Index.html

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8" />

<title>Smart Battery Health Monitoring and Maintenance System</title>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script src="https://cdn.jsdelivr.net/npm/@tensorflow/tfjs"></script>

<script src="https://cdn.jsdelivr.net/npm/raphael@2.3.0/raphael.min.js"></script>

<script src="https://cdn.jsdelivr.net/npm/justgage@1.4.0/justgage.min.js"></script>

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

</head>

<body>

<div class="mode">

<div class="toggle-switch">

<label class="switch-label">

<input type="checkbox" class="checkbox" onclick="toggleDarkMode()" />

<span class="slider"></span>

</label>

</div>

</div>

<div class="container">

<div class="top">

<h1>Battery Dashboard</h1>

</div>

<div class="dashboard">

<div class="card">

<h2>🔌Current</h2>

<span id="current">-- A</span>

</div>

<div class="card">

<h2>⚡Voltage</h2>

<span id="voltage">-- V</span>

</div>

<div class="card">

<h2>🌡️Temperature</h2>

<span id="temp">-- °C</span>

</div>

<div class="card">

<h2>🪫Power</h2>

<span id="power">-- W</span>

</div>

<div class="card">

<h2>🏎️Efficiency</h2>

<span id="efficiency">-- %</span>

</div>

<div class="card">

<h2>⏳Remaining Useful Life</h2>

<span id="rul">-- cycles</span>

</div>

<div class="card">

<h2>🕰️Remaining Life</h2>

<span id="hoursRemaining">-- hours</span>

</div>

<div class="card">

<h2>Battery Status</h2>

<span id="batteryStatus">--</span>

</div>

</div>

<div class="meter">

<div class="sepm">

<h2>📈State of Charge</h2>

<div id="socGauge"></div>

<span id="soc">100 %</span>

</div>

<div class="sepm">

<h2>❤️State of Health</h2>

<div id="sohGauge"></div>

<span id="soh">100</span>

</div>

</div>

<div class="charts">

<canvas id="currentChart" height="300px" width="400px"></canvas>

<canvas id="voltageChart" height="300px" width="400px"></canvas>

<canvas id="tempChart" height="300px" width="400px"></canvas>

<canvas id="powerChart" height="300px" width="400px"></canvas>

</div>

</div>

<script>

const currentSpan = document.getElementById("current");

const voltageSpan = document.getElementById("voltage");

const tempSpan = document.getElementById("temp");

function toggleDarkMode() {

document.body.classList.toggle("dark");

}

window.serialAPI.listPorts().then((ports) => {

if (ports.length > 0) {

window.serialAPI.startSerial(ports[0]);

}

});

const maxDataPoints = 20;

const createLineChart = (ctx, label, borderColor) => {

return new Chart(ctx, {

type: "line",

data: {

labels: [],

datasets: [

{

label: label,

data: [],

borderColor: borderColor,

fill: false,

tension: 0.3,

},

],

},

options: {

responsive: true,

animation: false,

scales: {

x: { display: false },

y: { beginAtZero: true },

},

},

});

};

const currentChart = createLineChart(

document.getElementById("currentChart"),

"Current (A)",

"#f44336"

);

const voltageChart = createLineChart(

document.getElementById("voltageChart"),

"Voltage (V)",

"#2196f3"

);

const tempChart = createLineChart(

document.getElementById("tempChart"),

"Temperature (°C)",

"#4caf50"

);

const powerChart = createLineChart(

document.getElementById("powerChart"),

"Power (W)",

"#ff9800"

);

const updateChart = (chart, value) => {

const now = new Date().toLocaleTimeString();

chart.data.labels.push(now);

chart.data.datasets[0].data.push(parseFloat(value));

if (chart.data.labels.length > maxDataPoints) {

chart.data.labels.shift();

chart.data.datasets[0].data.shift();

}

chart.update();

};

let model;

async function trainAndSaveModel() {

const trainingData = [

{ current: 0.144, voltage: 1.5, rul: 500 },

{ current: 0.144, voltage: 1.48, rul: 490 },

{ current: 0.144, voltage: 1.46, rul: 480 },

{ current: 0.144, voltage: 1.44, rul: 470 },

{ current: 0.144, voltage: 1.42, rul: 460 },

{ current: 0.144, voltage: 1.4, rul: 450 },

{ current: 0.144, voltage: 1.38, rul: 440 },

{ current: 0.144, voltage: 1.36, rul: 430 },

{ current: 0.144, voltage: 1.34, rul: 420 },

{ current: 0.144, voltage: 1.32, rul: 410 },

{ current: 0.144, voltage: 1.3, rul: 400 },

{ current: 0.144, voltage: 1.28, rul: 390 },

{ current: 0.144, voltage: 1.26, rul: 380 },

{ current: 0.144, voltage: 1.24, rul: 370 },

{ current: 0.144, voltage: 1.22, rul: 360 },

{ current: 0.144, voltage: 1.2, rul: 350 },

{ current: 0.144, voltage: 1.18, rul: 340 },

{ current: 0.144, voltage: 1.16, rul: 330 },

{ current: 0.144, voltage: 1.14, rul: 320 },

{ current: 0.144, voltage: 1.12, rul: 310 },

{ current: 0.144, voltage: 1.1, rul: 300 },

{ current: 0.144, voltage: 1.08, rul: 290 },

{ current: 0.144, voltage: 1.06, rul: 280 },

{ current: 0.144, voltage: 1.04, rul: 270 },

{ current: 0.144, voltage: 1.02, rul: 260 },

{ current: 0.144, voltage: 1.0, rul: 250 },

{ current: 0.144, voltage: 0.98, rul: 240 },

{ current: 0.144, voltage: 0.96, rul: 230 },

{ current: 0.144, voltage: 0.94, rul: 220 },

{ current: 0.144, voltage: 0.92, rul: 210 },

{ current: 0.144, voltage: 0.9, rul: 200 },

];

const inputs = trainingData.map((d) => [d.current, d.voltage]);

const labels = trainingData.map((d) => d.rul);

model = tf.sequential();

model.add(

tf.layers.dense({ units: 32, activation: "relu", inputShape: [2] })

);

model.add(tf.layers.dense({ units: 16, activation: "relu" }));

model.add(tf.layers.dense({ units: 1, activation: "linear" }));

model.compile({

optimizer: "adam",

loss: "meanSquaredError",

});

const xs = tf.tensor2d(inputs);

const ys = tf.tensor2d(labels, [labels.length, 1]);

await model.fit(xs, ys, {

epochs: 300,

batchSize: 10,

callbacks: {

onEpochEnd: (epoch, logs) => {},

},

});

await model.save("localstorage://battery-life-model");

console.log("✅ Model trained and saved to localStorage.");

}

// trainAndSaveModel();

async function loadModel() {

try {

model = await tf.loadLayersModel("localstorage://battery-life-model");

console.log("Model loaded from localStorage.");

} catch (error) {

console.log("No saved model found");

}

}

loadModel();

async function predictRUL(current, voltage, temperature) {

if (!model) {

console.log("Model not loaded yet.");

return;

}

if (voltage <= 0 || current < 0) {

document.getElementById("rul").textContent = `0 cycles`;

return;

}

const input = tf.tensor2d([[current, voltage]]);

const prediction = model.predict(input);

const predictedRUL = prediction.dataSync()[0];

document.getElementById("rul").textContent = `${predictedRUL.toFixed(

2

)} cycles`;

const cycleDurationHours = 2.29;

const predictedHours = predictedRUL \* cycleDurationHours;

document.getElementById(

"hoursRemaining"

).textContent = `${predictedHours.toFixed(2)} hours remaining`;

let status = "";

if (predictRUL >450){

status = "Good 🔋";

}if(predictRUL >400){

status = "Starts To Drain ⚡";

}else if (predictedRUL > 300) {

status = "Draining 📉";

} else if (predictedRUL > 100) {

status = "Warning ⚠️";

} else {

status = "Critical ❌";

}

document.getElementById("batteryStatus").textContent = status;

}

window.serialAPI.onData((data) => {

try {

const match = data.match(

/Current: (.\*?) A \| Voltage: (.\*?) V \| Temp: (.\*?) °C/

);

if (match) {

const [\_, current, voltage, temp] = match;

currentSpan.textContent = `${current} A`;

voltageSpan.textContent = `${voltage} V`;

tempSpan.textContent = `${temp} °C`;

updateChart(currentChart, current);

updateChart(voltageChart, voltage);

updateChart(tempChart, temp);

const resistance = 10;

const power = parseFloat(voltage) \*\* 2 / resistance;

document.getElementById("power").textContent = `${power.toFixed(

2

)} W`;

updateChart(powerChart, power);

const ratedVoltage = 1.6;

const idealInputPower = ratedVoltage \* current;

const efficiency =

idealInputPower > 0 ? (power / idealInputPower) \* 100 : 0;

document.getElementById(

"efficiency"

).textContent = `${efficiency.toFixed(2)} %`;

estimateSoCFromVoltage(voltage);

estimateSoHFromVoltage(voltage);

const currentVal = parseFloat(current);

const voltageVal = parseFloat(voltage);

const tempVal = parseFloat(temp);

predictRUL(currentVal, voltageVal);

}

} catch (e) {

console.log("Data parse error", e);

}

});

function estimateSoCFromVoltage(voltage) {

voltage = parseFloat(voltage);

const minV = 1.1;

const maxV = 1.55;

let soc = ((voltage - minV) / (maxV - minV)) \* 100;

soc = Math.min(100, Math.max(0, soc.toFixed(1)));

document.getElementById("soc").textContent = `${soc} %`;

updateSOCGauge(soc);

}

function estimateSoHFromVoltage(voltage) {

voltage = parseFloat(voltage);

const ratedVoltage = 1.55;

const minVoltage = 1.1;

let soh = ((voltage - minVoltage) / (ratedVoltage - minVoltage)) \* 100;

soh = Math.min(100, Math.max(0, soh.toFixed(1)));

document.getElementById("soh").textContent = `${soh}`;

updateSOHGauge(soh);

}

let socGauge = new JustGage({

id: "socGauge",

value: 0,

min: 0,

max: 100,

title: "State of Charge",

hideValue: true,

levelColors: ["#ff4d4d", "#ffcc00", "#66ff66"]

});

function updateSOCGauge(value) {

socGauge.refresh(value);

}

let sohGauge = new JustGage({

id: "sohGauge",

value: 0,

min: 0,

max: 100,

title: "State of Health",

hideValue: true,

levelColors: ["#ff4d4d", "#ffcc00", "#66ff66"]

});

function updateSOHGauge(value) {

sohGauge.refresh(value);

}

</script>

</body>

</html>

Main.js

const { app, BrowserWindow, ipcMain } = require('electron');

const path = require('path');

const { SerialPort } = require('serialport');

const { ReadlineParser } = require('@serialport/parser-readline');

let win;

let serialPort;

function createWindow() {

win = new BrowserWindow({

width: 1280,

height: 720,

webPreferences: {

preload: path.join(\_\_dirname, 'preload.js')

},

});

win.maximize();

win.setMenuBarVisibility(false);

win.loadFile('index.html');

}

app.whenReady().then(createWindow);

ipcMain.on('start-serial', (event, portName) => {

if (serialPort && serialPort.isOpen) {

serialPort.removeAllListeners();

serialPort.close(() => {

console.log('Previous port closed');

openSerial(portName, event);

});

} else {

openSerial(portName, event);

}

});

function openSerial(portName, event) {

serialPort = new SerialPort({

path: portName,

baudRate: 9600,

});

const parser = serialPort.pipe(new ReadlineParser({ delimiter: '\n' }));

serialPort.on('open', () => {

console.log('Serial port opened:', portName);

});

parser.on('data', (line) => {

event.sender.send('serial-data',line.trim());

});

serialPort.on('error', (err) => {

console.error('Serial port error:', err.message);

});

}

ipcMain.handle('list-ports', async () => {

const ports = await SerialPort.list();

return ports.map(p => p.path);

});

const { contextBridge, ipcRenderer } = require('electron');

Preload.js

contextBridge.exposeInMainWorld('serialAPI', {

startSerial: (port) => ipcRenderer.send('start-serial', port),

onData: (callback) => ipcRenderer.on('serial-data', (\_, data) => callback(data)),

listPorts: () => ipcRenderer.invoke('list-ports')

});