









Server Sentinel

A state-driven environmental monitoring system for mission-critical server rooms

Contents

- Project Brief 
- Features ✨
- Key Parameters 
- System Architecture 
- System Flowcharts 
- Key Assumptions 
- Project Plan 
- Conclusion 
- Team 



Project Brief

☀️ Project Objective

Students will develop software-only simulations of embedded systems using the C programming language. This is a foundational exercise intended to simulate real-world behavior of embedded devices while focusing on clean coding practices and logical system design. Graphical features are optional and may be developed using SDL or any cross-platform GUI library.



Project Allocation

Our group has been assigned with following project along with custom parameters.

Project Description

Temperature Logger – Monitors and logs temperature values and provides alerts.

Group 20:

- Project: Temp/Humidity Logger
- Alert Threshold: 60
- Log Size: 120



Features

Our implementation expands on the basic requirements with a comprehensive approach:

- **Advanced State Management:** Four operational states(NORMAL,CAUTION,DANGER,SHUTDOWN) 🟢🟡🔴💀
- **Dual-Parameter Monitoring:** Tracks both temperature and humidity with distinct thresholds 🌡️💧
- **Circular Buffer Logging:** Maintains a 120-entry historical log with automatic oldest-entry replacement 🔄📝
- **Escalating Alert System:** Progressive alerts based on environmental conditions 🚨📈
- **Simulated Scenarios:** Multiple environmental test scenarios including cooling failures and humidity spikes 🧊💧
- **Time-Based Shutdown Protocol:** Automatic system shutdown after sustained critical conditions ⌚🔴

Key Parameters

Parameter	Condition	Description
Caution Temperature	> 45 °C	System enters CAUTION state ⚠
Critical Temperature	> 60 °C	System enters DANGER state 🔥
Caution Humidity	> 70% or < 30%	System enters CAUTION state 💧 ⚠
Critical Humidity	> 80% or < 20%	System enters DANGER state 💧 🔥
Shutdown Timer	20 simulated seconds	Time at Critical Temperature before SHUTDOWN ⌚


System Architecture

Overview

The Server-Sentinel-C system is designed as a modular, state-driven application that simulates a mission-critical environmental controller for data centers. The system follows the Single Responsibility Principle, with each module having a clearly defined role.

Modules

1. Main Controller

 **Purpose:** Manages the main application loop and system state.

 **Responsibilities:**

- ✓ Initialize all subsystems
- ✓ Maintain the main program loop
- ✓ Coordinate between other modules

2. 🧠 System Logic

🎯 **Purpose:** Contains the core logic for threshold checking and state transitions.

📋 **Responsibilities:**

- ✓ Compare sensor readings against thresholds
- ✓ Manage system state transitions (NORMAL, CAUTION, DANGER, SHUTDOWN)
- ✓ Track time spent in critical states

3. 📊 Smart Data

🎯 **Purpose:** Generate simulated environmental data.

📋 **Responsibilities:**

- ✓ Simulate various environmental scenarios (normal, cooling failure, etc.)
- ✓ Produce realistic temperature and humidity readings
- ✓ Respond to user commands to change environmental conditions

4. 📝 Logger

🎯 **Purpose:** Manage the 120-entry circular log buffer.

📋 **Responsibilities:**

- ✓ Record all sensor readings
- ✓ Maintain circular buffer when maximum capacity is reached
- ✓ Provide access to historical data

5. 💻 User Interface

🎯 **Purpose:** Handle console input/output.

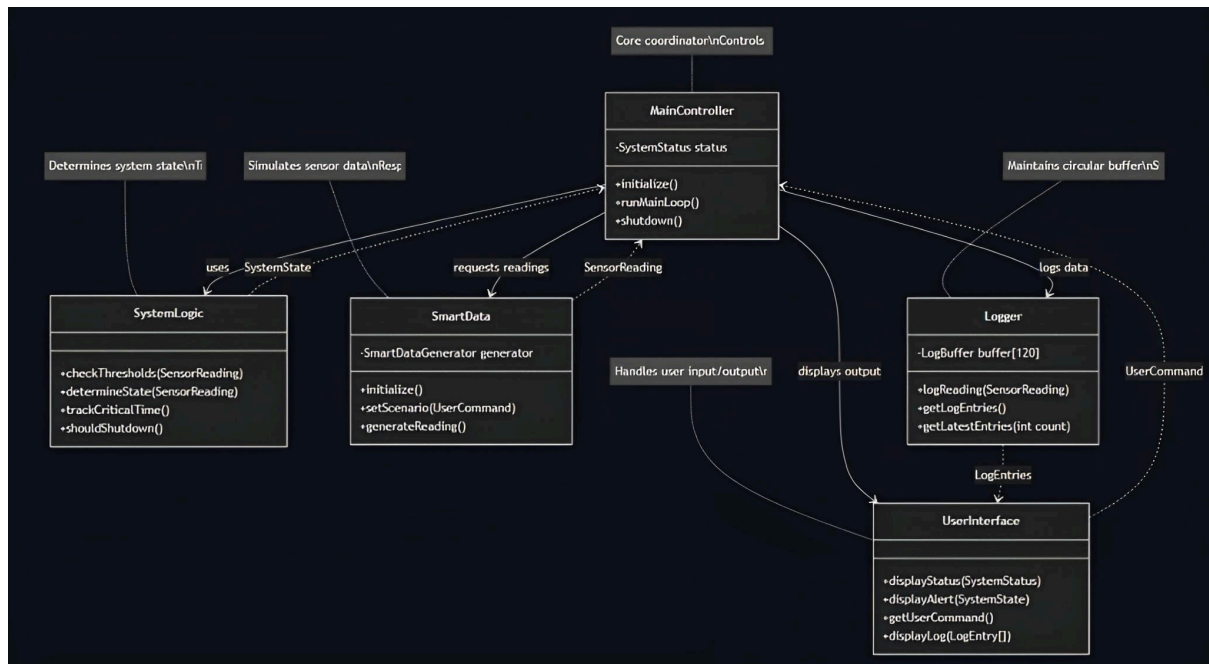
📋 **Responsibilities:**

- ✓ Process user commands
- ✓ Display system status and alerts
- ✓ Present log data in a readable format

🤝 Module Interactions

- ➡ Main Controller initializes all modules and manages the program flow.
- ➡ Smart Data generates readings that are passed to the System Logic.
- ➡ System Logic determines the system state based on readings.
- ➡ Logger records all readings regardless of system state.
- ➡ User Interface processes commands that influence the Smart Data module.

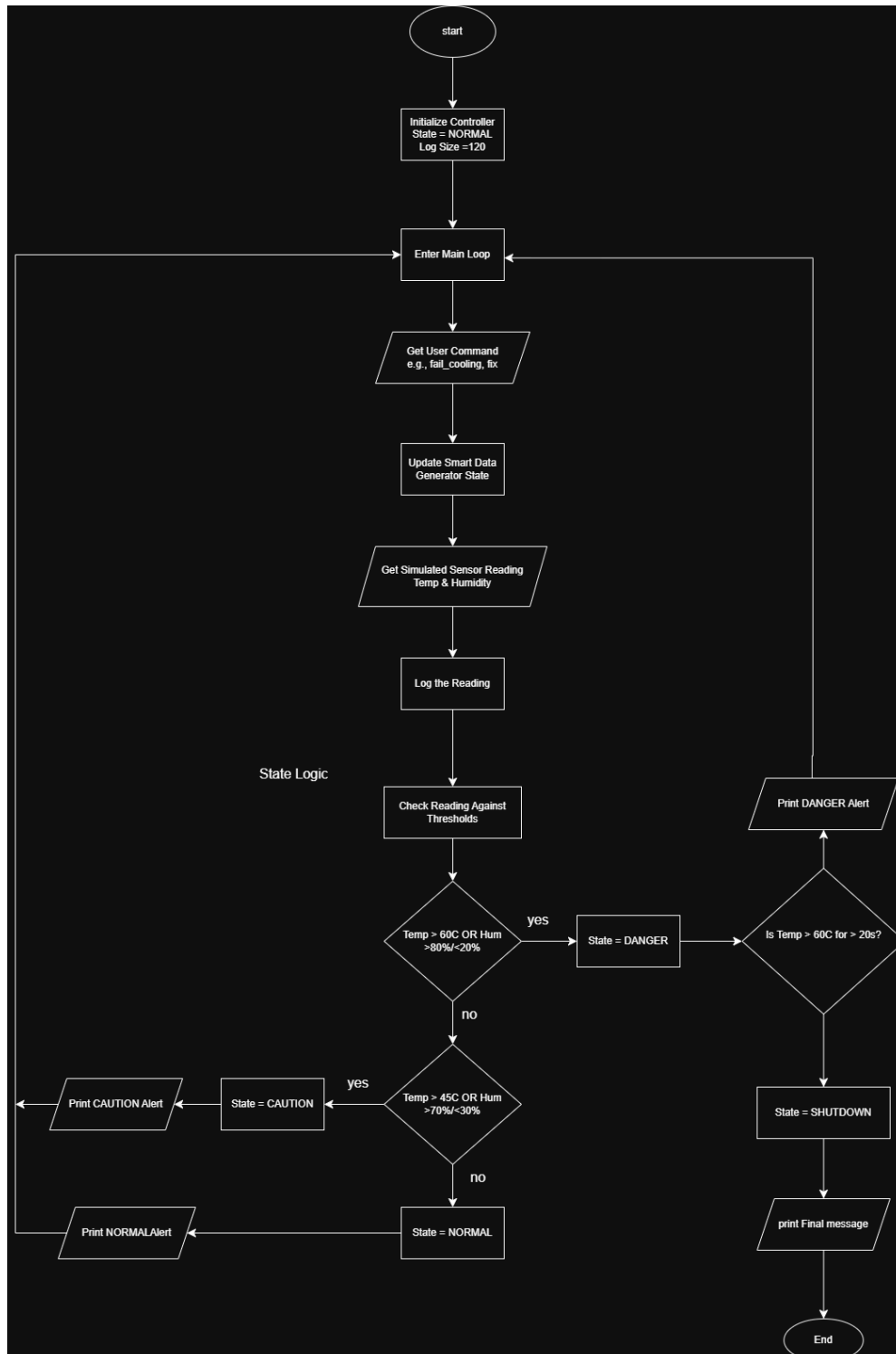
Architecture Diagram



System Flowcharts

Main System Flow

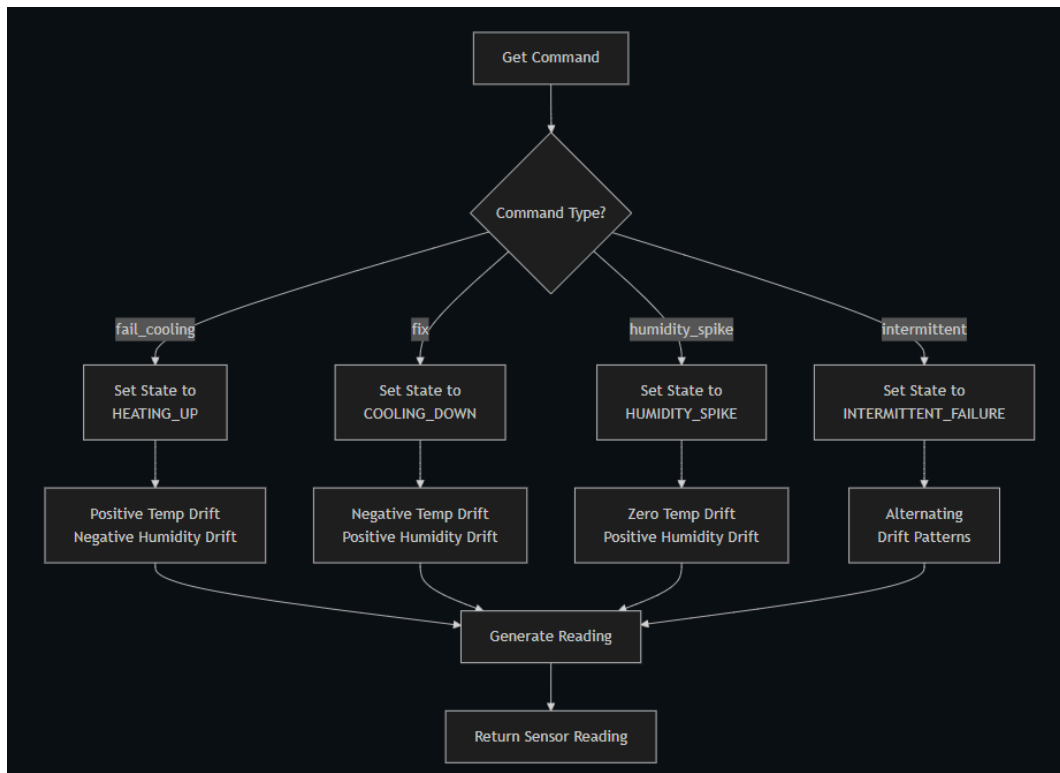
The following flowchart represents the core logic of the Server-Sentinel-C system:



[<https://drive.google.com/file/d/1pr3rTScCZTuQi83qDMRnbcUkF694pvjU/view?usp=sharing>]

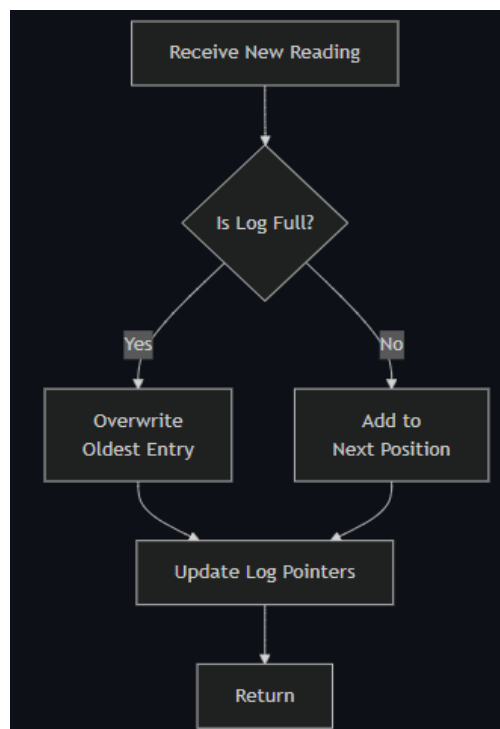
🧠 Smart Data Module Flow

The following flowchart shows how the Smart Data module processes commands and generates sensor readings:



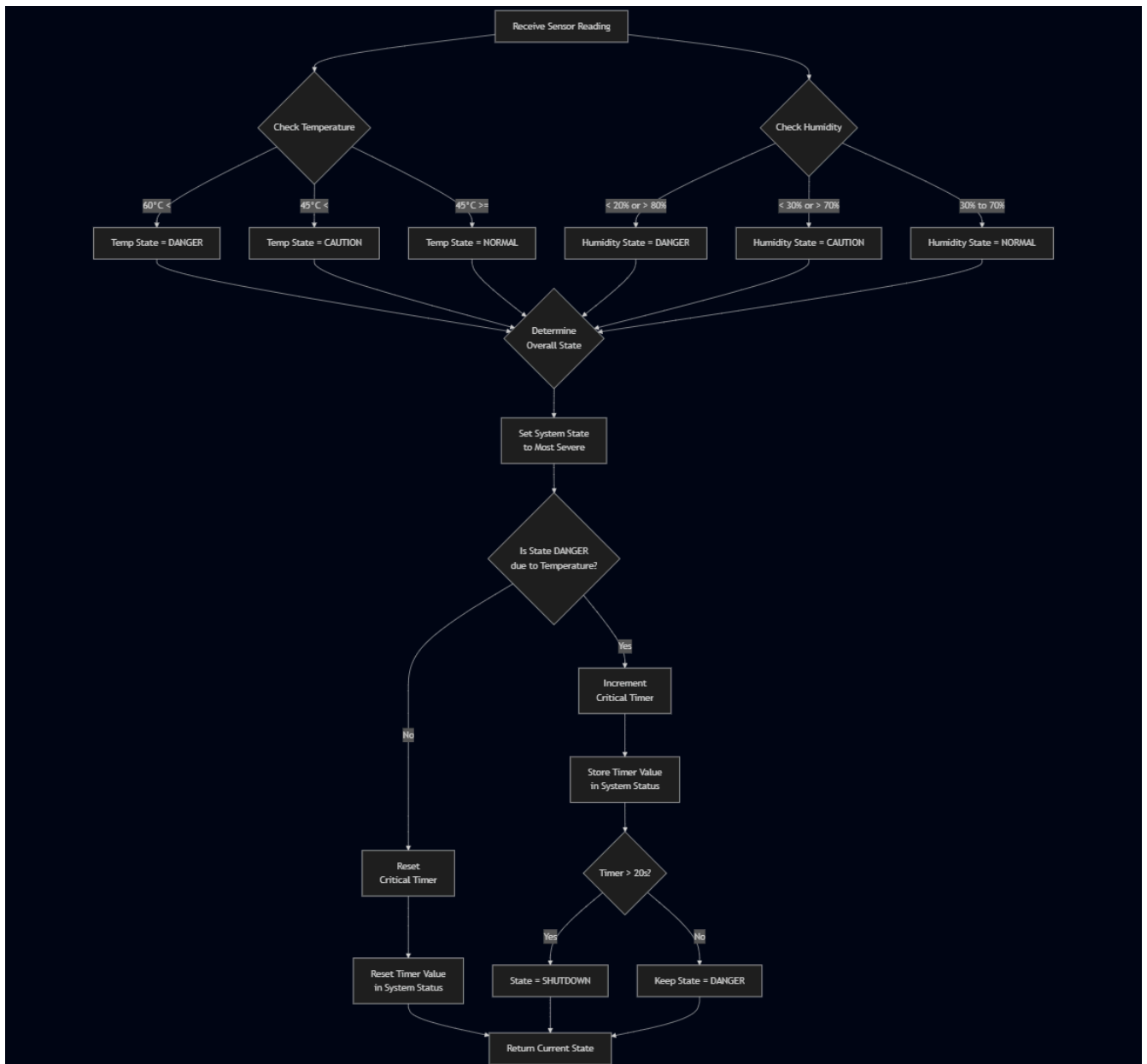
📝 Logger Module Flow

The following flowchart illustrates the circular buffer mechanism of the Logger module:



💡 System Logic Module Flow

The following flowchart illustrates the system logic module in great detail:

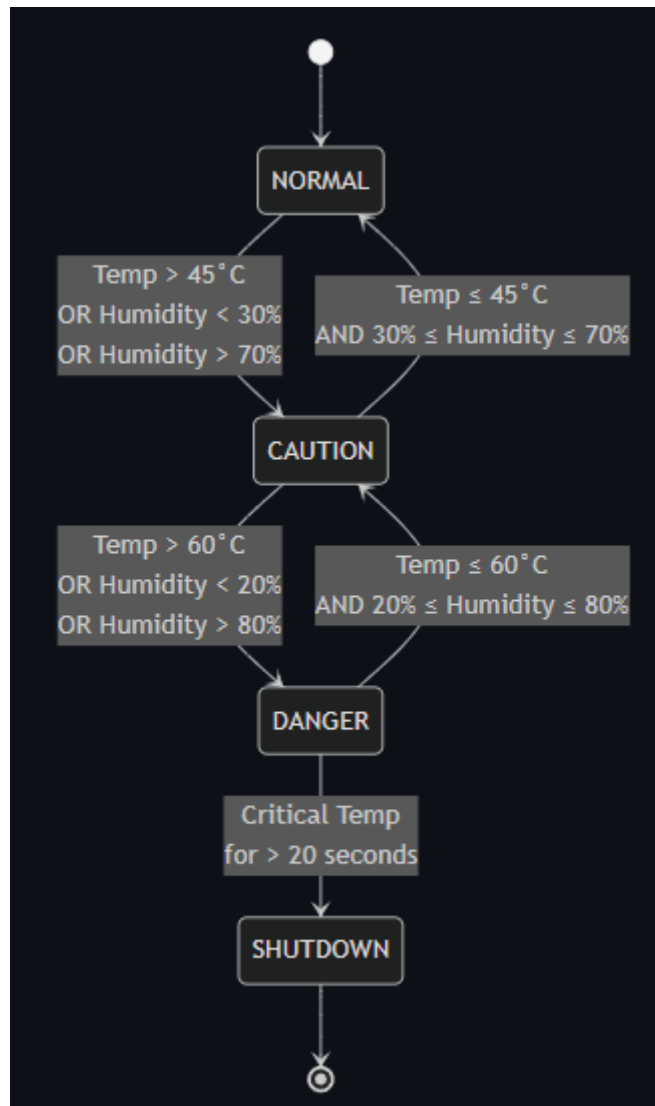


Visit this link to look flow charts more clearly

[\[https://github.com/Oshadha345/server-sentinel-c/blob/main/docs/flowchart.md\]](https://github.com/Oshadha345/server-sentinel-c/blob/main/docs/flowchart.md)



System State Diagram



Visit this link to look flow charts more clearly

[\[https://github.com/Oshadha345/server-sentinel-c/blob/main/README.md\]](https://github.com/Oshadha345/server-sentinel-c/blob/main/README.md)

Key Assumptions

General Assumptions

- Temperature is the primary critical factor for system shutdown, while humidity triggers alerts but not shutdowns.
- Server rooms require quick response to environmental changes, so our system updates continuously with minimal delay between readings.
- Operators need both immediate alerts and historical data, which is why we implement a circular buffer for logging.
- The system must be capable of recovering from alert states when conditions improve, allowing for a return to normal operations after interventions.

Technical Assumptions

- The Smart Data module simulates realistic environmental behavior with gradual changes rather than random values, assuming that temperature and humidity follow physical principles with momentum and gradual transitions.
- A state-driven design pattern using enumerated types rather than conditional logic makes the code more maintainable and easier to debug.
- The circular buffer implementation assumes a fixed size of 120 entries as specified in the requirements, providing approximately two hours of monitoring at one reading per minute.

Phase 1: Architecture & Design (The Blueprint Phase) 🚧

- [x] Define system requirements ✅
- [x] Establish repository structure 📁
- [x] Complete system architecture documentation 🏛️
- [x] Design system flowcharts 🗺️
- [x] Define module interfaces 🔌
- [] Finalize data structures 🏗️

Phase 2: Core Logic Development (The Coding Phase) 📝

- [] Implement smart_data.c module 😊
- [] Implement logger.c module 📝
- [] Implement system_logic.c module 🧠
- [] Implement user_interface.c module 💻
- [] Implement main.c controller 🎮
- [] Create Makefile 🛠️

Phase 3: Testing & Refinement (The Validation Phase) 🔬

- [] Develop test plan 📋
- [] Execute scenario testing 🎲
- [] Code review and refactor 🔍
- [] Performance optimization 🚀

Phase 4: GUI Integration (Optional Phase) 🖼️

- [] Select GUI library 🖋️
- [] Design UI components 🧩
- [] Integrate with core logic 🔗
- [] GUI testing 🔬

Phase 5: Final Testing & Deployment (The Launch Phase) 🚀

- [] Final system testing 🔬
- [] Compile for different platforms 💻
- [] Create deployment package 📦
- [] Project presentation and documentation 🎤

Conclusion

Server-Sentinel-C represents a comprehensive approach to environmental monitoring for data centers, going beyond the basic requirements to create a robust, **state-driven system**. Our architecture emphasizes modularity, clear separation of concerns, and reliable state management to ensure mission-critical performance.

The system's design addresses several key challenges in environmental monitoring. By implementing a four-state progression (**NORMAL, CAUTION, DANGER, SHUTDOWN**), we enable graduated responses to changing conditions. The circular buffer logging system ensures that historical data is maintained without memory overflow issues, providing operators with the context needed to analyze trends and respond appropriately.

A particular strength of our design is the **time-based shutdown protocol**, which prevents false alarms from momentary spikes while ensuring quick response to sustained critical conditions. This balance between responsiveness and stability is essential in a data center environment where both unnecessary shutdowns and delayed responses can be costly.

The modular architecture we've developed not only meets the current requirements but also provides a foundation for future enhancements. As we progress through implementation phases, this clean separation of concerns will facilitate parallel development, comprehensive testing, and potential expansion to include additional environmental parameters or integration with physical hardware.

We believe this design demonstrates how the principles of clean code and logical system organization can be applied to create an embedded system simulation that is both functional and maintainable. The Server-Sentinel-C project illustrates that even with the constraints of the C programming language, it is possible to create an elegant, robust solution to complex monitoring challenges.

Team

Group 20 members with their responsibilities:

Name	Index	Role	Responsibilities
Rathnasiri R.S. (Rumal)	E/21/326	Developer	User interface and testing 🖥️✍️
Rathnayaka P.G.I.N.B. (Induka)	E/21/327	Developer	Logger module and data structures 📝
Ratnayake R.M.K.T. (Kaweesha)	E/21/334	Developer	Project documentation and reporting & Smart data simulation module 📖🧐
Samarakoon S.M.O.T. (Oshadha)	E/21/345	Developer	System architecture, integration, and coordination 🏗️
Samaranayaka W.W.M.A. (Asanga)	E/21/346	Developer	System logic implementation 🧠