', data.error);

}

})

.catch(error => console.error('Error:', error));

}

function generateStudentReport() {

const studentId = document.getElementById('studentSelect').value;

const startDate = document.getElementById('startDate').value;

const endDate = document.getElementById('endDate').value;

if (!studentId) {

alert('Please select a student');

return;

}

if (!startDate || !endDate) {

alert('Please select both start and end dates');

return;

}

// This would be a real API call in a complete implementation

// For now, we'll simulate the data

const reportData = {

studentName: document.getElementById('studentSelect').options[document.getElementById('studentSelect').selectedIndex].text,

studentId: studentId,

startDate: startDate,

endDate: endDate,

attendanceRecords: [

{ date: '2023-10-01', status: 'Present' },

{ date: '2023-10-02', status: 'Present' },

{ date: '2023-10-03', status: 'Absent' },

{ date: '2023-10-04', status: 'Present' },

{ date: '2023-10-05', status: 'Present' }

]

};

displayStudentReport(reportData);

document.getElementById('exportReport').disabled = false;

}

function generateRangeReport() {

const startDate = document.getElementById('rangeDateStart').value;

const endDate = document.getElementById('rangeDateEnd').value;

if (!startDate || !endDate) {

alert('Please select both start and end dates');

return;

}

// This would be a real API call in a complete implementation

// For now, we'll simulate the data

const reportData = {

startDate: startDate,

endDate: endDate,

attendanceSummary: [

{ studentId: '1001', studentName: 'John Doe', presentDays: 18, absentDays: 2, percentage: 90 },

{ studentId: '1002', studentName: 'Jane Smith', presentDays: 20, absentDays: 0, percentage: 100 },

{ studentId: '1003', studentName: 'Bob Johnson', presentDays: 15, absentDays: 5, percentage: 75 }

]

};

displayRangeReport(reportData);

document.getElementById('exportReport').disabled = false;

}

function displayStudentReport(data) {

const tableHead = document.getElementById('reportTableHead');

const tableBody = document.getElementById('reportTableBody');

// Clear previous data

tableHead.innerHTML = '';

tableBody.innerHTML = '';

// Create header row

const headerRow = document.createElement('tr');

headerRow.innerHTML = `

<th>Date</th>

<th>Status</th>

`;

tableHead.appendChild(headerRow);

// Create data rows

data.attendanceRecords.forEach(record => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${record.date}</td>

<td>

<span class="badge ${record.status === 'Present' ? 'bg-success' : 'bg-danger'}">

${record.status}

</span>

</td>

`;

tableBody.appendChild(row);

});

// Add summary row

const presentCount = data.attendanceRecords.filter(r => r.status === 'Present').length;

const totalDays = data.attendanceRecords.length;

const percentage = (presentCount / totalDays \* 100).toFixed(2);

const summaryRow = document.createElement('tr');

summaryRow.className = 'table-info';

summaryRow.innerHTML = `

<td><strong>Summary</strong></td>

<td><strong>Present: ${presentCount}/${totalDays} (${percentage}%)</strong></td>

`;

tableBody.appendChild(summaryRow);

}

function displayRangeReport(data) {

const tableHead = document.getElementById('reportTableHead');

const tableBody = document.getElementById('reportTableBody');

// Clear previous data

tableHead.innerHTML = '';

tableBody.innerHTML = '';

// Create header row

const headerRow = document.createElement('tr');

headerRow.innerHTML = `

<th>Student ID</th>

<th>Name</th>

<th>Present Days</th>

<th>Absent Days</th>

<th>Attendance %</th>

`;

tableHead.appendChild(headerRow);

// Create data rows

data.attendanceSummary.forEach(record => {

const row = document.createElement('tr');

row.innerHTML = `

<td>${record.studentId}</td>

<td>${record.studentName}</td>

<td>${record.presentDays}</td>

<td>${record.absentDays}</td>

<td>

<div class="progress">

<div class="progress-bar ${getProgressBarClass(record.percentage)}"

role="progressbar"

style="width: ${record.percentage}%;"

aria-valuenow="${record.percentage}"

aria-valuemin="0"

aria-valuemax="100">

${record.percentage}%

</div>

</div>

</td>

`;

tableBody.appendChild(row);

});

}

function getProgressBarClass(percentage) {

if (percentage >= 90) return 'bg-success';

if (percentage >= 75) return 'bg-info';

if (percentage >= 60) return 'bg-warning';

return 'bg-danger';

}

function exportReportToCSV() {

// Get the table data

const table = document.querySelector('table');

let csv = [];

// Get header

const headerRow = table.querySelector('thead tr');

const headers = Array.from(headerRow.querySelectorAll('th')).map(th => th.textContent);

csv.push(headers.join(','));

// Get data rows

const rows = table.querySelectorAll('tbody tr');

rows.forEach(row => {

const rowData = Array.from(row.querySelectorAll('td')).map(cell => {

// Get text content without the progress bar

return cell.textContent.trim().replace(/,/g, ' ');

});

csv.push(rowData.join(','));

});

// Create and download CSV file

const csvContent = csv.join('\n');

const blob = new Blob([csvContent], { type: 'text/csv;charset=utf-8;' });

const url = URL.createObjectURL(blob);

const link = document.createElement('a');

link.setAttribute('href', url);

link.setAttribute('download', 'attendance\_report.csv');

link.style.visibility = 'hidden';

document.body.appendChild(link);

link.click();

document.body.removeChild(link);

}

</script>

</body>

</html>

[attendance.js](http://attendance.js)

// Global variables

let model, maxPredictions;

let isListening = false;

let attendanceRecords = {};

// DOM elements

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

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

const statusDisplay = document.getElementById('status');

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

// Check if elements exist

if (!startButton) console.error("Start button not found in the DOM");

if (!stopButton) console.error("Stop button not found in the DOM");

if (!statusDisplay) console.error("Status display not found in the DOM");

if (!attendanceList) console.error("Attendance list not found in the DOM");

// Event listeners

document.addEventListener('DOMContentLoaded', function () {

console.log("DOM fully loaded");

// Re-get elements in case they weren't available earlier

const startBtn = document.getElementById('startButton');

const stopBtn = document.getElementById('stopButton');

if (startBtn) {

console.log("Adding click listener to start button");

startBtn.addEventListener('click', startListening);

} else {

console.error("Start button not found after DOM loaded");

}

if (stopBtn) {

stopBtn.addEventListener('click', stopListening);

}

});

// Initialize the model

async function init() {

console.log("Initializing model...");

statusDisplay.textContent = "Loading model...";

statusDisplay.className = "alert alert-info";

if (typeof speechCommands === 'undefined') {

console.error('Speech Commands library not loaded');

statusDisplay.textContent = "Error: Speech Commands library not loaded. Please check your internet connection and refresh.";

statusDisplay.className = "alert alert-danger";

return;

}

// Fix the URLs - separate model.json and metadata.json

const baseURL = "https://teachablemachine.withgoogle.com/models/rVWQ-CEJ6/";

const modelURL = baseURL + "model.json";

const metadataURL = baseURL + "metadata.json";

console.log("Model URL:", modelURL);

console.log("Metadata URL:", metadataURL);

try {

console.log("Creating speech commands recognizer...");

// Load the model and metadata

const recognizer = speechCommands.create(

"BROWSER\_FFT", // fourier transform type

undefined, // speech commands vocabulary feature

modelURL,

metadataURL

);

console.log("Ensuring model is loaded...");

// Check if model is loaded

await recognizer.ensureModelLoaded();

console.log("Model loaded successfully!");

// Store the model in the global variable

model = recognizer;

// Get the total number of classes

maxPredictions = model.wordLabels().length;

console.log(`Model has ${maxPredictions} classes:`, model.wordLabels());

statusDisplay.textContent = "Model loaded. Ready to detect sounds.";

statusDisplay.className = "alert alert-success";

// Enable the start button

if (startButton) {

startButton.disabled = false;

}

} catch (error) {

console.error('Error loading model:', error);

statusDisplay.textContent = "Error loading model. Please refresh the page.";

statusDisplay.className = "alert alert-danger";

}

}

// Start listening for sounds

async function startListening() {

console.log("Start button clicked");

if (!model) {

console.log("Model not loaded, initializing...");

await init();

}

if (!model) {

console.error("Failed to initialize model");

return;

}

if (isListening) {

console.log("Already listening, ignoring click");

return;

}

isListening = true;

startButton.disabled = true;

stopButton.disabled = false;

statusDisplay.textContent = "Listening for sounds...";

statusDisplay.className = "alert alert-primary";

console.log("Starting audio classification...");

// Start the audio classification loop

classifyAudio();

}

// Stop listening for sounds

function stopListening() {

console.log("Stop button clicked");

isListening = false;

startButton.disabled = false;

stopButton.disabled = true;

statusDisplay.textContent = "Stopped listening.";

statusDisplay.className = "alert alert-secondary";

if (model) {

console.log("Stopping model listening");

model.stopListening();

}

// Refresh dashboard after stopping

refreshDashboard();

}

// Continuously classify audio

async function classifyAudio() {

if (!isListening) return;

try {

console.log("Starting to listen for audio...");

// Get the prediction from the model

await model.listen(result => {

// Get the top prediction

const scores = result.scores;

// Find the class with the highest score

let maxScore = 0;

let maxScoreIndex = -1;

for (let i = 0; i < scores.length; i++) {

if (scores[i] > maxScore) {

maxScore = scores[i];

maxScoreIndex = i;

}

}

// Get the label for the top prediction

const topPredictionLabel = model.wordLabels()[maxScoreIndex];

console.log(`Detected: ${topPredictionLabel} (${maxScore.toFixed(2)})`);

// Check if the confidence is high enough

if (maxScore > 0.8) {

const soundLabel = topPredictionLabel;

// If this is a student sound (not background noise)

if (soundLabel !== "Background Noise") {

statusDisplay.textContent = `Detected: ${soundLabel}`;

statusDisplay.className = "alert alert-success";

// Mark attendance for this student

markAttendance(soundLabel);

}

}

}, {

includeSpectrogram: false,

probabilityThreshold: 0.75,

invokeCallbackOnNoiseAndUnknown: false,

overlapFactor: 0.5

});

// Continue the classification loop

if (!isListening) {

model.stopListening();

}

} catch (error) {

console.error('Error during audio classification:', error);

statusDisplay.textContent = "Error detecting sound.";

statusDisplay.className = "alert alert-danger";

// Try to restart after a delay

if (isListening) {

setTimeout(classifyAudio, 2000);

}

}

}

// Mark attendance for a student

async function markAttendance(soundLabel) {

// Check if we've already marked this student recently

if (attendanceRecords[soundLabel] &&

(Date.now() - attendanceRecords[soundLabel].timestamp) < 60000) {

// Don't mark attendance again within 1 minute

return;

}

try {

// This would be a call to your backend in a real implementation

// For now, we'll simulate it

const studentId = soundLabel; // In a real system, you'd map the sound label to a student ID

// Record the attendance locally

const now = new Date();

attendanceRecords[soundLabel] = {

timestamp: Date.now(),

time: now.toLocaleTimeString()

};

// Add to the attendance list in the UI

addAttendanceRecord(studentId, now.toLocaleTimeString());

// Send to the server

await sendAttendanceToServer(studentId);

} catch (error) {

console.error('Error marking attendance:', error);

}

}

// Add an attendance record to the UI

function addAttendanceRecord(studentId, time) {

const row = document.createElement('tr');

row.innerHTML = `

<td>${studentId}</td>

<td>Student Name</td>

<td><span class="badge bg-success">Present</span></td>

<td>${time}</td>

`;

if (attendanceList) {

// Add at the top of the list

if (attendanceList.firstChild) {

attendanceList.insertBefore(row, attendanceList.firstChild);

} else {

attendanceList.appendChild(row);

}

}

}

// Send attendance data to the server

async function sendAttendanceToServer(studentId) {

try {

const response = await fetch('/mark\_attendance', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({

student\_id: studentId,

present: true

})

});

const data = await response.json();

if (!data.success) {

console.error('Server error marking attendance:', data.error);

}

} catch (error) {

console.error('Error sending attendance to server:', error);

}

}

// Refresh the dashboard page with latest attendance data

function refreshDashboard() {

console.log("Refreshing dashboard data");

// If we're already on the dashboard page

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

// If there's a loadAttendanceData function available (from dashboard.html)

if (typeof loadAttendanceData === 'function') {

// Get today's date in YYYY-MM-DD format

const today = new Date().toISOString().split('T')[0];

loadAttendanceData(today);

}

} else {

// Show a notification that attendance was updated

const notification = document.createElement('div');

notification.className = 'alert alert-info mt-3 d-flex justify-content-between align-items-center';

const messageContainer = document.createElement('div');

messageContainer.textContent = 'Attendance recorded! View details on the Dashboard.';

notification.appendChild(messageContainer);

const buttonContainer = document.createElement('div');

// Add a link to the dashboard

const link = document.createElement('a');

link.href = '/dashboard';

link.className = 'btn btn-primary btn-sm me-2';

link.textContent = 'Go to Dashboard';

buttonContainer.appendChild(link);

// Add a view here button

const viewHereBtn = document.createElement('button');

viewHereBtn.className = 'btn btn-outline-primary btn-sm';

viewHereBtn.textContent = 'View Today\'s Report';

viewHereBtn.onclick = function () {

// Request today's attendance summary from the server and display it

showAttendanceSummary();

};

buttonContainer.appendChild(viewHereBtn);

notification.appendChild(buttonContainer);

// Insert the notification before the attendance list

if (attendanceList && attendanceList.parentNode) {

// Look for existing notification and remove it

const existingNotification = document.querySelector('.alert.alert-info.mt-3');

if (existingNotification) {

existingNotification.remove();

}

attendanceList.parentNode.insertBefore(notification, attendanceList);

// Don't remove the notification automatically to allow user to click the buttons

}

}

}

// Show attendance summary from the server

async function showAttendanceSummary() {

statusDisplay.textContent = "Loading attendance summary...";

statusDisplay.className = "alert alert-info";

try {

// Get today's date in YYYY-MM-DD format

const today = new Date().toISOString().split('T')[0];

// Fetch attendance data from server

const response = await fetch(`/get\_attendance?date=${today}`);

const data = await response.json();

if (data.success) {

// Display summary in a modal or panel

const presentStudents = data.data.filter(student => student.STATUS === 'Present').length;

const totalStudents = data.data.length;

const absentStudents = totalStudents - presentStudents;

const attendanceRate = ((presentStudents / totalStudents) \* 100).toFixed(1);

// Create or update summary panel

let summaryPanel = document.getElementById('attendanceSummaryPanel');

if (!summaryPanel) {

summaryPanel = document.createElement('div');

summaryPanel.id = 'attendanceSummaryPanel';

summaryPanel.className = 'card mt-4 shadow-sm';

document.querySelector('.card-body').appendChild(summaryPanel);

}

summaryPanel.innerHTML = `

<div class="card-header bg-info text-white d-flex justify-content-between align-items-center">

<h5 class="mb-0">Today's Attendance Summary</h5>

<button type="button" class="btn-close btn-close-white" aria-label="Close" onclick="this.parentElement.parentElement.remove()"></button>

</div>

<div class="card-body">

<div class="row">

<div class="col-md-4">

<div class="card bg-primary text-white">

<div class="card-body text-center">

<h6>Total Students</h6>

<h3>${totalStudents}</h3>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card bg-success text-white">

<div class="card-body text-center">

<h6>Present</h6>

<h3>${presentStudents}</h3>

</div>

</div>

</div>

<div class="col-md-4">

<div class="card bg-danger text-white">

<div class="card-body text-center">

<h6>Absent</h6>

<h3>${absentStudents}</h3>

</div>

</div>

</div>

</div>

<div class="progress mt-3" style="height: 25px;">

<div class="progress-bar bg-success" role="progressbar" style="width: ${attendanceRate}%;"

aria-valuenow="${attendanceRate}" aria-valuemin="0" aria-valuemax="100">

${attendanceRate}% Present

</div>

</div>

</div>

`;

statusDisplay.textContent = "Attendance summary loaded!";

statusDisplay.className = "alert alert-success";

// Scroll to the summary panel

summaryPanel.scrollIntoView({ behavior: 'smooth' });

} else {

statusDisplay.textContent = "Error loading attendance summary: " + data.error;

statusDisplay.className = "alert alert-danger";

}

} catch (error) {

console.error('Error loading attendance summary:', error);

statusDisplay.textContent = "Error loading attendance summary. Please try again.";

statusDisplay.className = "alert alert-danger";

}

}

// Initialize the page

window.onload = function () {

// We'll initialize the model when the user clicks the start button

if (startButton) {

startButton.disabled = false;

}

};

[app.py](http://app.py)

from flask import Flask, render\_template, request, jsonify

import oracledb

from flask\_cors import CORS

import os

import json

import traceback

from datetime import datetime

from database.db\_config import get\_connection

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

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/dashboard')

def dashboard():

return render\_template('dashboard.html')

@app.route('/reports')

def reports():

return render\_template('reports.html')

@app.route('/mark\_attendance', methods=['POST'])

def mark\_attendance():

data = request.json

student\_identifier = data.get('student\_id') # This could now be a name, sound label, or ID

present = data.get('present', True)

print(f"Received attendance request for student: {student\_identifier}, present: {present}")

if not student\_identifier:

return jsonify({"success": False, "error": "Student identifier is required"}), 400

try:

# Connect to database

try:

conn = get\_connection()

print("Database connection successful")

except Exception as e:

error\_msg = f"Database connection error: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

cursor = conn.cursor()

# Get current date

today = datetime.now().strftime('%Y-%m-%d')

print(f"Marking attendance for date: {today}")

# First check if student exists - now checking by name, ID, or sound label

try:

print(f"Looking up student by ID, name, or sound label: {student\_identifier}")

cursor.execute(

"""

SELECT STUDENT\_ID, STUDENT\_NAME

FROM STUDENTS

WHERE STUDENT\_ID = :1

OR UPPER(STUDENT\_NAME) = UPPER(:2)

OR UPPER(SOUND\_LABEL) = UPPER(:3)

""",

[student\_identifier, student\_identifier, student\_identifier]

)

student\_record = cursor.fetchone()

if not student\_record:

error\_msg = f"Student not found with identifier: {student\_identifier}"

print(error\_msg)

# Let's add this student automatically

print(f"Adding new student with name: {student\_identifier}")

try:

# Generate a student ID

cursor.execute("SELECT COUNT(\*) FROM STUDENTS")

count = cursor.fetchone()[0]

new\_student\_id = f"S{count+1:03d}"

# Insert the new student

cursor.execute(

"INSERT INTO STUDENTS (STUDENT\_ID, STUDENT\_NAME, SOUND\_LABEL) VALUES (:1, :2, :3)",

[new\_student\_id, student\_identifier, student\_identifier]

)

conn.commit()

print(f"Added new student with ID: {new\_student\_id}, Name: {student\_identifier}")

# Use this new student ID

actual\_student\_id = new\_student\_id

except Exception as e:

error\_msg = f"Error adding new student: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

else:

actual\_student\_id = student\_record[0]

print(f"Found student with ID: {actual\_student\_id}, Name: {student\_record[1]}")

except Exception as e:

error\_msg = f"Error checking student: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

# Check if attendance already marked for today

try:

print(f"Checking if attendance already exists for today")

cursor.execute(

"SELECT \* FROM ATTENDANCE WHERE STUDENT\_ID = :1 AND ATTENDANCE\_DATE = TO\_DATE(:2, 'YYYY-MM-DD')",

[actual\_student\_id, today]

)

existing = cursor.fetchone()

if existing:

# Update existing record

print(f"Updating existing attendance record")

cursor.execute(

"UPDATE ATTENDANCE SET PRESENT = :1 WHERE STUDENT\_ID = :2 AND ATTENDANCE\_DATE = TO\_DATE(:3, 'YYYY-MM-DD')",

[1 if present else 0, actual\_student\_id, today]

)

else:

# Insert new record

print(f"Creating new attendance record")

cursor.execute(

"INSERT INTO ATTENDANCE (STUDENT\_ID, ATTENDANCE\_DATE, PRESENT) VALUES (:1, TO\_DATE(:2, 'YYYY-MM-DD'), :3)",

[actual\_student\_id, today, 1 if present else 0]

)

conn.commit()

print("Attendance marked successfully")

except Exception as e:

error\_msg = f"Error marking attendance: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

finally:

cursor.close()

conn.close()

return jsonify({"success": True, "message": "Attendance marked successfully"})

except Exception as e:

error\_msg = f"Unexpected error: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

@app.route('/get\_attendance', methods=['GET'])

def get\_attendance():

attendance\_date = request.args.get('date', datetime.now().strftime('%Y-%m-%d'))

try:

try:

conn = get\_connection()

print(f"Database connection successful for get\_attendance on date {attendance\_date}")

except Exception as e:

error\_msg = f"Database connection error: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

cursor = conn.cursor()

try:

# FIXED: Changed the bind variable name from :date to :attendance\_date

# and changed the binding method to use a dictionary instead of a list

cursor.execute(

"""

SELECT s.STUDENT\_ID, s.STUDENT\_NAME,

CASE WHEN a.PRESENT = 1 THEN 'Present' ELSE 'Absent' END as STATUS

FROM STUDENTS s

LEFT JOIN ATTENDANCE a ON s.STUDENT\_ID = a.STUDENT\_ID AND a.ATTENDANCE\_DATE = TO\_DATE(:attendance\_date, 'YYYY-MM-DD')

ORDER BY s.STUDENT\_NAME

""",

{"attendance\_date": attendance\_date} # Using dictionary with non-reserved keyword

)

columns = [col[0] for col in cursor.description]

attendance\_data = [dict(zip(columns, row)) for row in cursor.fetchall()]

print(f"Retrieved {len(attendance\_data)} attendance records")

except Exception as e:

error\_msg = f"Error retrieving attendance data: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

finally:

cursor.close()

conn.close()

return jsonify({"success": True, "data": attendance\_data})

except Exception as e:

error\_msg = f"Unexpected error in get\_attendance: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

@app.route('/get\_students', methods=['GET'])

def get\_students():

try:

try:

conn = get\_connection()

print("Database connection successful for get\_students")

except Exception as e:

error\_msg = f"Database connection error: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

cursor = conn.cursor()

try:

cursor.execute("SELECT STUDENT\_ID, STUDENT\_NAME, SOUND\_LABEL FROM STUDENTS ORDER BY STUDENT\_NAME")

columns = [col[0] for col in cursor.description]

students = [dict(zip(columns, row)) for row in cursor.fetchall()]

print(f"Retrieved {len(students)} students")

except Exception as e:

error\_msg = f"Error retrieving students: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

finally:

cursor.close()

conn.close()

return jsonify({"success": True, "data": students})

except Exception as e:

error\_msg = f"Unexpected error in get\_students: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

@app.route('/test\_db', methods=['GET'])

def test\_db():

"""Simple route to test database connection"""

try:

conn = get\_connection()

cursor = conn.cursor()

# Test query

cursor.execute("SELECT COUNT(\*) FROM STUDENTS")

count = cursor.fetchone()[0]

# Get table names

cursor.execute("SELECT table\_name FROM user\_tables")

tables = [row[0] for row in cursor.fetchall()]

cursor.close()

conn.close()

return jsonify({

"success": True,

"message": "Database connection successful",

"student\_count": count,

"tables": tables

})

except Exception as e:

error\_msg = f"Database test error: {str(e)}"

print(error\_msg)

print(traceback.format\_exc())

return jsonify({"success": False, "error": error\_msg}), 500

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

app.run(debug=True)

Requirements.txt

Flask==2.0.3

Werkzeug==2.0.3

oracledb==1.3.1

python-dotenv==1.0.0

gunicorn==20.1.0