IQRA UNIVERSITY IU

INTERNET OF THINGS

THERMO-DRIVE: TEMPERATURE ALERT SYSTEM & VEHICLE DETECTION SYSTEM

CCP Project Report

Group Members:

Ahmed Raza (IU02-0322-0374)
Muhammad Uzair (IU02-0322-0024)

Submitted to: Dr. Arshee Ahmed
Semester: Fall – 2026

ABSTRACT

The Thermo-Drive project integrates a Temperature Alert System and a Vehicle Detection System to provide a simulated smart vehicle monitoring platform. Using simulated Arduino sensors, the system continuously monitors temperature and vehicle proximity, generating real-time alerts whenever conditions exceed predefined safety thresholds. The system emphasizes complex problem-solving**,** addressing multiple conflicting requirements such as maintaining optimal environmental conditions and ensuring safe vehicle distances simultaneously. By logging sensor data and visualizing it through interactive dashboards, the project demonstrates in-depth knowledge of sensor data analysis, threshold handling, and real-time system monitoring. Both manual and automatic modes are supported, allowing the user to fetch individual readings or enable continuous monitoring. The system logs all data to a CSV file for historical analysis, which can be used for trend evaluation or risk assessment. Overall, Thermo-Drive provides a practical implementation of real-time monitoring systems in smart vehicles, showcasing WP1 (depth of knowledge), WP2 (handling conflicting requirements), and WP3 (depth of analysis).

Table of Content

1. Acknowledgement

This project was made possible through the dedicated efforts of a talented team of students from **Iqra University**, whose creativity, hard work, and commitment were instrumental in its success. The team comprised developers, designers, and project managers, all working together to achieve a common goal of creating a functional and practical smart vehicle monitoring system. Special recognition and gratitude go to **Dr. Arshee Ahmed**, whose guidance, mentorship, and encouragement played a pivotal role throughout the development process. Her expertise, technical insights, and constructive feedback helped shape the direction of this project, ensuring it adhered to modern technological standards and industry best practices.

We also wish to express our appreciation to Iqra University for providing a supportive environment, access to resources, and facilities necessary for undertaking this project. The contributions and trust from all stakeholders, including faculty members, have been critical in turning this vision into reality. The successful completion of the **Thermo-Drive Temperature and Vehicle Monitoring System** reflects the collaborative efforts, technical expertise, and problem-solving abilities of everyone involved. This project stands as a testament to the team's ability to tackle real-world challenges and create practical, user-centric solutions that integrate simulation, real-time monitoring, and interactive visualization.

2. Overview of Project

The Thermo-Drive project is designed to simulate the monitoring of both temperature and vehicle distance in smart vehicles. It combines real-time sensor simulation**,** web-based visualization, and automated alerting mechanisms. The temperature monitoring system simulates an Arduino sensor that captures environmental temperature and triggers alerts such as TOO COLD, NORMAL, WARNING and ALERT based on threshold values. The vehicle detection system simulates ultrasonic sensors to measure distances of nearby vehicles and generate status notifications such as SAFE, WARNING and ALERT**.**

This system demonstrates the integration of frontend and backend technologies, using Flask for the server, JavaScript for dynamic updates, and Plotly for graph visualization. It is capable of continuous monitoring, historical data logging, and visual alerting, making it a comprehensive tool for studying sensor-based monitoring in smart vehicles.

3. Purpose

The main purpose of this project is to design and implement a system that can:

- Monitor environmental temperature and vehicle proximity in real-time.
- Generate alerts for unsafe conditions to ensure operational safety.
- Record and log sensor data for future analysis and visualization.
- Provide an interactive and user-friendly interface for both manual and automatic monitoring.

By achieving these goals, the system demonstrates practical applications of smart vehicle safety mechanisms, real-time data analysis, and automated alerting.

4. Scope

The scope of the Thermo-Drive project includes:

- Simulation of temperature and vehicle distance sensors using randomized data.
- Web-based monitoring interface that allows user interaction.
- Generation of real-time alerts for unsafe conditions.
- Historical data logging to CSV files.
- Graphical visualization of both temperature trends and vehicle distances.
- Handling of multiple conflicting thresholds to ensure safety.

This project does not include actual hardware implementation; however, it provides a framework that can be easily integrated with real Arduino sensors in future extensions.

5. Introduction

Modern smart vehicles require advanced monitoring systems for both environmental and operational safety. Temperature extremes can affect vehicle performance, and vehicle proximity monitoring is essential to prevent collisions. The Thermo-Drive system simulates both of these monitoring tasks using virtual sensors and a web-based interface.

The system leverages Flask APIs to manage backend data processing and JavaScript to update frontend visualizations in real-time. Alerts are color-coded for immediate recognition: blue for TOO COLD, green for NORMAL/SAFE, orange for WARNING, and red for ALERT. By combining data collection, automated evaluation, and visualization, the system offers a complete workflow for monitoring and decision-making in smart vehicles.

6. Objectives

The primary objectives of the Thermo-Drive project are:

1. To simulate Arduino-based temperature and vehicle distance sensors.
2. To generate dynamic alerts for unsafe conditions.
3. To visualize real-time sensor data using interactive graphs.
4. To log all sensor readings in a structured CSV format for historical analysis.
5. To implement both manual and automated monitoring modes.
6. To provide a practical demonstration of complex problem-solving in safety-critical systems.

7. Methodology

The development of Thermo-Drive followed a structured methodology:

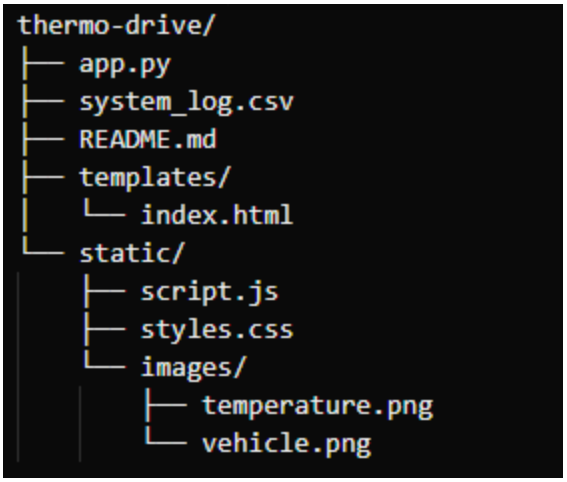
1. **Sensor Simulation**: Temperature and distance values are generated using Python's randomization functions to mimic Arduino sensors.
2. **Backend Development**: Flask is used to create RESTful APIs that provide temperature and vehicle distance data. Each API evaluates the data against thresholds and returns the corresponding status.
3. **Data Logging**: All readings are appended to system_log.csv along with timestamp, system name, value, and status.
4. **Frontend Development**: HTML and CSS define the interface, while JavaScript fetches data from the backend and updates the display. Plotly.js is used to render interactive graphs for historical and real-time trends.
5. **Auto Mode Implementation**: JavaScript intervals allow automatic fetching of sensor data at regular intervals for continuous monitoring.
6. **Alert Logic**: Temperature and distance values are compared against thresholds to assign statuses. Conflicting conditions are handled simultaneously to ensure accurate alerts.

8. Technology

The website leverages the following technologies:

- **Frontend**: HTML, CSS, JavaScript, Plotly.js for interactive graphs.
- **Backend**: Python Flask for APIs and server management.
- **Database/Logging**: CSV-based logging (system_log.csv).
- **Sensor Simulation**: Randomized data generation to mimic Arduino sensors.
- **Version Control**: GitHub for source code management.

File Structure:

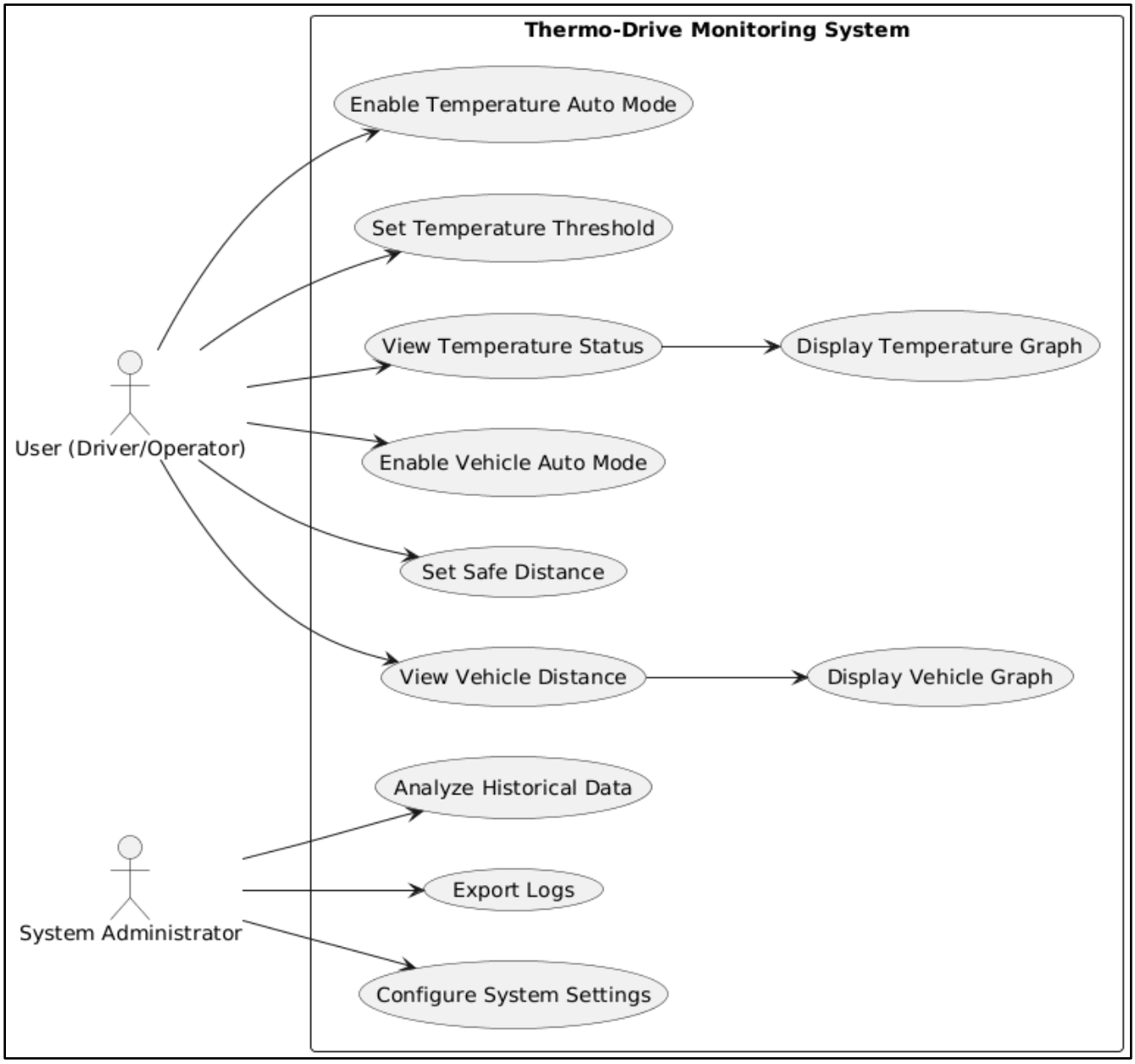


9. Features

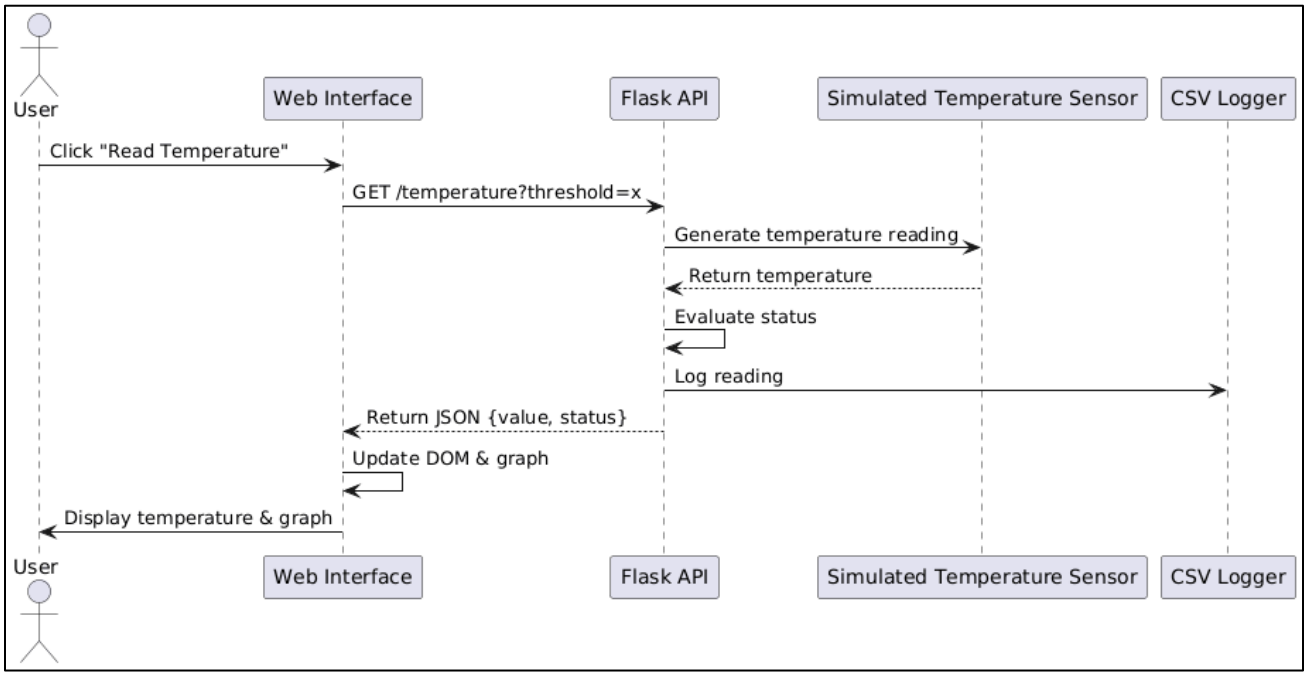
- Real-time temperature monitoring with alert statuses.
- Real-time vehicle detection with distance-based alerts.
- Manual and automatic monitoring modes for both systems.
- Data logging to CSV for historical analysis.
- Graphical visualization of sensor trends and statuses.
- Color-coded alerts for immediate recognition.
- User-friendly web interface with easy navigation.

10. Diagrams

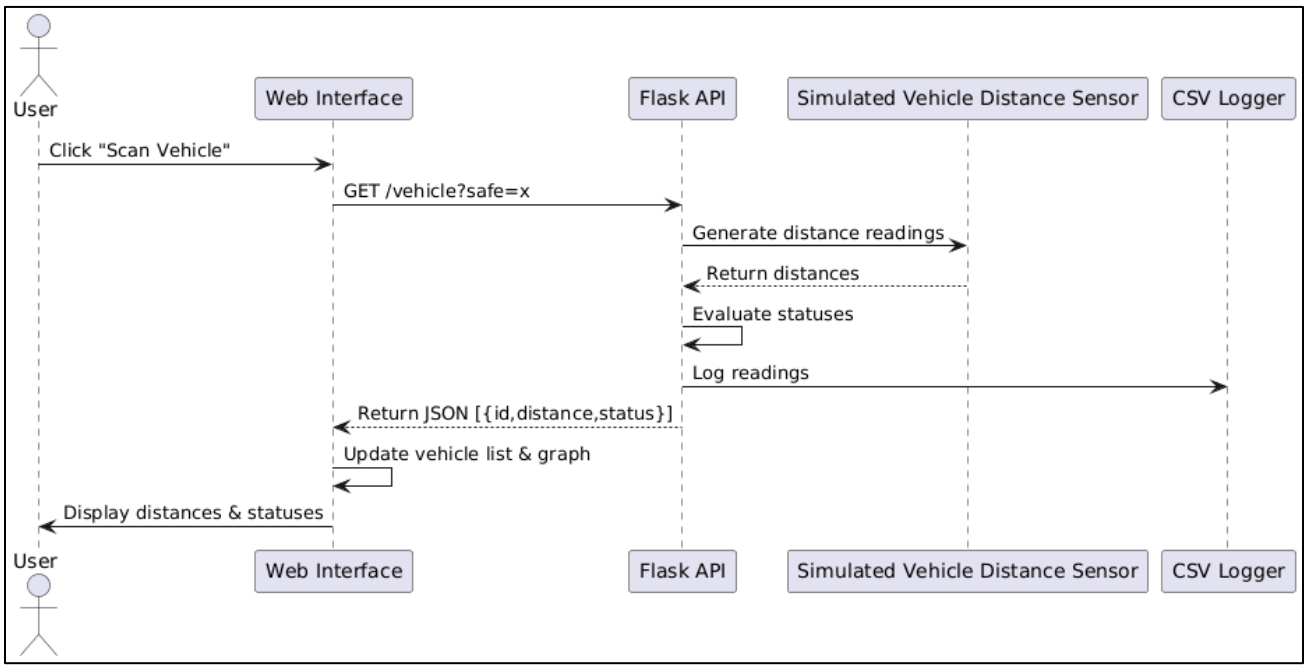
a) Use Case Diagram



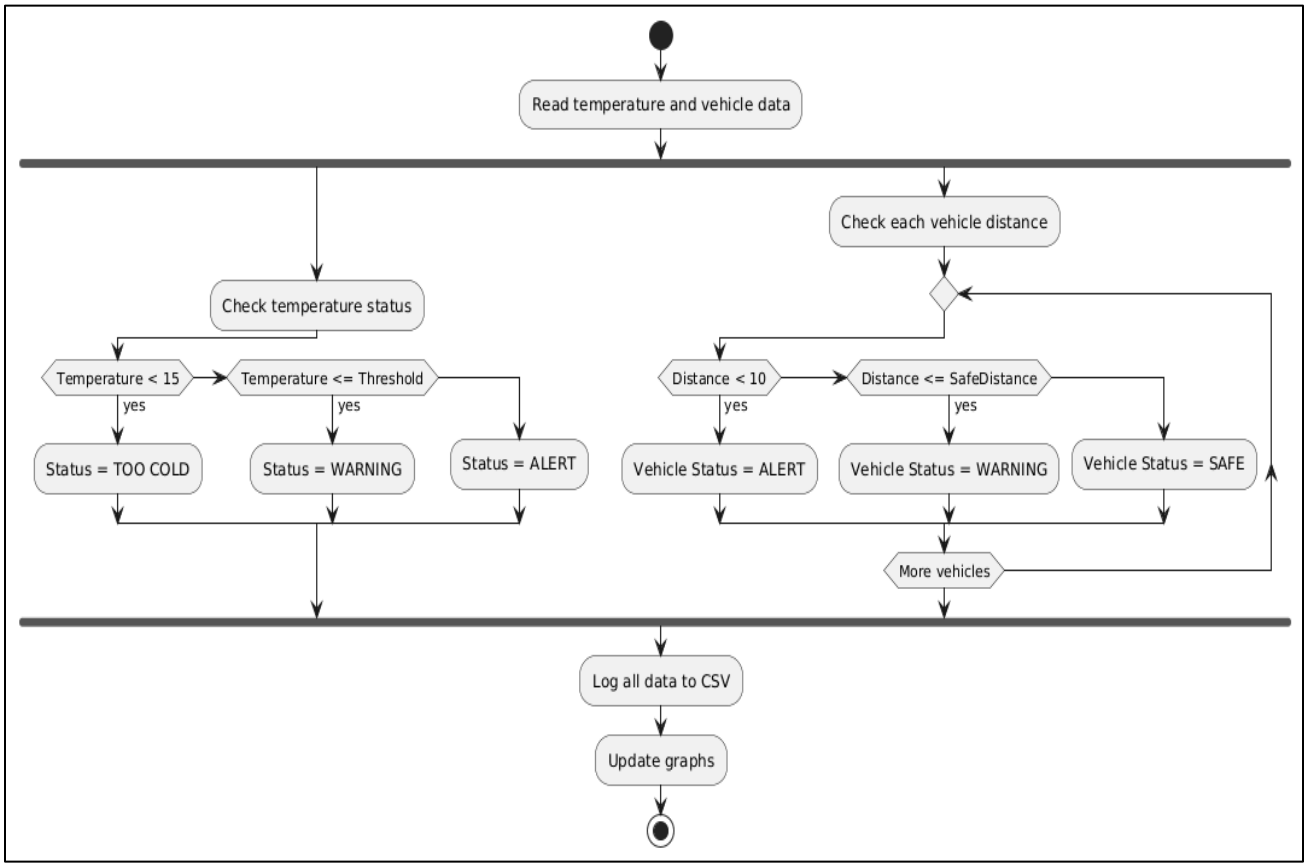
b) Sequence Diagram: Temperature Monitoring



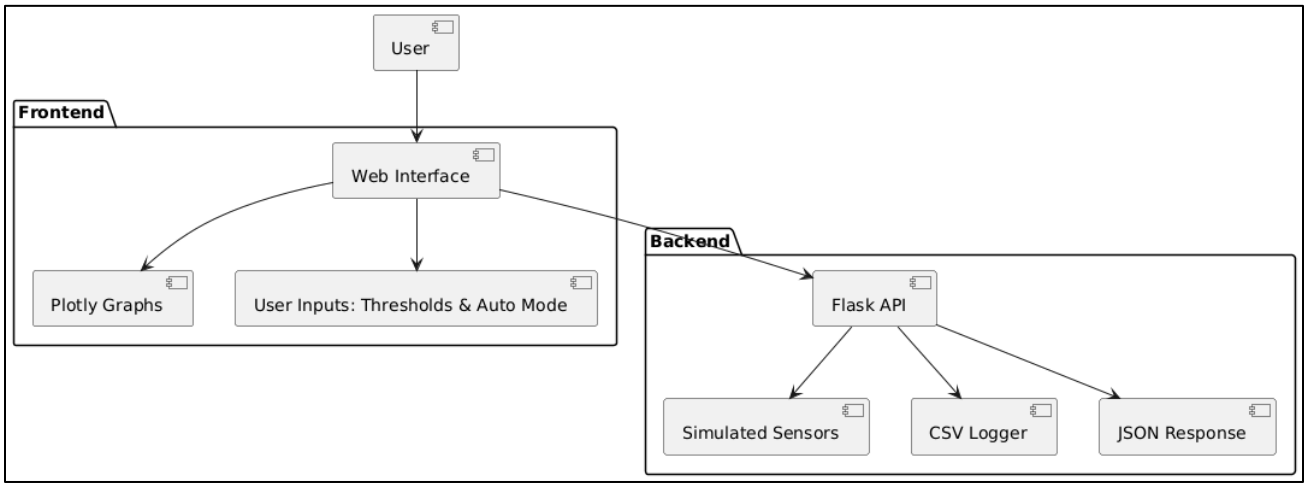
c) Sequence Diagram: Vehicle Detection



d) Activity Diagram: Alert Processing



e) Architecture Diagram



11. Database

The project uses a **CSV log file** to store historical sensor readings. All temperature readings and vehicle distance measurements are stored in `system_log.csv` in real time. This acts as a lightweight database for the system.

File: `system_log.csv`
C:\Users\Admin\Desktop\IOT CCP\thermo-drive\system_log.csv

Structure of the log file:

Column Name	Description
Time	Timestamp of the reading (DateTime)
System	Indicates which system the reading belongs to (`Temperature` or `Vehicle-<id>`)
Value	Measured value (temperature in °C, vehicle distance in cm)
Status	Status of the reading (e.g., `NORMAL`, `WARNING`, `ALERT`, `TOO COLD`, `SAFE`)

Example rows from `system_log.csv`:

Time	System	Value	Status
2026-01-03 19:05:10	Temperature	36.5	NORMAL
2026-01-03 19:05:12	Vehicle-1	18.7	WARNING
2026-01-03 19:05:12	Vehicle-2	50.3	SAFE

This allows for **trend analysis**, alert evaluation, and historical comparison.

12. Conclusion

The Thermo-Drive project successfully demonstrates a **complete monitoring system** for temperature and vehicle proximity in smart vehicles. The system provides real-time alerts, historical logging, and interactive visualization. It showcases **practical implementation of complex problem-solving**, handling conflicting thresholds, and detailed sensor data analysis.

This framework can be extended to real hardware sensors for actual deployment in smart vehicles. The project also highlights the importance of integrating frontend, backend, and data visualization technologies to deliver actionable insights in real-time.

13. Source Code

This project consists of a frontend (HTML/CSS/JS) and a backend (Flask Python server). The system monitors temperature and vehicle distances in smart vehicles using simulated Arduino sensors. It generates alerts for unsafe conditions, records data, and displays real-time visualizations.

Backend Implementation:

app.py C:\Users\Admin\Desktop\IOT CCP\thermo-drive\app.py

```
from flask import Flask, render_template, jsonify, request
import random, csv
from datetime import datetime
app = Flask(__name__)
LOG_FILE = "system_log.csv"
try:
   open(LOG_FILE, "r")
except:
   with open(LOG_FILE, "w", newline="") as f:
      csv.writer(f).writerow(["Time", "System", "Value", "Status"])
temperature = 36.0

@app.route("/")
def index():
   return render_template("index.html")
@app.route("/temperature")
def temperature_api():
   global temperature
   try:
      threshold = float(request.args.get("threshold", 45))
   except:
      threshold = 45
   temperature += random.uniform(-1.5, 1.5)
   temperature = max(-10, min(temperature, 65))
   if temperature < 15:
      status = "TOO COLD"
   elif temperature < 36:
      status = "NORMAL"
   elif temperature <= threshold:
```

```
        status = "WARNING"
    else:
        status = "ALERT"
    with open(LOG_FILE, "a", newline="") as f:
        csv.writer(f).writerow(
            [datetime.now(), "Temperature", round(temperature, 2), status]
        )
    return jsonify({"value": round(temperature, 2), "status": status})
@app.route("/vehicle")
def vehicle_api():
    try:
        safe = float(request.args.get("safe", 20))
    except:
        safe = 20
    vehicles = []
    for i in range(random.randint(1, 4)):
        dist = random.uniform(5, 150)
        if dist < 10:
            status = "ALERT"
        elif dist <= safe:
            status = "WARNING"
        else:
            status = "SAFE"
        vehicles.append({
            "id": i + 1,
            "distance": round(dist, 2),
            "status": status
        })
        with open(LOG_FILE, "a", newline="") as f:
            csv.writer(f).writerow(
                [datetime.now(), f"Vehicle-{i+1}", round(dist, 2), status]
            )
    return jsonify(vehicles)

@app.route("/logs/<system>")
def logs(system):
    values, statuses = [], []
    with open(LOG_FILE, "r") as f:
        reader = csv.reader(f)
```

```
        next(reader)
        for row in reader:
            if row[1].startswith(system):
                values.append(float(row[2]))
                statuses.append(row[3])
    return jsonify({"values": values, "statuses": statuses})
if __name__ == "__main__":
    app.run(debug=True)
```

Frontend Implementation:

script.js C:\Users\Admin\Desktop\IOT CCP\thermo-drive\static\script.js

```
let tempAuto = false, tempInterval = null;
let vehicleAuto = false, vehicleInterval = null;
function showSection(sec) {
    ["home", "temperature", "vehicle"].forEach(id => {
        document.getElementById(id).classList.add("hidden");
    });
    document.getElementById(sec).classList.remove("hidden");
}
function readTemp() {
    let t = document.getElementById("tempThreshold").value;
    fetch(`/temperature?threshold=${t}`)
        .then(r => r.json())
        .then(d => {
            document.getElementById("tempValue").innerText = d.value;
            let s = document.getElementById("tempStatus");
            s.innerText = d.status;
            s.className =
                d.status === "TOO COLD" ? "status cold" :
                d.status === "ALERT" ? "status alert" :
                d.status === "WARNING" ? "status warning" :
                "status normal";
            updateTempGraph();
        });
}
```

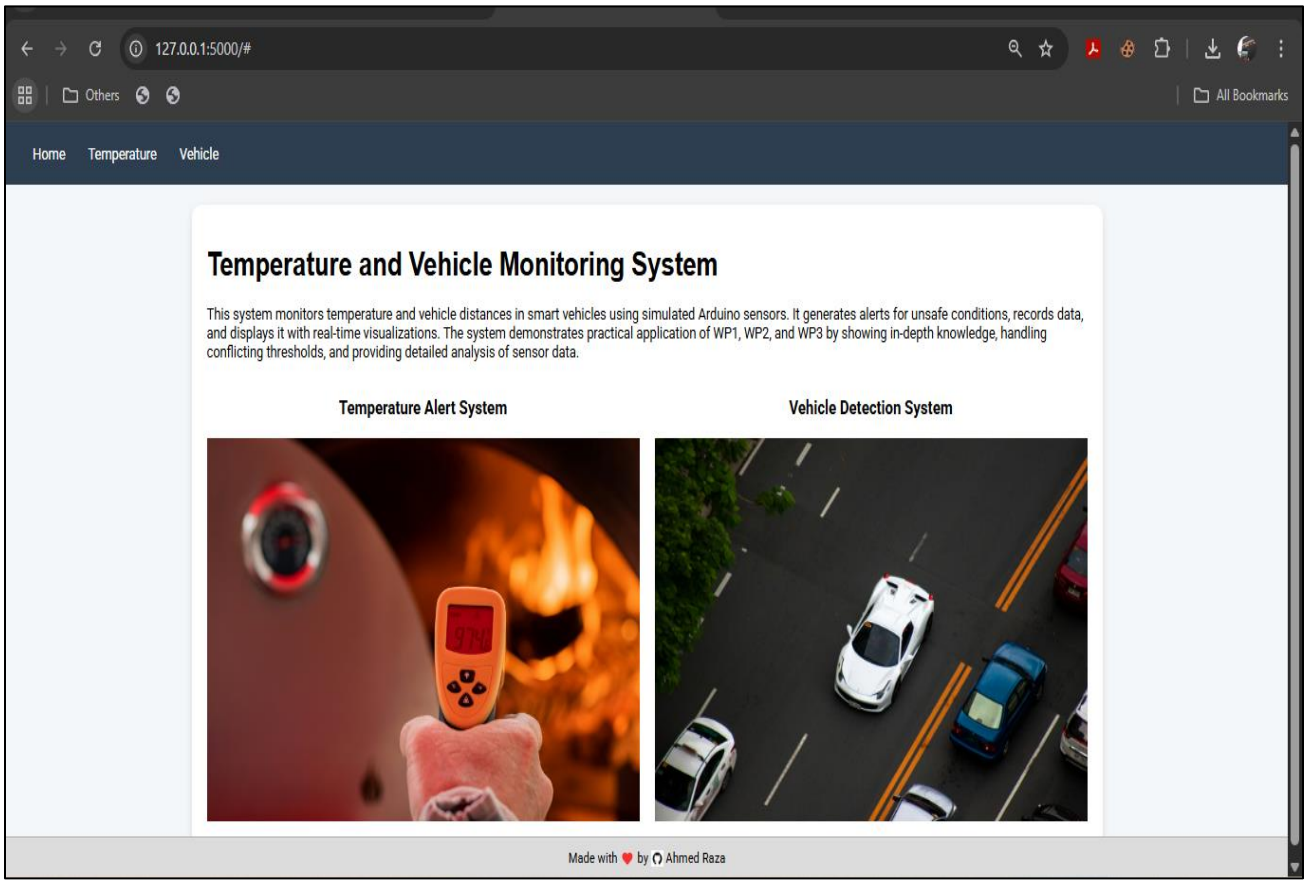
```
function toggleTempAuto() {
   if (tempInterval) clearInterval(tempInterval);
   tempAuto = !tempAuto;
   if (tempAuto) tempInterval = setInterval(readTemp, 2000);
}
function updateTempGraph() {
   fetch("/logs/Temperature")
      .then(r => r.json())
      .then(d => {
         let traces = [];
         let x = d.values.map((_, i) => i);
         let colorMap = {
            "TOO COLD": "blue",
            "NORMAL": "green",
            "WARNING": "orange",
            "ALERT": "red"
         };
         for (let i = 0; i < d.values.length; i++) {
            let color = colorMap[d.statuses[i]];
            traces.push({
               x: [x[i]],
               y: [d.values[i]],
               mode: "markers",
               marker: { color: color, size: 8 },
               showlegend: false
            });
            if (i > 0) {
               traces.push({
                  x: [x[i - 1], x[i]],
                  y: [d.values[i - 1], d.values[i]],
                  mode: "lines",
                  line: { color: color, width: 3 },
                  showlegend: false
               });
            }
         }
         Plotly.newPlot("tempGraph", traces, {
            title: "Temperature",
            margin: { t: 40 }
```

```
        });
    });
}
function scanVehicle() {
    let s = document.getElementById("safeDistance").value;
    fetch(`/vehicle?safe=${s}`)
        .then(r => r.json())
        .then(d => {
            let out = "<ul>";
            d.forEach(v => {
                out += `<li class="${
                    v.status === "ALERT" ? "alert" :
                    v.status === "WARNING" ? "warning" : "normal"
                }">
                    Vehicle ${v.id}: ${v.distance} cm (${v.status})
                </li>`;
            });
            out += "</ul>";
            document.getElementById("vehicleOutput").innerHTML = out;
            updateVehicleGraph();
        });
}
function toggleVehicleAuto() {
    if (vehicleInterval) clearInterval(vehicleInterval);
    vehicleAuto = !vehicleAuto;
    if (vehicleAuto) vehicleInterval = setInterval(scanVehicle, 1500);
}
function updateVehicleGraph() {
    fetch("/logs/Vehicle")
        .then(r => r.json())
        .then(d => {
            let traces = [];
            let x = d.values.map((_, i) => i);
            let colorMap = {
                "NORMAL": "green",
                "WARNING": "orange",
                "ALERT": "red"
            };
            for (let i = 0; i < d.values.length; i++) {
```

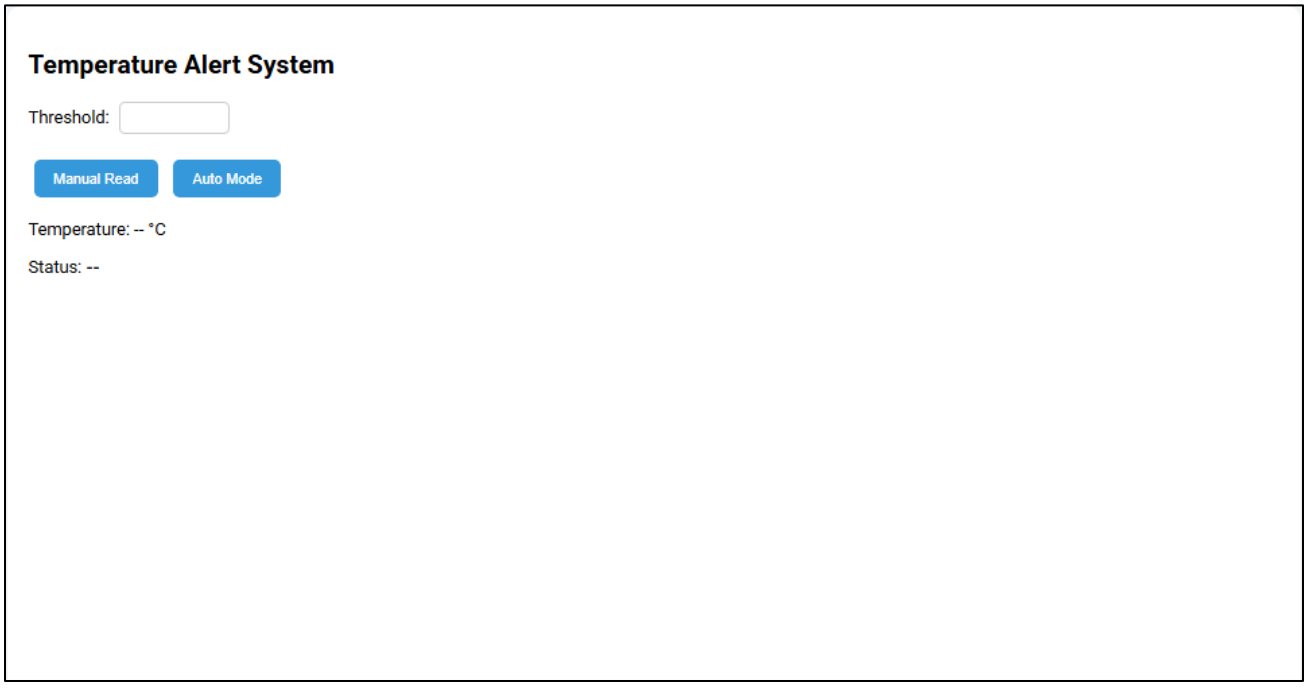
```
            let color = colorMap[d.statuses[i]];
            traces.push({
               x: [x[i]],
               y: [d.values[i]],
               mode: "markers",
               marker: { color: color, size: 8 },
               showlegend: false
            });
            if (i > 0) {
               traces.push({
                  x: [x[i - 1], x[i]],
                  y: [d.values[i - 1], d.values[i]],
                  mode: "lines",
                  line: { color: color, width: 2 },
                  showlegend: false
               });
            }
         }
         Plotly.newPlot("vehicleGraph", traces, {
            title: "Vehicle Distance",
            margin: { t: 40 }
         });
      });
}
showSection("home");
```

14. SnapShots

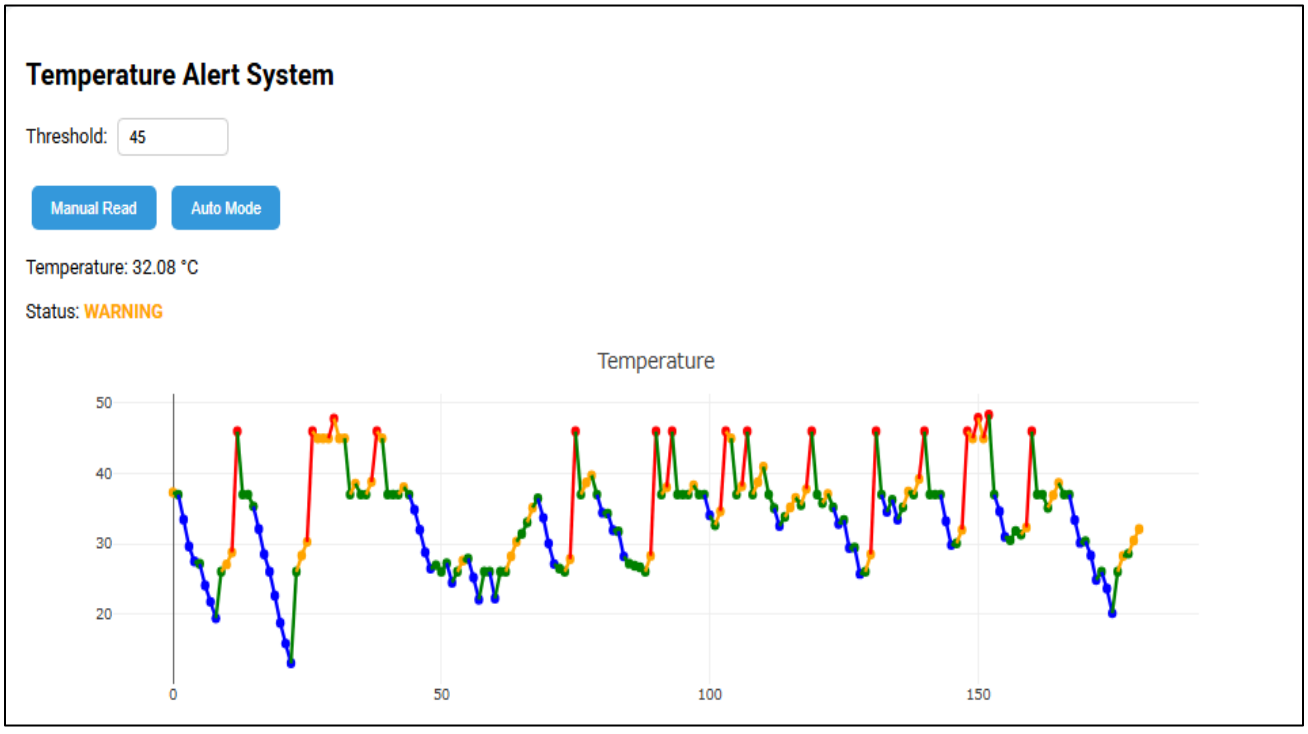
Home Screen: Central dashboard with system overview and navigation to monitoring sections.



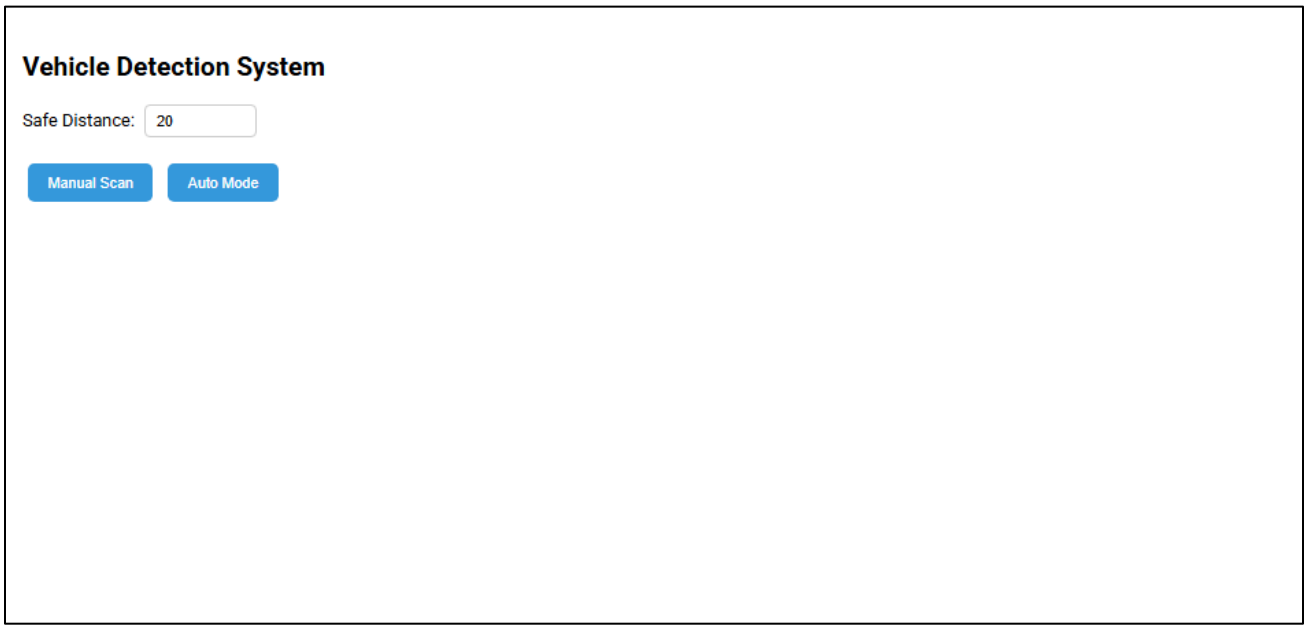
Temperature Page: Real-time temperature monitoring interface with threshold configuration, manual/auto modes and color-coded status.



Temperature Graph: Interactive visualization of temperature trends with status-based color coding.



Vehicle Page: Vehicle proximity detection interface with safe distance setting, manual/auto scanning, and real-time status display.



Vehicle Graph: Interactive chart of vehicle distance history with proximity alert color indicators.