

```

//NODE_MCU.ino

#include <avr/pgmspace.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <ESP8266HTTPClient.h>
#include <TimeLib.h>
#include <FS.h>
#include <LittleFS.h>
#include <ArduinoJson.h>
#include <stdarg.h>
#include <SoftwareSerial.h>

#include "****" //Web Content Header, Censored

// =====
// Network & Notification Constants (PROGMEM)
// =====

PROGMEM const char* ssid          = "****";
PROGMEM const char* password      = "****"; // Censored
PROGMEM const char* NTFY_SERVER   = "ntfy.sh";
PROGMEM const int  NTFY_PORT      = 80; // Standard HTTP port
PROGMEM const char* NTFY_TOPIC    = "****"; // Censored
PROGMEM const int  HttpPort = 80;

// Notification Status Strings (PROGMEM)
PROGMEM const char* P_DEFAULT_PRIORITY = "3";
PROGMEM const char* P_URGENT_PRIORITY  = "5";
PROGMEM const char* P_TAGS_ADDED       = "green_circle";
PROGMEM const char* P_TAGS_GOING_BAD   = "orange_circle";
PROGMEM const char* P_TAGS_EXPIRED     = "red_circle";
// LittleFS File Path (PROGMEM)
PROGMEM const char* P_INVENTORY_FILE = "/inventory.json";

SoftwareSerial Arduino(D6, D5);

const int DAYS_GOING_BAD = 2;
// Switched from WiFiClientSecure to standard WiFiClient
WiFiClient espClient;
ESP8266WebServer server(HttpPort);

// =====
// Sensor Configuration
// =====

// Global State Management (used only for notification logic now)
enum ConditionState { FRESH, GOING_BAD, EXPIRED, STALE_ERROR };
volatile ConditionState currentBoxA1State = FRESH;

// New variable to store CH3 reading
volatile unsigned long lastSensorUpdateTime_A1 = 0;
const long SENSOR_TIMEOUT_MS = 60000;
// Warn if data is older than 60 seconds

// =====
// TEST CONFIGURATION: Faux data injection
#define ENABLE_TEST_INJECTION false
// =====

bool inventoryFileMissingWarned = false;
unsigned long lastExpiryCheckTime = 0;
const long EXPIRY_CHECK_INTERVAL = 30000;

volatile int A1_STATS = 0;

// Function prototypes
void setup_wifi();
void handleRoot();
void handleNotFound();
void setupNTP();
String dateString(time_t t);
time_t dateStringToTime(const char* dateStr);
long daysRemaining(time_t expiryTime);

```

```

void saveInventory(const JsonArray& inventory);
void loadInventory(JsonArray& inventory);
void handleGetInventory();
void handleClearAll();
void handleSerialCommand(const String& dataString);
void handleSensorData(const String& boxID, const String& alcoholStr, const String& ch3Str);
void handleLogItemWeb();
void handleLogItem(const String& boxID, const String& itemName, const String& daysStr);

void checkAndPublishExpiries();
void sendNtfyNotification(const char* title, const char* message, const char* priority, const char* tags);
void printMemoryStats();
void verifyLastSave();

// =====
// String Formatting Helper
// =====

const size_t MAX_NOTIFY_STRING_LEN = 512;
String safeFormat(const char *format, ...) {
    char buffer[MAX_NOTIFY_STRING_LEN];
    va_list args;
    va_start(args, format);
    int result = vsnprintf(buffer, MAX_NOTIFY_STRING_LEN, format, args);
    va_end(args);

    if (result < 0 || result >= MAX_NOTIFY_STRING_LEN) {
        buffer[MAX_NOTIFY_STRING_LEN - 1] = '\0';
    }

    return String(buffer);
}

// =====
// NTP Time Helpers
// =====

void setupNTP()
{
    Serial.println("Configuring NTP client...");
    configTime(5.5 * 3600, 0, "pool.ntp.org", "time.nist.gov"); // IST is UTC+5:30

    Serial.print("Waiting for NTP time synchronization");
    time_t now_check = time(nullptr);
    int attempts = 0;
    while(now_check < 1000000000 && attempts < 30) { // Use a more reasonable threshold
        delay(500);
        now_check = time(nullptr);
        Serial.print(".");
        attempts++;
    }

    if (now_check > 1000000000) {
        Serial.println(" SUCCESS!");
        Serial.printf("Current Unix time: %lu\n", now_check);

        // CRITICAL: Sync TimeLib with system time
        setTime(now_check);

        Serial.printf("Current date: %s\n", dateString(now()).c_str());
        Serial.printf("TimeLib now(): %lu\n", now());
        sendNtfyNotification("Connected to the local time ⌚!", "NTP connection verified!", "", "");
    } else {
        Serial.println(" FAILED!");
        Serial.println("WARNING: NTP sync failed. Time-based features will not work.");
    }
}

String dateString(time_t t) {
    return String(year(t)) + "-" +
        (month(t) < 10 ? "0" : "") + String(month(t)) + "-" +
        (day(t) < 10 ? "0" : "") + String(day(t));
}

```

```

time_t dateStringToTime(const char* dateStr) {
    if (strlen(dateStr) < 10) return 0;

    char yearStr[5];
    strncpy(yearStr, dateStr, 4);
    yearStr[4] = '\0';
    int y = atoi(yearStr);

    char monthStr[3];
    strncpy(monthStr, dateStr + 5, 2);
    monthStr[2] = '\0';
    int m = atoi(monthStr);

    char dayStr[3];
    strncpy(dayStr, dateStr + 8, 2);
    dayStr[2] = '\0';
    int d = atoi(dayStr);

    tmElements_t tm;
    tm.Year = CalendarYrToTm(y);
    tm.Month = m;
    tm.Day = d;
    tm.Hour = 0;
    tm.Minute = 0;
    tm.Second = 0;

    return makeTime(tm);
}

long daysRemaining(time_t expiryTime) {
    time_t nowTime = now();
    tmElements_t tmNow;
    breakTime(nowTime, tmNow);
    tmNow.Hour = 0;
    tmNow.Minute = 0; tmNow.Second = 0;
    time_t midnightToday = makeTime(tmNow);

    return (expiryTime - midnightToday) / 86400L;
}

// =====
// SYSTEM MONITORING
// =====

void printMemoryStats() {
    FSInfo fs_info;
    LittleFS.info(fs_info);

    Serial.println("--- System Memory Stats ---");
    Serial.printf("Free Heap: %u bytes\n", ESP.getFreeHeap());
    Serial.printf("LittleFS Total: %u bytes\n", fs_info.totalBytes);
    Serial.printf("LittleFS Used: %u bytes\n", fs_info.usedBytes);
    Serial.println("-----");
}

// =====
// LittleFS
// =====

const size_t JSON_DOC_SIZE = 4096;

/**
 * Saves the inventory JsonArray to the LittleFS file.
 * @param inventory The JsonArray containing all item objects.
 */
void saveInventory(const JsonArray& inventory) {
    const char* filePath = (const char*)P_INVENTORY_FILE;
    Serial.printf("Attempting to save inventory to %s...\n", filePath);

    // 1. Open the file in write mode ("w") - this overwrites the existing file
    File file = LittleFS.open(filePath, "w");
    if (!file) {

```

```

        Serial.println("ERROR: Failed to open file for writing.");
        return;
    }

    // 2. Serialize and write to the file
    size_t bytesWritten = serializeJson(inventory, file);

    // 3. CRITICAL: Flush buffer before closing to ensure data is written
    file.flush();

    // 4. Close the file - THIS IS CRITICAL FOR PERSISTENCE
    file.close();

    if (bytesWritten > 0) {
        Serial.printf("SUCCESS: Inventory saved. Bytes written: %u\n", bytesWritten);
    } else {
        Serial.println("ERROR: Failed to write JSON data to file (0 bytes written). Doc size issue?");
    }

    // 5. Optional: Verify the save
    verifyLastSave();
    yield();
}

/**
 * Loads the inventory from LittleFS into the provided JsonArray.
 * @param inventory Reference to JsonArray to populate with loaded data.
 */
void loadInventory(JsonArray& inventory) {
    const char* inventoryFilePath = (const char*)P_INVENTORY_FILE;

    File file = LittleFS.open(inventoryFilePath, "r");
    if (!file) {
        if (!inventoryFileMissingWarned) {
            Serial.println("Warning: Inventory file not found. Creating empty inventory.");
            inventoryFileMissingWarned = true;
        }
        return;
    }
    if (inventoryFileMissingWarned) {
        inventoryFileMissingWarned = false;
    }

    DynamicJsonDocument doc(JSON_DOC_SIZE);
    DeserializationError error = deserializeJson(doc, file);
    file.close();

    if (error) {
        Serial.printf("Error: Failed to read inventory file, error: %s\n", error.c_str());
        return;
    }

    // Copy loaded array into the passed reference
    JsonArray loadedArray = doc.as<JsonArray>();
    for (JsonObject item : loadedArray) {
        inventory.add(item);
    }

    Serial.printf("Inventory loaded from file. Items: %d\n", inventory.size());
}

/**
 * Verifies that the last save operation wrote data to disk.
 */
void verifyLastSave() {
    const char* filePath = (const char*)P_INVENTORY_FILE;
    File file = LittleFS.open(filePath, "r");
    if (file) {
        Serial.printf("Verification: File size after save: %u bytes\n", file.size());
        file.close();
    } else {
        Serial.println("Verification: WARNING - Could not open file after save!");
    }
}

```

```

// =====
// NTFY.SH Notification Handler
// =====

void sendNtfyNotification(const char* title, const char* message, const char* priority, const char* tags) {
    HTTPClient http;
    IPAddress ntfy_ip;

    Serial.printf("Resolving host: %s\n", NTFY_SERVER);
    if (WiFi.hostByName(NTFY_SERVER, ntfy_ip) == 0) {
        Serial.println("FATAL: DNS resolution failed for ntfy.sh");
        return;
    }

    String topicPath = String("/") + String(NTFY_TOPIC);
    if (!http.begin(espClient, String(NTFY_SERVER), NTFY_PORT, topicPath)) {
        Serial.println("FATAL: HTTP begin failed.");
        http.end();
        return;
    }

    http.addHeader("X-Title", title);
    http.addHeader("X-Priority", priority);
    http.addHeader("X-Tags", tags);
    http.addHeader("Content-Type", "text/plain");
    int httpResponseCode = http.POST(message);

    if (httpResponseCode > 0) {
        Serial.printf("NTFY.SH POST SUCCESS. Code: %d\n", httpResponseCode);
    } else {
        Serial.printf("NTFY.SH POST FAILED. Error: %s (%d)\n", http.errorToString(httpResponseCode).c_str(),
httpResponseCode);
    }

    http.end();
    yield();
}

// =====
// Serial Command Handlers
// =====

void handleLogItem(const String& boxID, const String& itemName, const String& daysStr) {
    DynamicJsonDocument doc(JSON_DOC_SIZE);
    JsonArray inventory = doc.to<JsonArray>();

    loadInventory(inventory);

    int daysExpiry = daysStr.toInt();
    time_t nowTime = now();
    int currentYear = year(nowTime);

    Serial.printf("DEBUG: Current time: %lu (year: %d)\n", nowTime, currentYear);
    if (currentYear < 2024) {
        Serial.printf("ERROR: NTP time not synchronized! Year is %d (expected 2024+)\n", currentYear);
        Serial.println("Please wait for time sync and try again.");
        return; // Abort the operation
    }
    Serial.println("Time check passed. Proceeding with item logging...");

    time_t expiryTime = nowTime + (time_t)daysExpiry * 24 * 3600;

    JsonObject newItem = inventory.createNestedObject();
    newItem["id"] = boxID;
    newItem["box_id"] = boxID;
    newItem["name"] = itemName;
    newItem["date_logged"] = dateString(nowTime);
    newItem["expiry_date"] = dateString(expiryTime);
    newItem["initial_days"] = daysExpiry;
    newItem["notified_going_bad"] = false;
    newItem["notified_expired"] = false;
    newItem["last_notified_time"] = 0;
}

```

```

saveInventory(inventory);

String title = safeFormat("✓ %s added to Box %s!", itemName.c_str(), boxID.c_str());
String message = safeFormat("Item **%s** (%s) has been logged into the system! Expiry: %d days",
                             itemName.c_str(), boxID.c_str(), daysExpiry);
sendNtfyNotification(title.c_str(), message.c_str(), P_DEFAULT_PRIORITY, P_TAGS_ADDED);

Serial.printf("LOGGED: %s - %s (Initial Shelf Life: %d days)\n", boxID.c_str(), itemName.c_str(), daysExpiry);
}

/**
 * Handles incoming serial sensor data for box A1.
 * Updates global sensor state and triggers an immediate LED update.
 */
void handleSensorData(const String& boxID, const String& alcoholStr, const String& ch3Str) {
    if (boxID != "A1") {
        Serial.printf("Warning: Received sensor data for unknown box %. Ignoring.\n", boxID.c_str());
        return;
    }

    int alcohol = alcoholStr.toInt();
    int ch3 = ch3Str.toInt();
    // Update global volatile state variables
    lastSensorUpdateTime_A1 = millis();
    Serial.printf("SENSOR UPDATE: Box A1 -> Alcohol: %d PPM, CH3: %d PPM\n", alcohol, ch3);
    // Evaluate state and check for notifications immediately
    checkAndPublishExpiries();
}

void handleSensorData(const String& boxID, int Cond)
{
    if (boxID != "A1")
    {
        Serial.printf("Warning: Received sensor data for unknown box %. Ignoring.\n", boxID.c_str());
        return;
    }
}

void handleSerialCommand(const String& dataString) {
    // Serial message format: COMMAND|ARG1|ARG2|ARG3\n
    // For SENSOR: SENSOR|A1|AlcoholPPM|CH3PPM
    // For LOG: LOG|BoxID|ItemName|DaysToExpiry

    int firstPipe = dataString.indexOf('|');
    if (firstPipe == -1) {
        Serial.printf("Error parsing command: Missing '|' separator in: %s\n", dataString.c_str());
        return;
    }

    String command = dataString.substring(0, firstPipe);

    String partsString = dataString.substring(firstPipe + 1);
    int pipe1 = partsString.indexOf('|');
    int pipe2 = partsString.indexOf('|', pipe1 + 1);

    if (pipe1 == -1 || pipe2 == -1) {
        Serial.printf("Error parsing arguments for command %s: %s\n", command.c_str(), partsString.c_str());
        return;
    }

    String boxID = partsString.substring(0, pipe1);
    String arg2 = partsString.substring(pipe1 + 1, pipe2);
    String arg3 = partsString.substring(pipe2 + 1);

    boxID.trim();
    arg2.trim();
    arg3.trim();

    if (command.equalsIgnoreCase("LOG")) {
        handleLogItem(boxID, arg2, arg3);
    } else if (command.equalsIgnoreCase("SENSOR")) {
        handleSensorData(boxID, arg2, arg3);
    } else {

```

```

    Serial.printf("Unknown serial command received: %s\n", command.c_str());
}
}

// =====
// Expiry Checker (Only handles notifications and persistence now)
// =====

void checkAndPublishExpiries() {
    DynamicJsonDocument doc(JSON_DOC_SIZE);
    JsonArray inventory = doc.to<JsonArray>();
    loadInventory(inventory);

    bool inventoryModified = false;
    unsigned long currentTime = millis();
    // Temporary state holder for A1 to determine if a notification needs to fire
    ConditionState newStateA1 = FRESH;
    for (JsonObject item : inventory) {
        yield();

        String boxID = item["box_id"].as<String>();
        String itemName = item["name"].as<String>();

        String message;
        String title;
        const char* priority = P_DEFAULT_PRIORITY;
        const char* tags = "";
        bool statusChanged = false;
        // --- 1. SENSOR OVERRIDE LOGIC (For Box A1) ---
        if (boxID.equalsIgnoreCase("A1"))
        {
            Serial.println("A1 is Active!");
            if(A1_STATS == 2)
            {
                title = safeFormat("%s in Box %s has EXPIRED.", itemName.c_str(), boxID.c_str());
                message = safeFormat("Please clear **%s** (%s) at the earliest!", itemName.c_str(), boxID.c_str());
                priority = P_URGENT_PRIORITY;
                tags = P_TAGS_EXPIRED;

                item["notified_expired"] = true;
                item["notified_going_bad"] = true;
                statusChanged = true;
            }
            else if (A1_STATS == 1)
            {
                if (!item["notified_going_bad"].as<bool>()) {
                    title = safeFormat("⚠ %s in Box %s is going bad!", itemName.c_str(), boxID.c_str());
                    message = safeFormat("Please check **%s** (%s)! (Only 1 day left)", itemName.c_str(), boxID.c_str());
                    priority = P_URGENT_PRIORITY;
                    tags = P_TAGS_GOING_BAD;
                    item["notified_going_bad"] = true;
                    statusChanged = true;
                }
            }
        }
    }
    else //date based logic
    {
        const char* expiryDateStr = item["expiry_date"];
        time_t expiryTime = dateStringToTime(expiryDateStr);

        if (expiryTime == 0) continue;

        long daysLeft = daysRemaining(expiryTime);

        int initialDays = item.containsKey("initial_days") ?
            item["initial_days"].as<int>() : DAYS_GOING_BAD + 1;
        int goingBadThreshold = DAYS_GOING_BAD;

        if (initialDays <= 3) {
            goingBadThreshold = 1;
        }

        // --- Check for Expired Status (Highest Priority) ---
        if (daysLeft <= 0) {

```

```

        if (!item["notified_expired"].as<bool>()) {
            title = safeFormat("%s in Box %s has EXPIRED.", itemName.c_str(), boxID.c_str());
            message = safeFormat("Please clear **%s** (%s) at the earliest!", itemName.c_str(), boxID.c_str());
            priority = P_URGENT_PRIORITY;
            tags = P_TAGS_EXPIRED;

            item["notified_expired"] = true;
            item["notified_going_bad"] = true;
            statusChanged = true;
        }
    }
    // --- Check for Going Bad Status ---
    else if (daysLeft <= goingBadThreshold) {
        if (!item["notified_going_bad"].as<bool>()) {
            title = safeFormat("⚠ %s in Box %s is going bad!", itemName.c_str(), boxID.c_str());
            message = safeFormat("Please check **%s** (%s)! (Only %ld days left)", itemName.c_str(), boxID.c_str(),
daysLeft);
            priority = P_URGENT_PRIORITY;
            tags = P_TAGS_GOING_BAD;
            item["notified_going_bad"] = true;
            statusChanged = true;
        }
    }
}

if (statusChanged) {

    if (!title.isEmpty()) {
        sendNtfyNotification(title.c_str(), message.c_str(), priority, tags);
        item["last_notified_time"] = currentTime;
        inventoryModified = true;
    }
}

if (inventoryModified) {
    saveInventory(inventory);
}
}
// =====
// Web Server Handlers
// =====

void handleGetInventory() {
    DynamicJsonDocument doc(JSON_DOC_SIZE);
    JsonArray inventory = doc.to<JsonArray>();

    loadInventory(inventory);

    String jsonResponse;
    serializeJson(inventory, jsonResponse);
    server.send(200, "application/json", jsonResponse);
    Serial.println("Web client requested inventory data.");
}

void handleClearAll() {
    const char* filePath = (const char*)P_INVENTORY_FILE;
    if (LittleFS.remove(filePath)) {
        Serial.printf("SUCCESS: Removed file: %s. Restarting device...\n", filePath);
        server.send(200, "text/plain", "Inventory cleared successfully. Restarting...");
        delay(500);
        ESP.restart();
    } else {
        Serial.printf("ERROR: Failed to remove file: %s. File may not exist.\n", filePath);
        server.send(500, "text/plain", "Failed to clear inventory file. Check serial monitor for details.");
    }
    printMemoryStats();
}

void handleRoot() {
    server.send_P(200, "text/html", HTML_CODE);
}

void handleNotFound() {

```

```

    String message = "File Not Found\n\n";
    message += "URI: ";
    message += server.uri();
    server.send(404, "text/plain", message);
}

// =====
// WiFi Setup
// =====

void setup_wifi() {
    delay(10);
    Serial.print("Connecting to ");
    Serial.println(ssid);

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }

    Serial.println("\nWiFi connected");
    Serial.print("IP address: ");
    Serial.println(WiFi.localIP());

    // --- NEW: NTFY Notification on Connection ---
    // Use safeFormat to construct the title and message
    String title = safeFormat("✅ Device Connected: %s", WiFi.localIP().toString().c_str());
    String message = safeFormat("That Inventory Manager has successfully connected to network '%s' with IP: %s.", (const
char*)ssid, WiFi.localIP().toString().c_str());

    // Send the notification using default priority and 'added' tags (green circle)
    sendNtfyNotification(title.c_str(), message.c_str(), P_DEFAULT_PRIORITY, P_TAGS_ADDED);
    // --- END NEW: NTFY Notification on Connection ---
}

void handleLogItemWeb() {
    // 1. Check for required parameters
    if (!server.hasArg("box") || !server.hasArg("name") || !server.hasArg("days")) {
        server.send(400, "text/plain", "Missing box, name, or days parameter.");
        Serial.println("Web Log Error: Missing parameters.");
        return;
    }

    // 2. Extract and sanitize parameters
    String boxID = server.arg("box");
    String itemName = server.arg("name");
    String daysStr = server.arg("days");

    // 3. Call the existing logging logic (which saves to LittleFS and sends ntfy notification)
    handleLogItem(boxID, itemName, daysStr);

    // 4. Send success response back to the browser
    server.send(200, "text/plain", "Item logged successfully via web.");
}

// =====
// MAIN SETUP AND LOOP
// =====

void setup()
{
    Serial.begin(115200);
    Arduino.begin(9600);
    // Debug/Upload Serial

    if (!LittleFS.begin()) {
        Serial.println("FATAL: LittleFS Mount Failed.");
    }

    setup_wifi();a
    setupNTP();

```

```

// Give the time library a moment to stabilize
delay(2000);

// Verify sync one more time
time_t finalCheck = now();
Serial.printf("Final time check: %lu (year: %d)\n", finalCheck, year(finalCheck));

if (year(finalCheck) < 2024) {
    Serial.println("WARNING: Time sync verification failed!");
} else {
    Serial.println("Time sync OK - System ready!");
}

printMemoryStats();

server.on("/", handleRoot);
server.on("/inventory", handleGetInventory);
server.on("/clearall", handleClearAll);
server.on("/log", handleLogItemWeb);
server.onNotFound(handleNotFound);
server.begin();
Serial.println("Web server has started....");
}

void loop()
{
    server.handleClient();

    JsonDocument JsonBuff;

    DeserializationError error = deserializeJson(JsonBuff, Arduino);

    if (!error)
    {
        serializeJsonPretty(JsonBuff, Serial);
        A1_STATS = JsonBuff["A1_Stats"].as<int>();
        Serial.println("\n");
        Serial.println(A1_STATS);
        Serial.println("\n");
        JsonBuff.clear();
    }

    // Check Expiries periodically (Rate limited)
    if (millis() - lastExpiryCheckTime >= EXPIRY_CHECK_INTERVAL) {
        checkAndPublishExpiries();
        lastExpiryCheckTime = millis();
    }

    // Periodic memory check (every 60 seconds)
    static unsigned long lastMemCheck = 0;
    if (millis() - lastMemCheck > 60000) {
        printMemoryStats();
        lastMemCheck = millis();
    }

    delay(10);
}

```