SUSTAINABLE SHIP AND SHIPPING 4.0
Erasmus Mundus Joint Master Degree
Excellence in education for a new generation of Naval Architects and Marine Engineers

INDUSTRY 4.0 ENABLING TECHNOLOGIES

Report for an

INTELLIGENT SHIP HVAC SYSTEM WITH SMART CONTRACT INTEGRATED FOR BLOCKCHAIN-BASED MAINTENANCE

Presented By

OJOR PRAISE CHIDERA

DHALAYAN HARI

KACI WASSIM

Submitted To

PROF. PAULA FRAGA LAMAS
PROF. TIAGO M. FERNANDEZ CARAMES

ABSTRACT

Ever been on a ship where it's too hot, too cold, or the air just feels off? Onboard marine vessels, traditional ship HVAC systems run on fixed schedules and manual controls, so cabins often end up uncomfortable as a result of poor air-quality management. CO_2 buildup in enclosed compartments may go undetected until it poses a safety risk. To overcome these limitations, we developed an intelligent, cloud-connected HVAC system based on an ESP8266 NodeMCU for ship compartments. The system continuously monitors Temperature, Humidity, Occupancy, and CO_2 levels, applying threshold-based logic to drive heating, cooling, and ventilation in real time. Environmental data and alerts are published via MQTT for remote monitoring and historical logging.

We also created a Solidity smart contract on Ethereum to make maintenance and service transparent. The contract logs HVAC runtime, triggers maintenance requests when usage or temperature thresholds are exceeded and handles ETH-based payments to fuel and maintenance providers. The contract's transparent, tamper-proof records replace old paper logs, making sure service gets done on time.

In our tests, the system responded quickly when conditions changed, stayed reliably connected to Wi-Fi, and proved it can keep ship compartments both comfortable and safe to breathe while the smart contract records runtime events and automates maintenance workflows in a tamper-proof, decentralized way.

TABLE OF CONTENTS

TABLE OF FIGURES

INTRODUCTION

MOTIVATION AND BACKGROUND

Maritime vessels operate under highly variable environmental conditions—ranging from humid tropics to frigid polar waters—and must provide reliable climate control to keep crew comfortable, protect sensitive equipment, and comply with safety standards. Yet most shipboard HVAC systems still run on fixed schedules and manual controls. You set a thermostat and hope for the best—only to find cabins too stuffy or CO_2 levels climbing unnoticed until they become a hazard. At the same time, keeping up with routine maintenance often falls to paper logs and guesswork, meaning service can be missed or delayed.

To tackle both real-time comfort and long-term upkeep, we paired:

- An ESP8266-based, cloud-connected HVAC system that uses threshold-based logic to monitor Temperature, Humidity, Occupancy and CO_2, then automatically adjusts heating, cooling, or ventilation. It publishes the data and alerts over MQTT for live bridge dashboards and historical logging.
- A Solidity smart contract on Ethereum that logs HVAC runtime, triggers maintenance requests once usage or temperature thresholds are crossed and automates ETH-based payments to fuel and maintenance providers—ensuring service happens on schedule, backed by transparent, tamper-proof records.

This dual approach ensures cabins stay comfortable and safe in the moment, and that maintenance happens on schedule, backed by transparent, tamper-proof records.

PROBLEM IDENTIFICATION

Traditional shipboard HVAC falls short in three key ways:

- **FIXED OPERATION:** Heating, ventilation and air-conditioning run on continuous schedules or fixed thresholds, without regard for actual occupancy or changing ambient conditions.
- **BLIND TO AIR QUALITY AND HUMIDITY:** Without real-time monitoring of temperature, humidity and CO_2, poor ventilation and rising moisture levels can go unnoticed—allowing unsafe conditions to build up in enclosed compartments before anyone takes action.
- **ISOLATED CONTROLS & MANUAL MAINTENANCE:** Local control panels and paper-based logs mean there's no live dashboard on the bridge, no digital record of runtime, and no

automated alerts or service triggers—leaving maintenance to guesswork and paper trails rather than transparent, tamper-proof workflows.

PROJECT OBJECTIVES

To address the shortcomings of traditional shipboard HVAC and maintenance workflows, this project combines an Intelligent Cyber-Physical Control System (ICPS) with a blockchain-based smart contract. The integrated objectives are:

- **REAL-TIME SENSING:** Continuously monitor temperature, relative humidity, occupancy and CO_2 via DHT11, HC-SR04 and MQ-7 sensor, so air-quality and comfort never go unchecked.
- **AUTONOMOUS DECISION MAKING:** Onboard decision logic implementing threshold-based and occupancy-aware control to maintain temperature within 16°C – 25°C, humidity under 65%, and to trigger air-quality alarms only when spaces are occupied.
- **CLOUD CONNECTIVITY:** Use MQTT over Wi-Fi to feed live telemetry to a dashboard and archive historical logs, giving crew a centralized view of compartment conditions.
- **BLOCKCHAIN DRIVEN MAINTENANCE:** Deploy a Solidity smart contract that records HVAC runtime, automatically issues maintenance requests when usage or temperature thresholds are crossed and handles ETH payments—replacing paper logs and manual billing with transparent, tamper-proof workflows.

Together, these setups ensure ship compartments stay comfortable and safe in real time, while maintenance is guaranteed on schedule and backed by immutable records.

CHAPTER 1: ICPS SYSTEM DESIGN AND IMPLEMENTATION

OVERVIEW

Our Intelligent Ship HVAC (Heating, Ventilation and Air Conditioning) System sits at the heart of the cabin's climate loop, performing four main functions in a continuous cycle:

1. **MEASURING:** ESP8266 reads all sensors (temperature, humidity, occupancy, CO_2, and real-time clock).
2. **DECIDING:** On-board logic compares readings against comfort and safety thresholds (e.g. T < 25 °C & RH < 65 %; CO_2 alarm only if occupied).
3. **ACTUATING:** Relays switch the air-conditioning unit, heater, and fan; LEDs and a buzzer communicate status and alerts.
4. **PUBLISHING:** Every 5 seconds the system packages sensor data, actuator states, and any CO_2 alerts into a JSON message and sends it via MQTT.

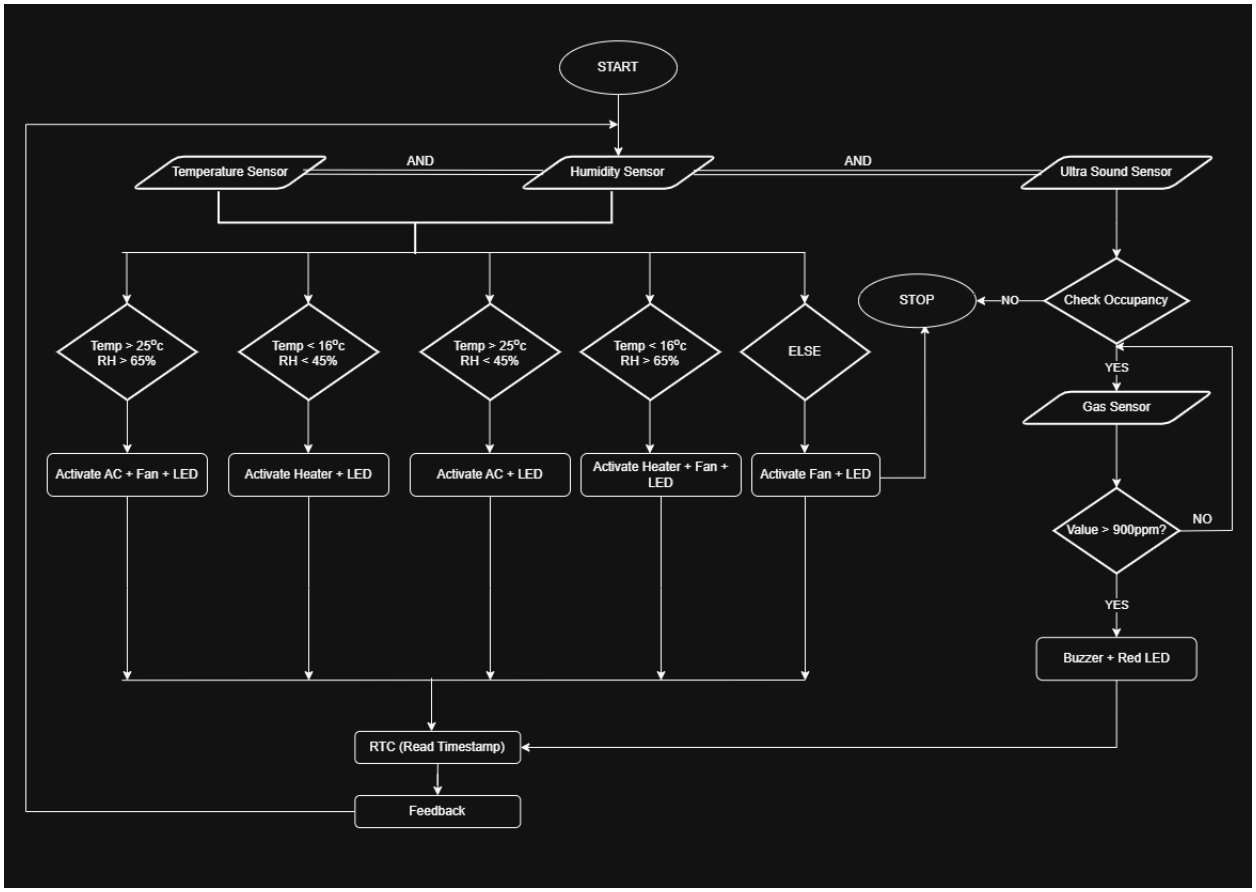

Figure 1: Flowchart of Ship HVAC System

The flowchart follows this sequence:

1. Read all sensors
2. Determine occupancy (distance < MAX_RANGE)
3. Apply HVAC thresholds:
    - **Too hot** (T > 25 °C) → AC + fan
    - **Too cold** (T < 16 °C) → heater (± fan if RH out of bounds)
    - **Comfort zone** → fan only
4. If occupied and CO_2 > 900 ppm → sound buzzer + red LED
5. Timestamp with RTC, publish JSON over MQTT
6. Wait 5s → loop

REQUIRED SENSORS AND ACTUATORS

To gather real-time environmental data and translate decisions into action, the system relies on a suite of sensors for monitoring and actuators for control:

1. **DHT11** – Temperature and Humidity
2. **HC-SR04** – Ultrasonic module for occupancy detection
3. **MQ-7** – CO_2 sensor
4. **DS1307 RTC** – Real-time clock for accurate timestamps
5. **LEDs** – Indicate mode (Yellow = AC, Red = Heater, Green = Fan)
6. **Buzzer** – Audible CO_2 alarm

COMMUNICATIONS ARCHITECTURE

The heart of the system's connectivity is the lightweight MQTT protocol running over the ship's Wi-Fi network.

1. **Wi-Fi CONNECTION**
    - The ESP8266 joins the vessel's onboard Wi-Fi using saved SSID and password credentials.
    - It maintains a persistent connection, automatically reconnecting if the link drops
2. **MQTT BROKER**
    - A broker (test.mosquitto) was used to handle message routing
    - The sensors publish data to the broker and the dashboard subscribes to the topic in real time.

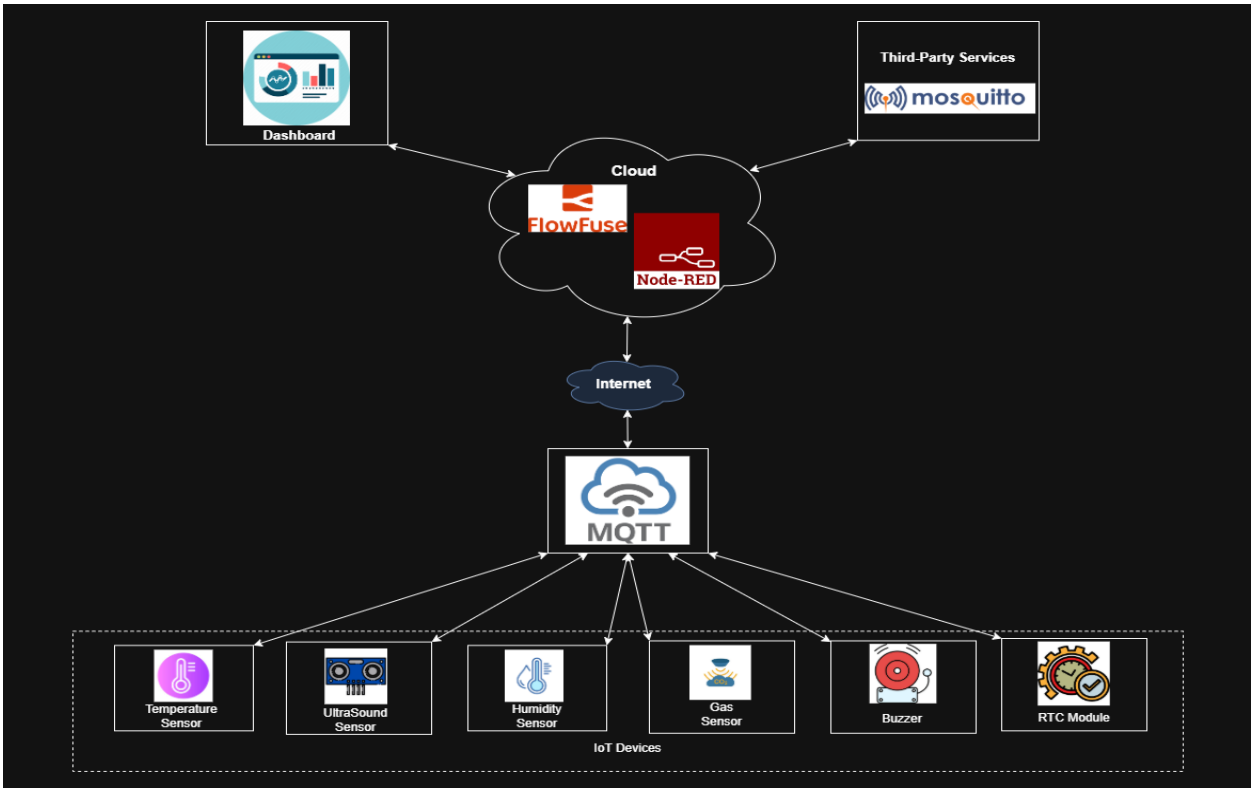

Figure 2: Cloud Architecture of Ship HVAC System

3. DASHBOARD INTEGRATION

- The dashboard subscribes to both topics to display live gauges (Temperature and Humidity), Occupancy, CO_2 and System alert indicators.

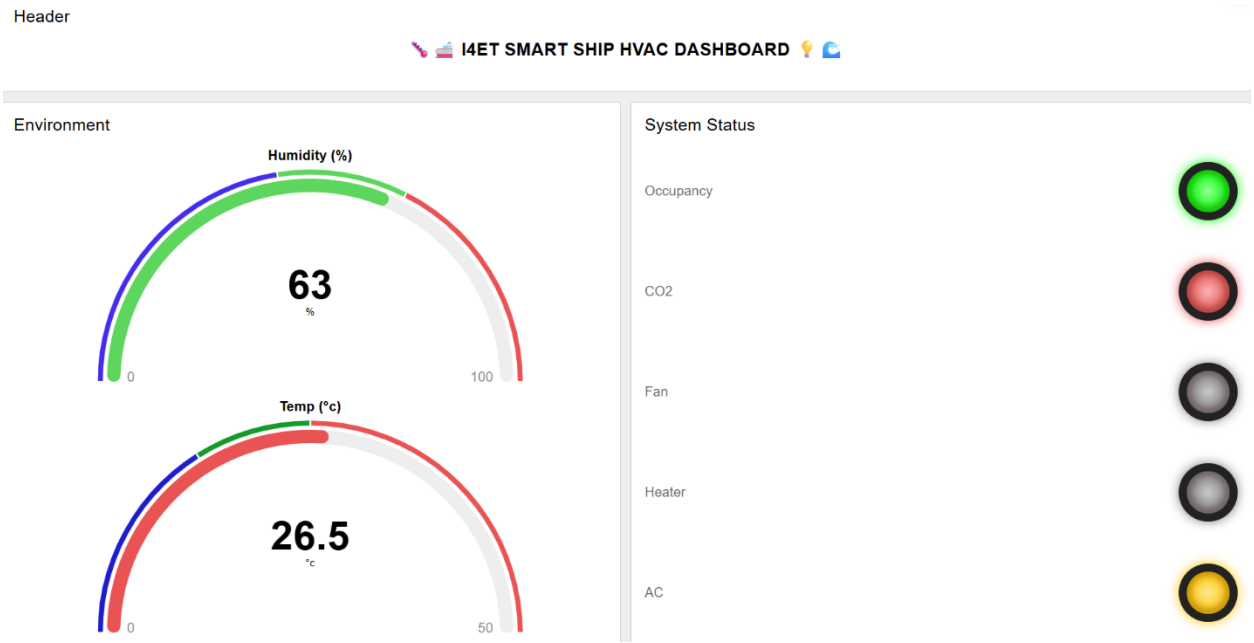

Figure 3: Ship HVAC System Dashboard

With this setup, any authorized station on the ship can keep an eye on compartment conditions, receive immediate warnings, and review past performance without manual downloads or local configuration.

SYSTEM IMPLEMENTATION

- ## HARDWARE

  The physical platform is built around an ESP8266 NodeMCU “device layer” that interfaces directly with all sensors and actuators. All connections follow the schematic in the figure below:

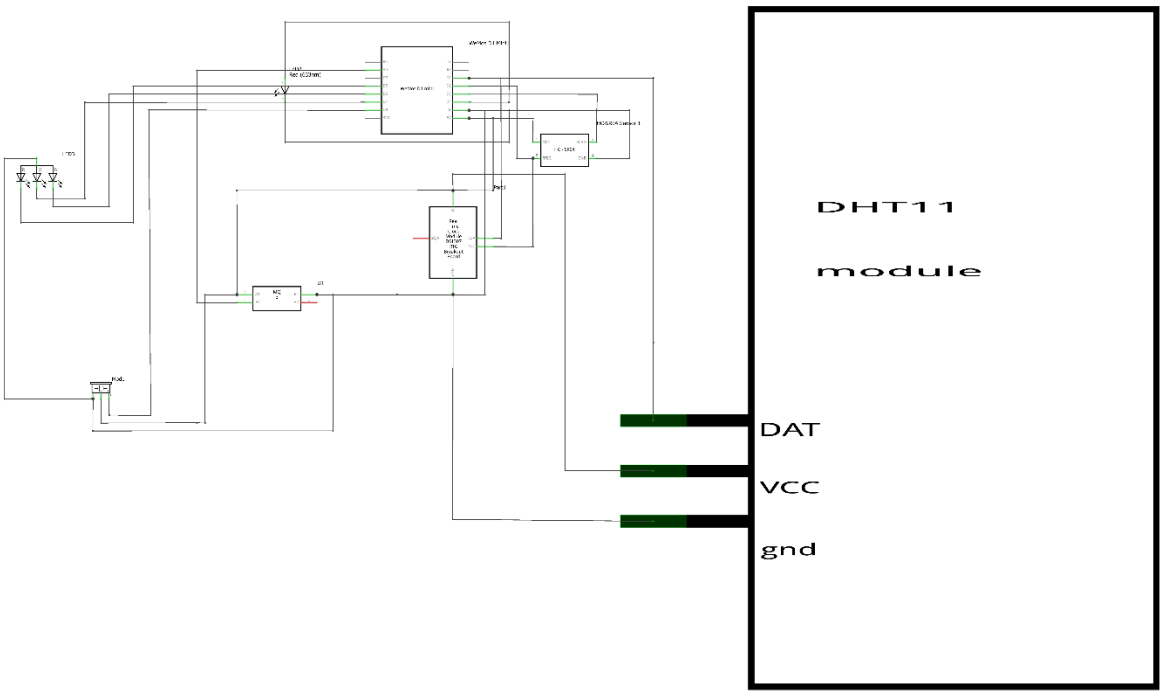

Figure 4: Schematic layout of System Architecture

- Temperature and Humidity: DHT11 data pin → ESP8266 digital I/O
- Occupancy: HC-SR04 ultrasonic module’s Trig/Echo → two digital I/O lines
- CO_2 Monitoring: MQ-7 analog output → ESP8266 A0
- Real-Time Clock: DS1307 via I^2C (SDA/SCL) for timestamping
- Status Indicators: LEDs (mode display) and a buzzer for high–CO_2 alarms

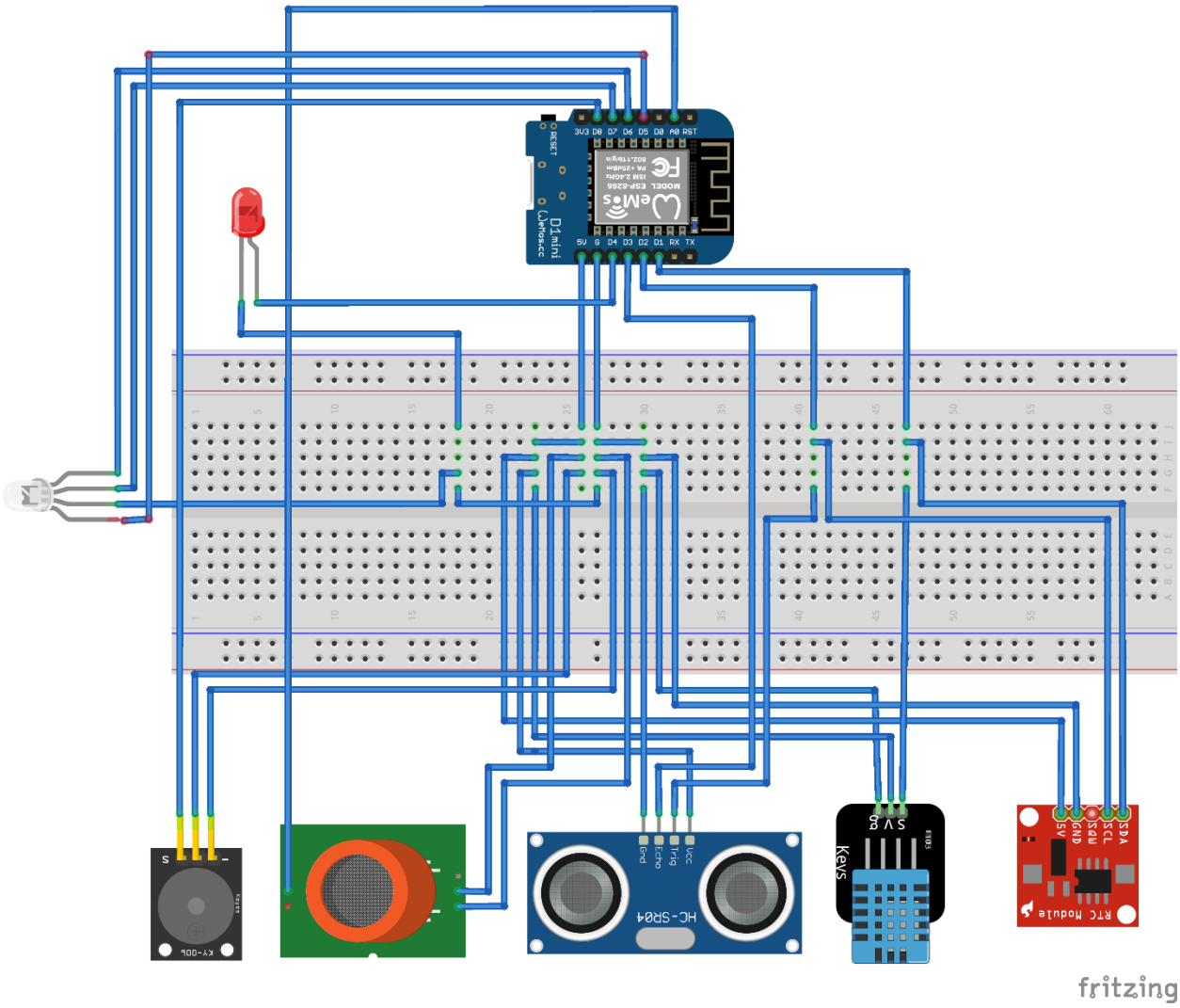

Figure 5: Breadboard view of the HVAC system hardware

- SOFTWARE

  The firmware follows the classic Arduino pattern, split into two main routines:

  - **Initialization (setup())**
    - Start Serial at 115200 bps for debugging
    - Initialize DHT, ultrasonic sensor, CO_2 sensor, and RTC
    - Configure all GPIO pins (inputs for sensors, outputs for relays/LEDs/buzzer)
    - Connect to Wi-Fi using saved SSID/password
    - Establish MQTT client with the bridge-side broker
    - Verify connections and register callback handlers

```
void setup_wifi() {
  WiFi.begin(SSID, PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
}

void reconnect_mqtt() {
  while (!Client.connected()) {
    if (Client.connect("I4ETHVACsensordata")) {
      // connected
    } else {
      delay(5000);
    }
  }
}

void setup() {
  Serial.begin(115200);
  dht.begin();
  rtc.begin();
  // Only run once to set RTC from compile time—then comment out:
  rtc.adjust(DateTime(F(__DATE__), F(__TIME__)));

  // pin modes
  pinMode(TRIGGER_PIN,    OUTPUT);
  pinMode(ECHO_PIN,       INPUT);
  pinMode(CO2_PIN,        INPUT);

  pinMode(YELLOWPIN,     OUTPUT);
  pinMode(GREENPIN,      OUTPUT);
  pinMode(REDPIN,        OUTPUT);
  pinMode(CO2_ALERT_LED, OUTPUT);
  pinMode(BUZZER_PIN,    OUTPUT);

  // Wi-Fi + MQTT
  setup_wifi();
  Client.setServer(MQTT_SERVER, MQTT_PORT);
}
```

Figure 6: Initialization Code block

- **Main Loop (loop())** (runs every 5 seconds)
  - Read temperature and humidity from DHT11
  - Trigger HC-SR04 and calculate distance → determine occupancy
  - Read analog CO_2 level; only evaluate alarm if occupied
  - Apply threshold logic:
    - ✓ **T > 25 °C** → AC + fan
    - ✓ **T < 16 °C** → heater (± fan for humidity)
    - ✓ **Else** → fan only
    - ✓ **CO_2 > 900 ppm & occupied** → buzzer + red LED
  - Timestamp the data via RTC
  - Serialize readings and actuator states into JSON
  - Publish payload over MQTT
  - Call client.loop() to maintain connection
  - Delay 5000ms

```
void loop() {
  if (!Client.connected()) reconnect_mqtt();
  Client.loop();

  // 1. Read sensors
  float temperature = dht.readTemperature();
  float humidity    = dht.readHumidity();
  unsigned int distance = sonar.ping_cm();
  bool occupied    = (distance > 0 && distance < MAX_DISTANCE);

  // 2. HVAC logic → LEDs only
  bool ac     = false,
       heater = false,
       fan    = false;

  if (temperature > 25.0 && humidity > 65.0) {
    ac  = true; fan = true;
    analogWrite(REDPIN,   0);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,255);
  }
  else if (temperature < 16.0 && humidity < 45.0) {
    heater = true;
    analogWrite(REDPIN,   255);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,0);
  }
  else if (temperature > 25.0 && humidity < 45.0) {
    ac = true;
    analogWrite(REDPIN,   0);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,255);
  }
  else if (temperature < 16.0 && humidity > 65.0) {
    heater = true; fan = true;
    analogWrite(REDPIN,   255);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,0);
  }
```

```
  else if (temperature > 25.0) {
    ac = true;
    analogWrite(REDPIN,   0);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,255);
  }
  else if (temperature < 16.0) {
    heater = true;
    analogWrite(REDPIN,   255);
    analogWrite(GREENPIN, 0);
    analogWrite(YELLOWPIN,0);
  }
  else {
    fan = true;
    analogWrite(REDPIN,   0);
    analogWrite(GREENPIN, 255);
    analogWrite(YELLOWPIN,0);
  }

  // 3. CO₂ logic (only if occupied)
  float co2_val   = 0.0;
  bool  co2_alert = false;
  if (occupied) {
    co2_val   = analogRead(CO2_PIN);
    co2_alert = (co2_val > 900);
    digitalWrite(BUZZER_PIN,    co2_alert ? HIGH : LOW);
    digitalWrite(CO2_ALERT_LED, co2_alert ? HIGH : LOW);
  } else {
    digitalWrite(BUZZER_PIN,    LOW);
    digitalWrite(CO2_ALERT_LED, LOW);
  }

  // 4. Update traffic R-Y-G LEDs
  setSystemLEDs(ac, fan, heater);
```

```

  // 5. Timestamp & payload
  DateTime now = rtc.now();
  char ts[25];
  snprintf(ts, sizeof(ts),
           "%04d-%02d-%02d %02d:%02d:%02d",
           now.year(), now.month(), now.day(),
           now.hour(), now.minute(), now.second());

  String payload = "{";
  payload += "\"timestamp\":\"" + String(ts) + "\",";
  payload += "\"temperature\":" + String(temperature,1) + ",";
  payload += "\"humidity\":"    + String(humidity,1)    + ",";
  payload += "\"occupied\":"    + String(occupied?"true":"false") + ",";
  payload += "\"co2_level\":"   + String(co2_val,1)     + ",";
  payload += "\"co2_alert\":"   + String(co2_alert?"true":"false") + ",";
  payload += "\"ac\":"          + String(ac?"true":"false") + ",";
  payload += "\"fan\":"         + String(fan?"true":"false") + ",";
  payload += "\"heater\":"      + String(heater?"true":"false");
  payload += "}";

  Client.publish(MQTT_PUB_TOPIC, payload.c_str());
  Serial.println("Published: " + payload);

  delay(3000);
}
```

Figure 7: Main Loop Code Block

Key Libraries Used were:

- **ESP8266WiFi.h** for network connectivity
- **PubSubClient.h** for MQTT messaging
- **DHT.h** for temperature/humidity
- **NewPing8266.h** for ultrasonic distance
- **RTClib.h** for DS1307 RTC

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>
#include <DHT.h>
#include <NewPingESP8266.h>
#include <Wire.h>
#include "RTClib.h"
```

Figure 8: Libraries Used in Code Block

TESTS AND VALIDATION

- ## TEST SETUP

We evaluated our Smart Ship HVAC System in a room, with sensors mounted on a dashboard and we artificially altered each sensor's input to verify its behavior.

- For the DHT11 temperature and humidity sensor, we introduced hot, humid air using our breathe to adjust its temperature and humidity readings.
- For the HC-SR04 ultrasonic module, we placed objects at various distances within its detection range to simulate occupancy.
- Since the MQ-series gas sensor isn't CO_2-specific, we defined a working threshold in code and emulated high-CO_2 conditions to test the alarm logic.

- ## TEST PROCEDURES

Rather than formal calibration, we simply changed the environmental conditions we could control and observed how the system reacted:

- Temperature Change: We introduced hot air using our breathe to increase the temperature, checking that the AC engaged at the programmed thresholds. We also tried using ice packs to decrease the temperature around the sensor, but we did not achieve any reasonable readings

- Humidity Shift: We also used the breathe trick to introduce damp air in order to raise the humidity around the sensor above the set limits and confirmed that the system worked perfectly.
- Occupancy Simulation: We placed objects at various distances within the programmed range of the HC-SR04 sensor to trigger "occupied" readings, then moved them away to test "unoccupied" behavior.
- CO_2 Alarm Logic: Since the MQ-series sensor isn't CO_2-specific, we defined a code threshold of 900 ppm and emulated high-CO_2 conditions to test the alarm logic and verify the LED and buzzer responded correctly.
- Network Drop Test: Disabled the Wi-Fi for 30s, then restored it, to measure reconnection time and ensure no manual reset was needed.

- RESULTS

Figure 9: Payload Snippet on Debug Console

- Temperature Response: Blowing warm air over the DHT11, we observed it took a while for the sensor to adopt the actual temperature change but it eventually pushed the readings above the 25°C threshold, and the AC indicator turned on within 4 seconds on average. Attempts to cool the area around the sensor with ice packs only dropped the reported temperature by about 1°C and did not consistently trigger the heater indicator.
- Humidity response: Giving that the same sensor reads both temperature and humidity, we observed the same delayed adoption of the actual environmental changes while introducing moisture via breath though it eventually raised relative humidity above the 65% limit,

before the system triggered the fan mode and the indicator almost immediately—typically within one control loop (≈5 s).

- Occupancy Detection: Placing objects within the set distance range of the Ultrasound sensor reliably set the "Occupied" flag. Likewise, moving them out of the set range cleared the flag.
- CO2 Alarm Logic: With a set code threshold of 900 ppm, manually emulating high-CO_2 conditions caused the red LED and buzzer to activate on the next loop iteration (~5 s later).
- Network Reliability: Disconnecting Wi-Fi manually for about 30 seconds caused a disconnect, but the ESP8266 rejoined the network and resumed MQTT in under 3 seconds without any resets being required.

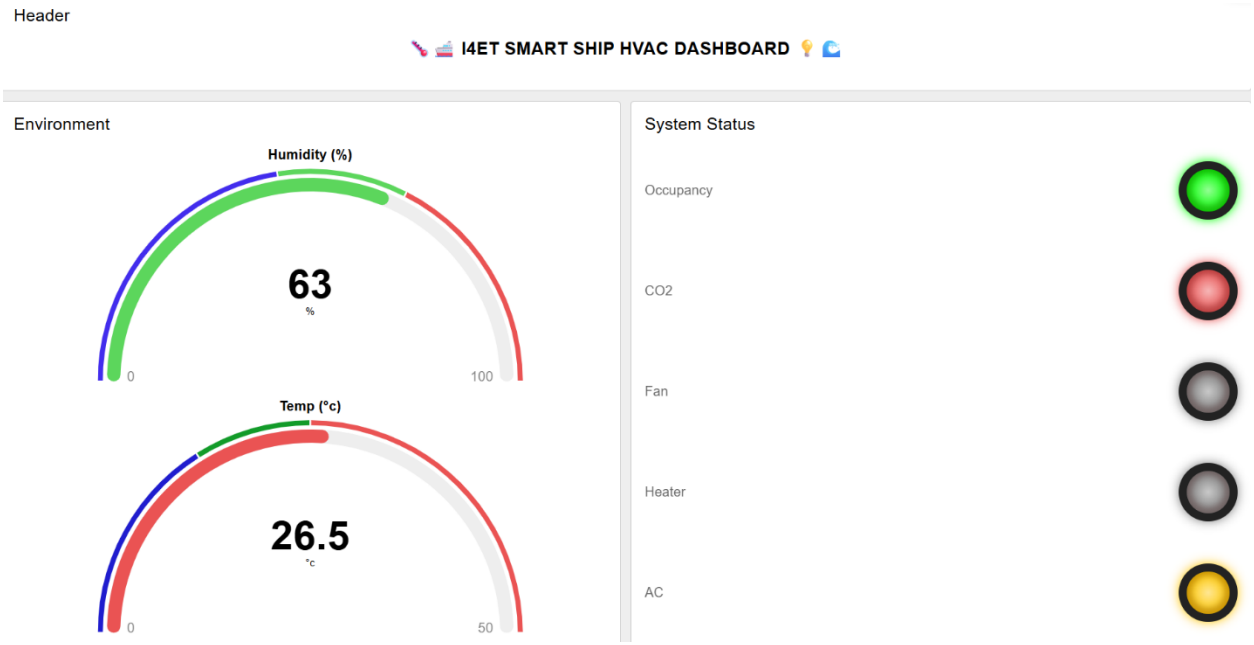

Figure 10: Sample system test results on the dashboard

Overall, this testing confirms the system reacts quickly to our makeshift condition changes and keeps a solid connection for real-time monitoring.

CHAPTER 2: SMART CONTRACT

OVERVIEW

A blockchain-based smart contract system was designed to manage HVAC operations on ships, written in Solidity and deployed on an Ethereum-compatible testnet. It automates HVAC runtime tracking, calculates fuel-related costs, and ensures essential maintenance is requested and executed when needed. By leveraging blockchain's immutability and decentralization, the contract records system usage, flags maintenance when thresholds are crossed, and handles ETH-based fuel and service payments in a transparent, tamper-proof way—eliminating paper logs and guesswork while aligning with Industry 4.0 goals of smarter, more transparent maritime systems.

- ## OBJECTIVES

To guide the design and implementation of our blockchain layer, we defined the following key objectives:

- **AUTOMATE TRACKING**: Record heater, fan and AC runtimes on chain instead of manual entries.
- **ENFORCE MAINTENANCE**: Trigger maintenance requests automatically when cumulative runtime exceeds 1000 hours or temperatures exceed 50°C.
- **STREAMLINE PAYMENTS**: Calculate fuel costs and transfer ETH to the fuel provider each run, plus a fixed maintenance fee when service is due.
- **ENSURE TRANSPARENCY**: Emit events for every major action so anyone can audit system status, runtime, maintenance flags and balances.

- ## KEY PARTICIPANTS AND WORKFLOW

  - KEY PARTICIPANTS:
    - ✓ Ship Engineer (Owner): Deploys the contract and calls functions to activate equipment and run the system.
    - ✓ Fuel Provider: Receives proportional ETH payments for energy consumed based on AC, Heater and Fan hours
    - ✓ Maintenance Provider: Called upon when the contract flags maintenance, resets counters and collects the service fee after performing maintenance.
  - SCENERIO:
    - ✓ The engineer activates the required systems (AC, Heater, Fan)
    - ✓ Contract logs hours and deducts ETH for fuel based on fuel cost

- ✓ If Temp > 50°C or Total running hours > 1000, the contract sets a maintenance flag.
- ✓ Once maintenance is completed, the authorized provider receives the maintenance fee

CONTRACT DESIGN

A built-in feature of smart contracts is their ability to enforce security, automate operations, and ensure transparency. Our smart contract is designed to do just that for our system.

- ACCESS CONTROL:
  - ➢ Only the Owner can initiate operations
  - ➢ Only the maintenance provider can close maintenance requests
- HVAC CONTROL AND TRACKING:
  - ➢ Tracks status of Heater, Fan and AC
  - ➢ Logs total running hours of system
- MAINTENANCE LOGIC:
  - ➢ Flags maintenance if cumulative hours > 1000 or temperature > 50°C.

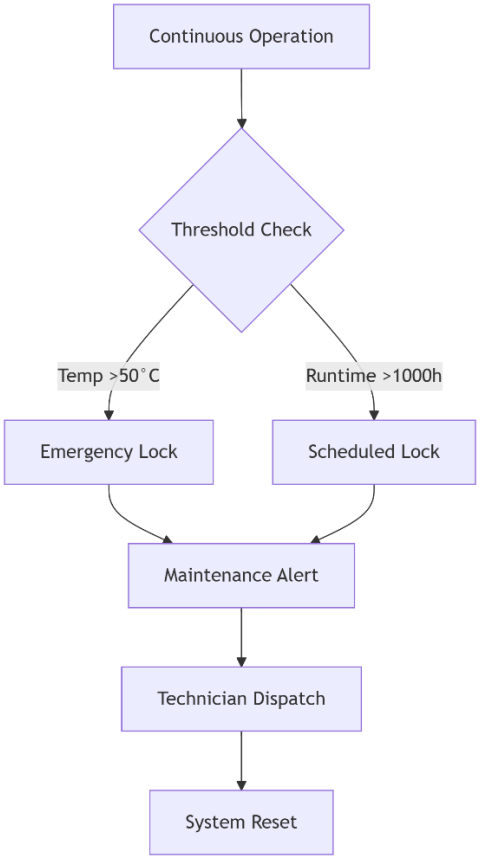

Figure 11: Maintenance Workflow Logic

- AUTOMATED PAYMENT SYSTEM
  - ➢ Calculates and deducts ETH based on usage

- Transfers maintenance fees to the correct provider

- TRANSPARENCY AND EVENTS:
  - Emits events on equipment setup, run completion, maintenance flagging and service completion.
  - Allows anyone to read current states and balances.

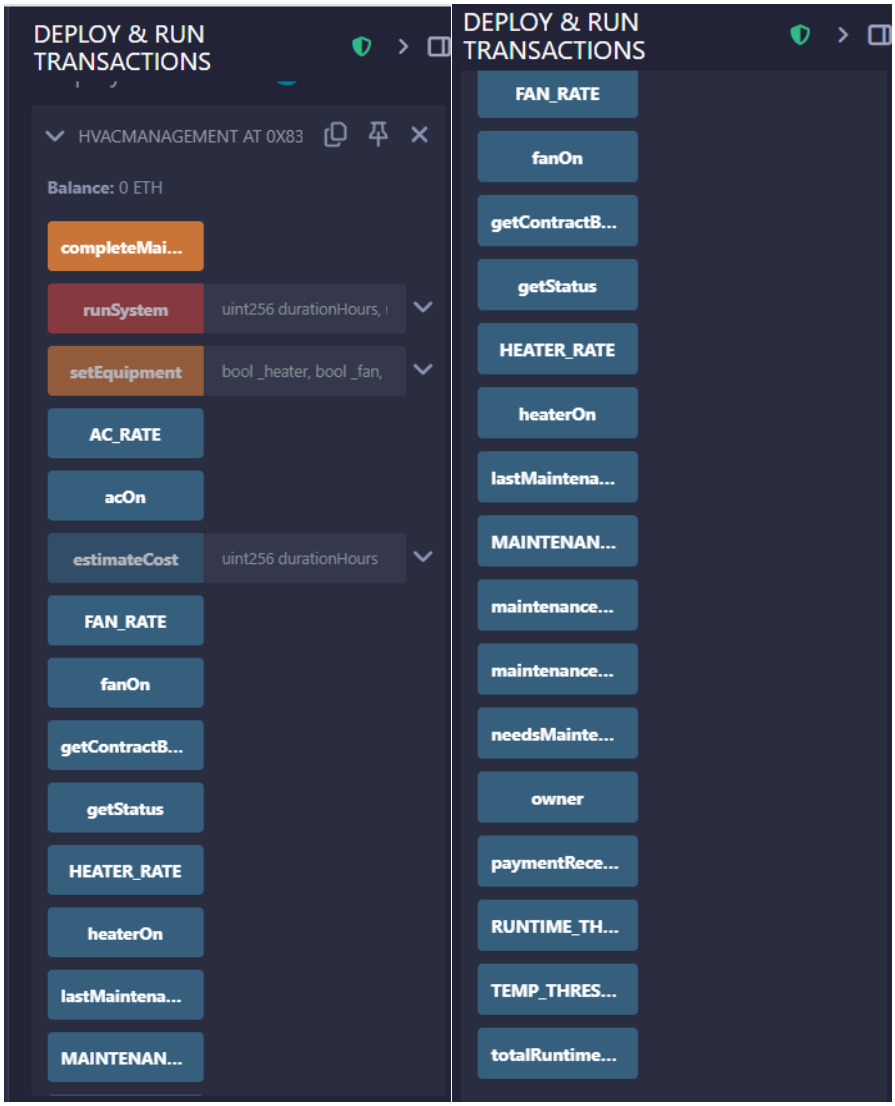

Figure 12: Variables and Functions of Smart Contract

IMPLEMENTATION DETAILS

- RATES AND FEES

The smart contract uses predefined ETH rates for each equipment type to calculate fuel usage costs. These rates are:

- Heater: 0.0002 ETH/hr

- AC: 0.0003 ETH/hr
- Fan: 0.0001 ETH/hr
- Maintenance: 0.01 ETH flat fee

When the system is activated using setEquipment, it records which components are running. During runSystem, the contract calculates the total cost based on the selected components and the number of hours specified. If the system is running under abnormally high temperature (>50°C) or exceeds 1000 cumulative hours, an additional maintenance fee of 0.01 ETH is added to the total as a constant maintenance fee.

TECHNOLOGY STACK

In order to ensure secure, reliable, and transparent smart contract operations, the following development environment and deployment framework was used.

- Programming Language: Solidity ^0.8.0
- Platform: Ethereum-Compatible testnet (Via Remix IDE)

CORE FUNCTIONS

- setEquipment(bool heater, bool fan, bool ac) – configure active components and check for pending maintenance.
- runSystem(uint256 hours, uint256 temp) – log runtime, calculate and transfer fuel ETH, flag maintenance if needed.
- estimateCost(uint256 hours) – view projected fuel cost without altering state.
- completeMaintenance() – payable by maintenance provider to reset runtime and clear flags, transferring the 0.01 ETH fee.

TESTING AND VALIDATION

- ## TEST WALLETS

  To validate role-based access and transaction flows, we defined three test wallets on the Ethereum testnet—each assigned a specific role in the contract:

  - Owner: 0x5B38Da6a701c568545dCfcB03FcB875f56beddC4
  - Fuel Provider: 0xAb8483F64d9C6d1EcF9b849Ae677dD3315835cb2
  - Maintenance Provider: 0x4B20993Bc481177ec7E8f571ceCaE8A9e22C02db

  These wallets let us simulate real-world interactions, confirm permissions, and ensure each function behaves as intended under different user identities.

- ## TEST CASES

  - Deploy: Contract Initializes with zero runtime and no flags with designated provider addresses

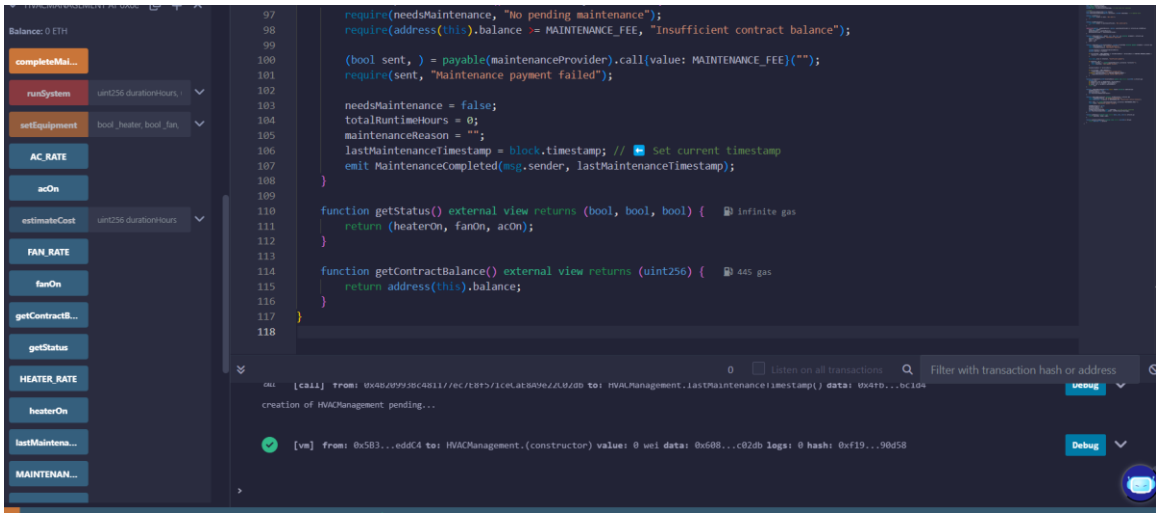

Figure 13: Test Case I

- Activation: setEquipment(true, true, false) turns on Heater and fan.

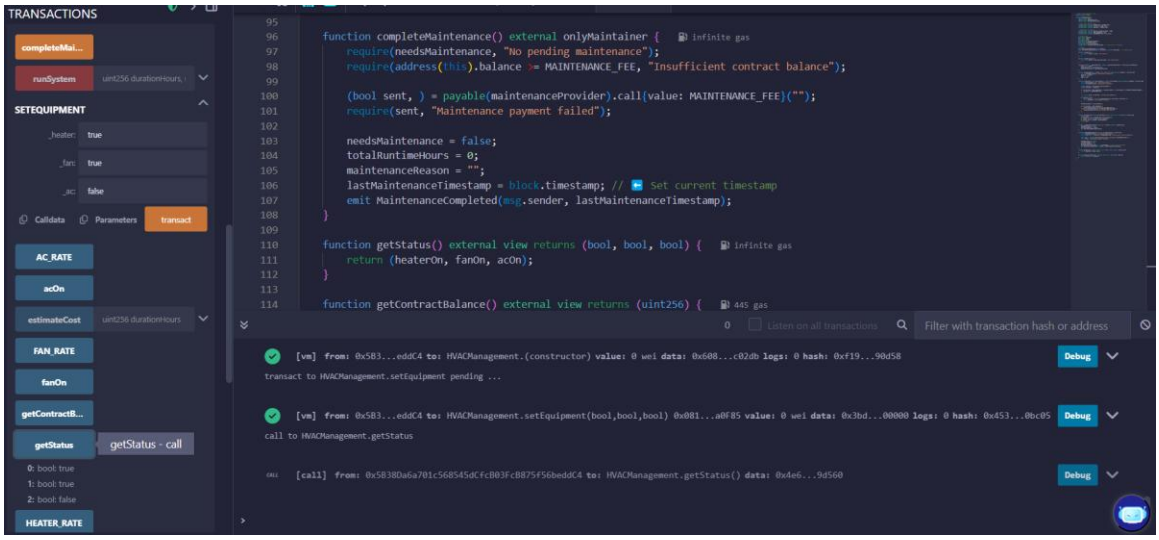

Figure 14: Test Case II

- Normal Run: runSystem(5, 25) deducts correct ETH, increments runtime.

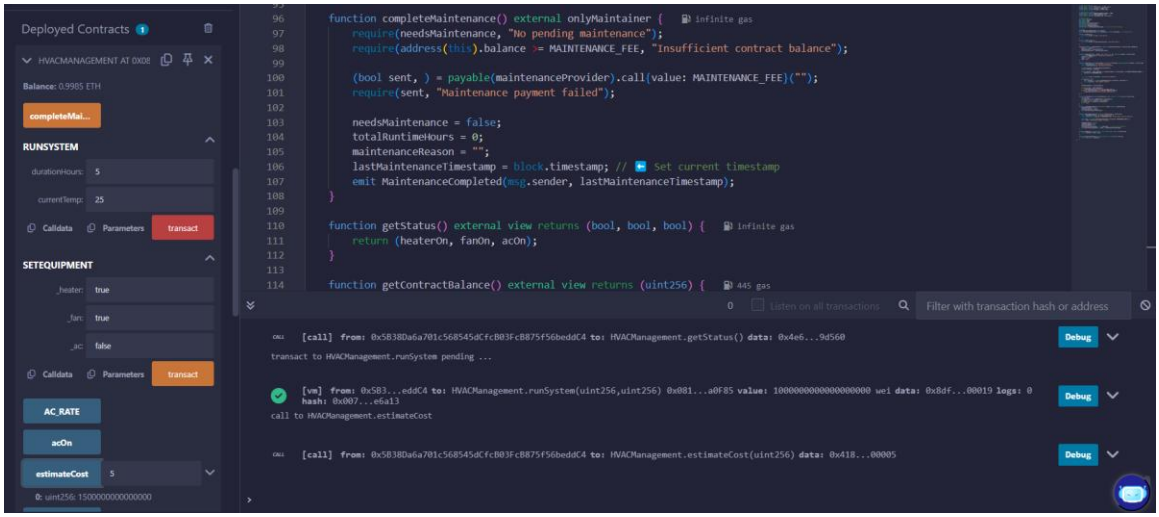

Figure 15: Test Case III

- Maintenance Trigger: runSystem(5, 60) exceeds temperature, flags maintenance, and adds service fee.

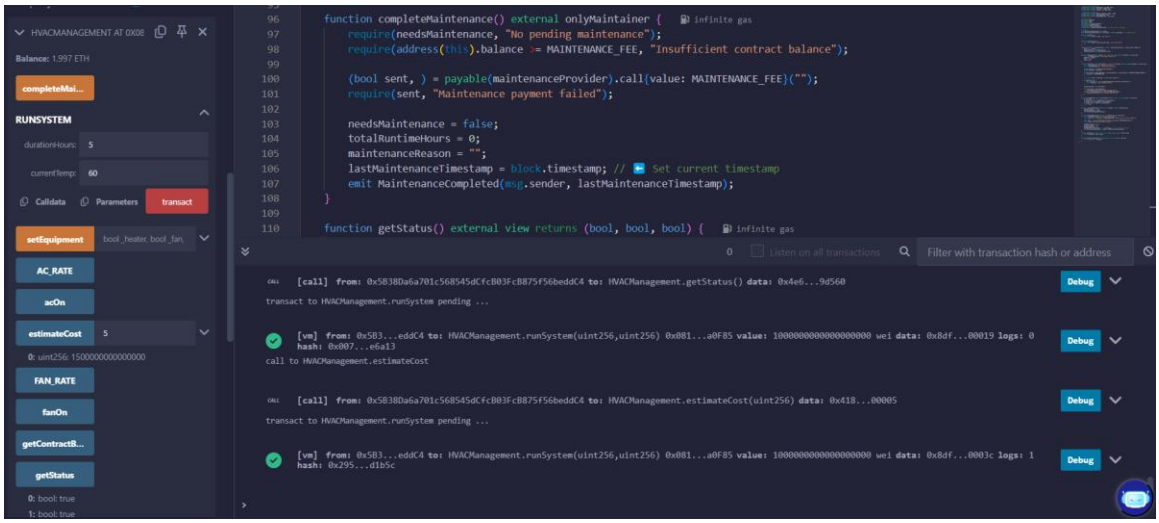

Figure 16: Test Case IV

- Blocked Operation: Further runSystem calls are rejected due to pending maintenance.

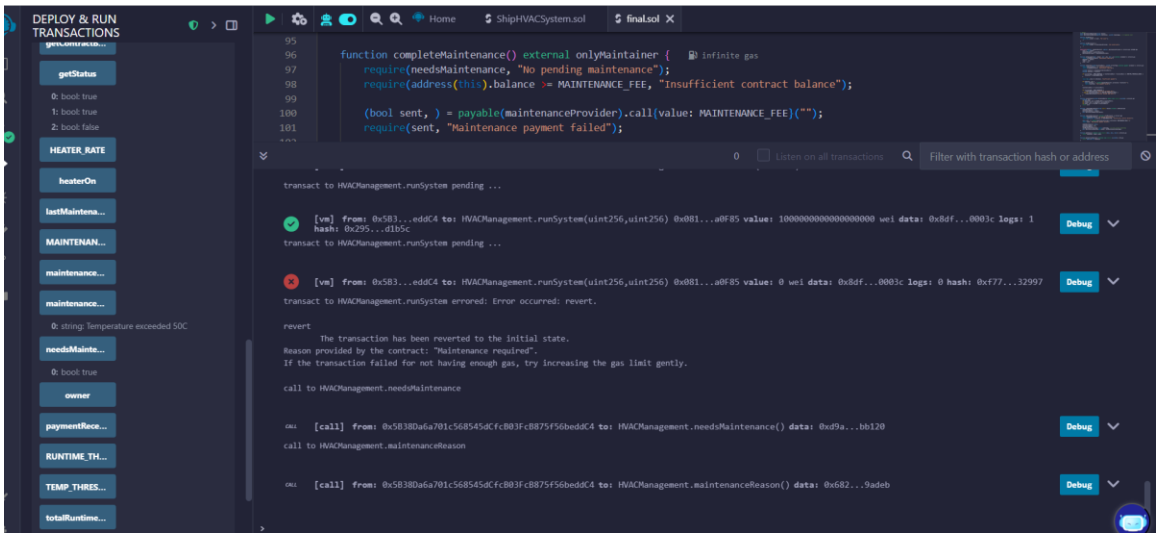

Figure 17: Test Case V

- Service Completion: Maintenance provider calls completeMaintenance to reset runtime, clear flag and receive fee

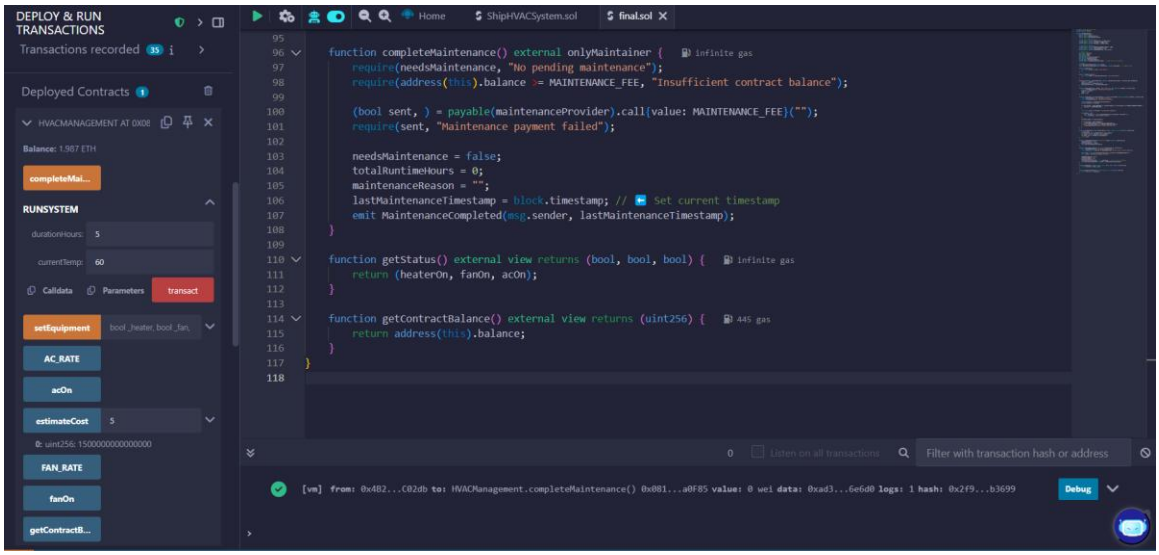

Figure 18: Test Case VI

- Status Checks: getStatus and getContractBalance reflect values after each step

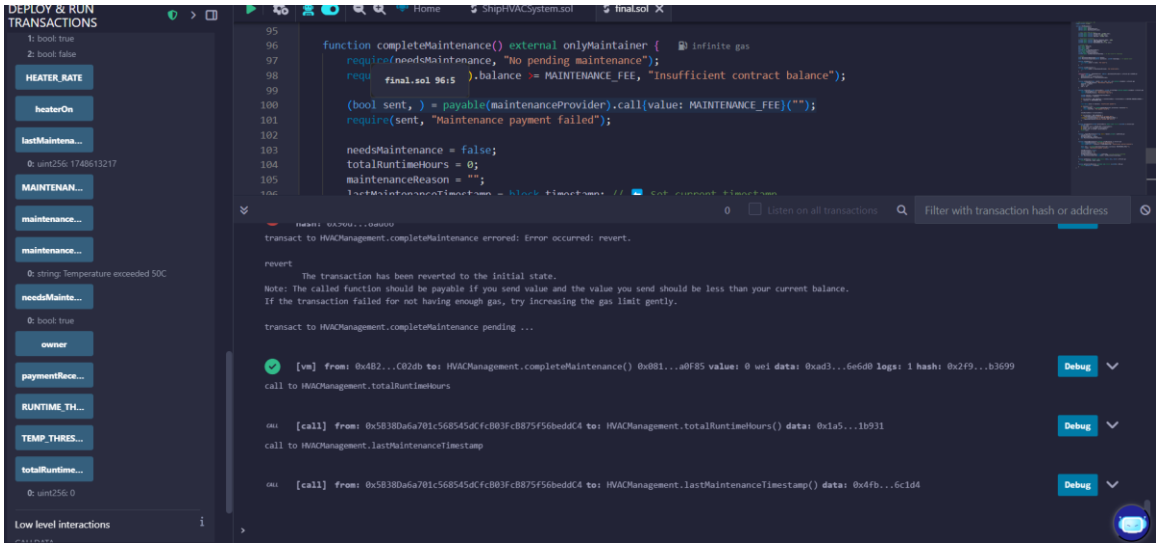

Figure 19: Test Case VII

BENEFITS AND NEXT STEPS

The development and testing of this smart contract has provided strong evidence of its ability to accurately track equipment usage and enforce maintenance based on real operational conditions. By automating ETH-based payments and logging all activities on-chain, the system ensures a high level of trust and efficiency between the ship operator, fuel provider, and maintenance team. This leads to reduced risk of oversight and enhances overall equipment longevity.

The adoption of blockchain for HVAC management demonstrates how Industrial 4.0 principles can be practically implemented in the maritime sector. With features like transparent auditing, secure data storage, and decentralized enforcement, the smart contract simplifies logistics while enabling smarter decision-making. As the maritime industry increasingly turns to digital solutions, this project sets a strong foundation for broader blockchain-based maintenance and operational systems.

CONCLUSION

This project set out to modernize shipboard climate control and maintenance by combining an Intelligent, Cloud-connected HVAC system with a blockchain-backed smart contract. In practice, our Intelligent Cyber-Physical System kept compartment conditions within the 16°C – 25°C and 45% - 65% RH comfort band, silenced unnecessary CO_2 alarms when spaces were empty, and recovered smoothly from brief network outages. On the maintenance side, the Solidity contract reliably logged equipment runtime, automatically issued service requests when hours or temperature limits were hit, and handled ETH payments.

Together, these components close the loop between real-time comfort and long-term upkeep. Crew members benefit from consistently breathable air, gain live visibility and remote monitoring, while ship operators can trust that maintenance happens exactly when it's needed. Looking ahead, swapping in more precise CO_2 sensors, adding predictive control, or migrating to a private maritime blockchain would further strengthen this Industry 4.0 solution.

Ultimately, our dual approach proves that even modest hardware and a few lines of smart-contract code can transform HVAC from a manual chore into an intelligent, self-driving service.