Diagnostic Trouble Code (DTC) Proof of Concept

1. Objective of the POC

The primary objective of this Proof of Concept (POC) was to design and demonstrate a **true diagnostics system** capable of detecting fault conditions and generating a **Diagnostic Trouble Code (DTC)**, rather than simple real-time monitoring or data logging.

This POC specifically focuses on **Battery Temperature Sensor Fault Detection** using a clear, deterministic, and automotive-aligned approach.

> Scope note: This document strictly covers the **DTC use case only**. It does not include immobilizer logic, tilt detection, or unrelated features.

---

2. Problem Statement

Most existing implementations in the system were limited to:

- Reading sensor values
- Comparing against a threshold
- Pushing data to Firebase
- Displaying values on the cluster

This approach represents **monitoring**, not **diagnostics**.

A diagnostics system must:

- Detect abnormal or faulty behavior
- Confirm faults over time (debounce / validation)
- Generate a standardized fault code (DTC)
- Clearly differentiate between *normal*, *warning*, and *fault* states

---

3. Initial Approaches Attempted

During early exploration, multiple sensing approaches were evaluated and later rejected for diagnostics purposes:

3.1 Potentiometer-Based Input

- Used to simulate analog fault conditions
- Rejected because:

- Not representative of a real vehicle fault
- Difficult to define meaningful fault semantics
- More suitable for demos, not diagnostics

3.2 3-Axis Motion / IMU Sensor

- Considered for fault detection based on motion anomalies
- Rejected because:
  - Motion changes are contextual, not deterministic faults
  - High false positives
  - Complex calibration and interpretation

3.3 Switch / Indicator-Based Faults

- Attempted to detect faults via switch removal or disconnection
- Observations:
  - Hardware pull-ups masked disconnection events
  - Fault detection was unreliable without additional circuitry

These experiments helped clarify a key insight:

> **A good DTC POC requires a sensor with clearly defined valid and invalid operating ranges.**

---

4. Finalized Approach: Temperature Sensor–Based DTC

4.1 Why Temperature Sensor (DS18B20)

The temperature sensor was finalized because it:

- Has well-defined electrical and physical limits
- Explicitly reports disconnection conditions
- Matches real automotive fault scenarios
- Supports deterministic fault logic

This made it ideal for a **first-principles diagnostics implementation**.

---

5. DTC Definition

Item	Description
DTC Code	**P0A1A**

Item	Description
Fault Name	Battery Temperature Sensor Fault
Category	Powertrain / EV Battery
Fault Type	Sensor disconnection / invalid data

---

6. Diagnostics Logic Design

6.1 Fault Conditions

A fault is detected if **any** of the following occur:

- Sensor reports `DEVICE_DISCONNECTED_C`
- Temperature value is outside valid physical limits
    - Below −40°C
    - Above +125°C

6.2 Fault Confirmation Strategy

To avoid false triggers:

- Fault must persist continuously for **3 seconds**
- Transient glitches are ignored
- Fault is latched only after confirmation time

This mirrors real automotive ECU behavior.

---

7. System Architecture

7.1 Data Flow

1. ESP32 reads temperature sensor
2. Diagnostics logic evaluates fault conditions
3. On confirmed fault:
    - DTC code is generated
    - Status is logged
4. Data and alerts are pushed to Firebase
5. Display/Cluster team reads directly from Firebase
6. Warning icon and message are shown on cluster

7.2 Role of Each Component

Component	Responsibility
ESP32	Sensor reading + diagnostics logic
Diagnostics Module	Fault detection & confirmation
Firebase	Central data and alert source
Cluster / Display	Visual warning & icon display

Note: OKT board integration was intentionally excluded from this POC to reduce complexity and focus on diagnostics logic.

---

8. Why Firebase-Based Integration Was Chosen

- Decouples embedded diagnostics from UI logic
- Allows display team to work independently
- Enables rapid iteration and validation
- Simplifies debugging and demonstration

This ensured that diagnostics logic correctness—not communication plumbing—was the primary focus.

---

9. Challenges Faced

9.1 False Fault Triggers

- Initial implementations triggered faults immediately
- Solved by adding fault confirmation time

9.2 Hardware Assumptions

- Switch-based fault assumptions failed due to pull-ups
- Reinforced need for sensor-aware diagnostics

9.3 Conceptual Confusion: Monitoring vs Diagnostics

- Clarified that threshold alerts ≠ DTCs
- Diagnostics must represent *system health*, not values

---

10. Key Learnings

- Diagnostics require **state validation over time**
- Not all sensors are suitable for DTCs

- Fault semantics are more important than raw data
- Clear separation between:
    - Monitoring
    - Alerts
    - Diagnostics

---

11. Current Status of the POC

- Battery Temperature Sensor DTC implemented
- Fault detection validated
- Firebase integration complete
- Cluster display successfully shows warning
- POC objectives achieved

---

12. Future Enhancements (Optional)

- Add multiple DTCs
- Fault history and counters
- DTC clear conditions
- CAN or ECU-level integration
- Standardized ISO DTC mapping

---

13. Conclusion

This POC successfully demonstrates a **true diagnostics workflow** aligned with automotive principles. By carefully selecting the sensor, defining fault semantics, and implementing confirmation logic, the system moves beyond simple monitoring into meaningful fault detection.

The approach is scalable, clear, and suitable as a foundation for future ECU-level diagnostics development.