```
//ESP8266 Board Code

#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>

#include <time.h> // For NTP time

#define STATUS_LED_PIN 5
#define APP_TIMEOUT_MS 5000
#define AI_MODE_TIMEOUT_MS 1800000

// Heartbeat logic variable
unsigned long lastAppRequestTime = 0;
bool isConnectionLedOn = false;

unsigned long lastAiCommandTime = 0;

unsigned long previousTimeSyncMillis = 0;
const long timeSyncInterval = 300000; // 5 minutes in milliseconds
int lastHourSent = -1; // To avoid sending the same hour repeatedly

// Serial & WiFi Setup
SoftwareSerial SerialDVE(12, 13); // RX, TX
```

```
const char* ssid = "SLT-Fiber-8613";
const char* password = "20030728T";
ESP8266WebServer server(80);

String requestDataFromDue(String command) {
  // Step 1: Clear any old, unread data from the serial buffer to prevent reading stale values.
  while(SerialDVE.available() > 0) {
    SerialDVE.read();
  }

  // Step 2: Send the new command to the Arduino Due.
  SerialDVE.println(command);

  // Step 3: Wait for a response, with a timeout.
  long startTime = millis();
  while (millis() - startTime < 1000) { // 1-second timeout
    if (SerialDVE.available() > 0) {
      String response = SerialDVE.readStringUntil('\n');
      response.trim();
      return response; // Success! Return the fresh data.
    }
  }

  // Step 4: If no data arrives within the timeout, return an error string.
 return "timeout";
}
```

```
void handleStatus() {
  lastAppRequestTime = millis();
  digitalWrite(STATUS_LED_PIN, HIGH);
  Serial.println("App Connected. Status LED ON");
  isConnectionLedOn = true;

  String temp = requestDataFromDue("get_temp");
  String humidity = requestDataFromDue("get_humidity");
  String moisture = requestDataFromDue("get_moisture");
  String ldr1 = requestDataFromDue("get_ldr1");
  String ldr2 = requestDataFromDue("get_ldr2");
  String ldr3 = requestDataFromDue("get_ldr3");
  String servo1 = requestDataFromDue("get_servo_angle1");
  String servo2 = requestDataFromDue("get_servo_angle2");
  String servo3 = requestDataFromDue("get_servo_angle3");

  String water = requestDataFromDue("get_water_level");
  String nutrient = requestDataFromDue("get_nutrient_level");

  StaticJsonDocument<300> jsonDoc;
  jsonDoc["temperature"] = temp.toFloat();
  jsonDoc["humidity"] = humidity.toFloat();
  jsonDoc["moisture"] = moisture.toInt();
  jsonDoc["ldr_value1"] = ldr1.toInt();
  jsonDoc["ldr_value2"] = ldr2.toInt();
```

```
  jsonDoc["ldr_value3"] = ldr3.toInt();
  jsonDoc["cover_angle1"] = servo1.toInt();
  jsonDoc["cover_angle2"] = servo2.toInt();
  jsonDoc["cover_angle3"] = servo3.toInt();

  jsonDoc["water_level"] = water;
  jsonDoc["nutrient_level"] = nutrient;

  jsonDoc["led_status"] = isConnectionLedOn ? "ON" : "OFF";

  String jsonResponse;
  serializeJson(jsonDoc, jsonResponse);

  server.send(200, "application/json", jsonResponse);
}

void handleCheckLight() {
  requestDataFromDue("check_light");
  server.send(200, "application/json", "{\"status\": \"light_check_triggered\"}");
}

void handleResponse() {
  if(server.hasArg("plain") == false) {
    server.send(400, "text/plain", "Bad Request.Body Not Recieved");
    return;
  }
```

```
  String planJson = server.arg("plain");
  Serial.println("Recieved New Plan From App");

  // Forward the new plan to due
  SerialDVE.print("plan:");
  SerialDVE.println(planJson);

  lastAiCommandTime = millis();

  server.send(200, "application/json", "{\"status\":\"PLAN_FORWARDED\"}");
}

void setup() {
  Serial.begin(9600);

  // Arduino DVE communication
  SerialDVE.begin(9600);

  pinMode(STATUS_LED_PIN, OUTPUT);
  digitalWrite(STATUS_LED_PIN, LOW);

  // WiFi Connection
  Serial.println("\n\nESP8266 Booting...");
  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());

server.on("/status", HTTP_GET, handleStatus);
server.begin();
Serial.println("HTTP server started. Connection status LED is active.");
Serial.println("Waiting for app heartbeat...");

configTime(19800, 0, "pool.ntp.org", "time.nist.gov");
Serial.println("Waiting for NTP time sync...");
while (time(nullptr) < 8 * 3600 * 2) {
  delay(500);
  Serial.print(".");
}
Serial.println("\nTime synchronized!");

server.on("/status", HTTP_GET, handleStatus);
server.on("/manual/checklight", HTTP_POST, handleCheckLight);
server.on("/plan/update", HTTP_POST, handleResponse);

server.begin();
lastAiCommandTime = millis();
Serial.println("HTTP server started. Now with LDR/Servo control!");
```

```
}

void loop() {
  server.handleClient();

  // Periodically send the hour to the Due ---
  unsigned long currentMillis = millis();
  if (currentMillis - previousTimeSyncMillis >= timeSyncInterval) {
    previousTimeSyncMillis = currentMillis;

    time_t now = time(nullptr);
    struct tm* timeinfo = localtime(&now);
    int currentHour = timeinfo->tm_hour;

    // Only send the hour if it has changed since the last send
    if (currentHour != lastHourSent) {
      Serial.print("Current hour is ");
      Serial.print(currentHour);
      Serial.println(". Sending to Due...");

      String timeCommand = "time:" + String(currentHour);
      SerialDVE.println(timeCommand);

      lastHourSent = currentHour;
    }
```

```
  }

  if (millis() - lastAiCommandTime > AI_MODE_TIMEOUT_MS) {
    Serial.println("Ai Plan Timeout! Telling Due to use its default brain.");
    SerialDVE.println("USE_DEFAULTS");

    lastAiCommandTime = millis();
  }

  // Status LED Logic
  if (millis() - lastAppRequestTime > APP_TIMEOUT_MS) {
    if (isConnectionLedOn) {
      digitalWrite(STATUS_LED_PIN, LOW);
      isConnectionLedOn = false;
      Serial.println("App connection timed out. Status LED OFF.");
    }
  }
}
```