Department of Electronic & Telecommunication Engineering,
University of Moratuwa, Sri Lanka.

Design Document
Farm Assistor

D.E.U Jayathilaka	210254T
H.M.D.P. Herath	210215C

Submitted in partial fulfillment of the requirements for the module
EN2160 - Electronic Design Realization

1st May 2024

Contents

1 Introduction

The farm assistor, is a reliable tool for measuring the moisture level, temperature level and general fertility level of the soil. Our product can be used across diverse and specialized fields, including green house cultivation and research applications. And this tool helps users make informed decisions about watering practices, soil management, and plant care, resulting in healthy and thriving plants. Therefore, our soil monitoring and management product helps farmers achieve exceptional yields and quality. Our flexible soil moisture monitoring and management systems enable farmers to respond intelligently to changing conditions so they can apply the correct amount of water and fertilizer to their crops at exactly the right time. There is no guesswork, no unnecessary applications, and a lot less waste.

2 Final Product details

2.1 Schematic

The following is the schematic of the product. A hierarchical design has been used. Separate sheets were used for the microcontroller unit, Sensor inputs and their relevent ICs, Power unit and data logger unit.

The circuitry is designed in such a manner that i can read the soil parameters from the connected sensors and transmit the values to the Micro SD card in data logger via the microcontroller. The saved data is transmitted to the data base by ESP 01 Wi-Fi module. The circuit promotes power saving by having a data logging unit so that data can be transmitted as a bulk at adequate intervals. This saves power because Wi-Fi module is the high power consumers in the circuit.

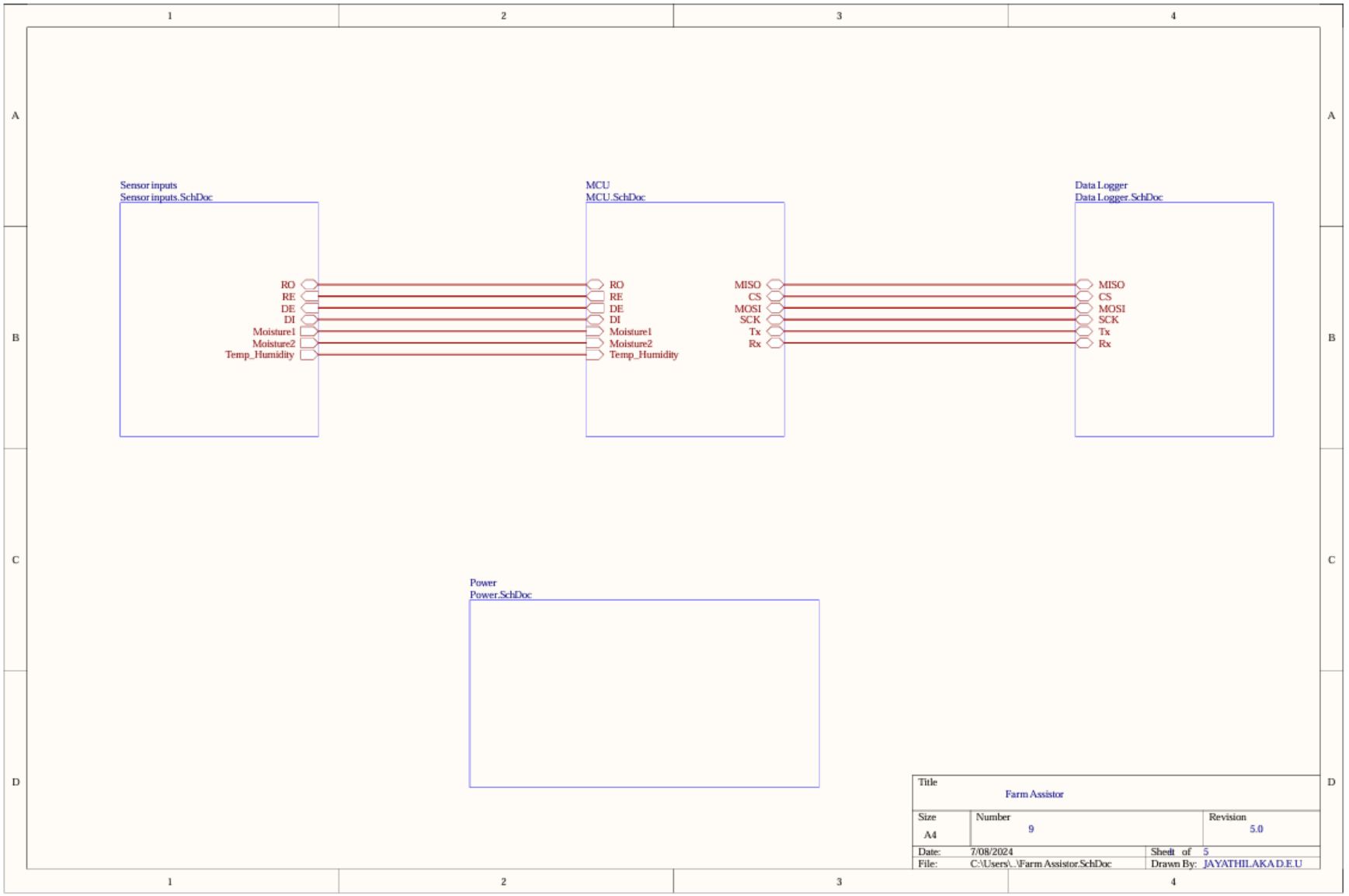


Figure 1: Block Diagram

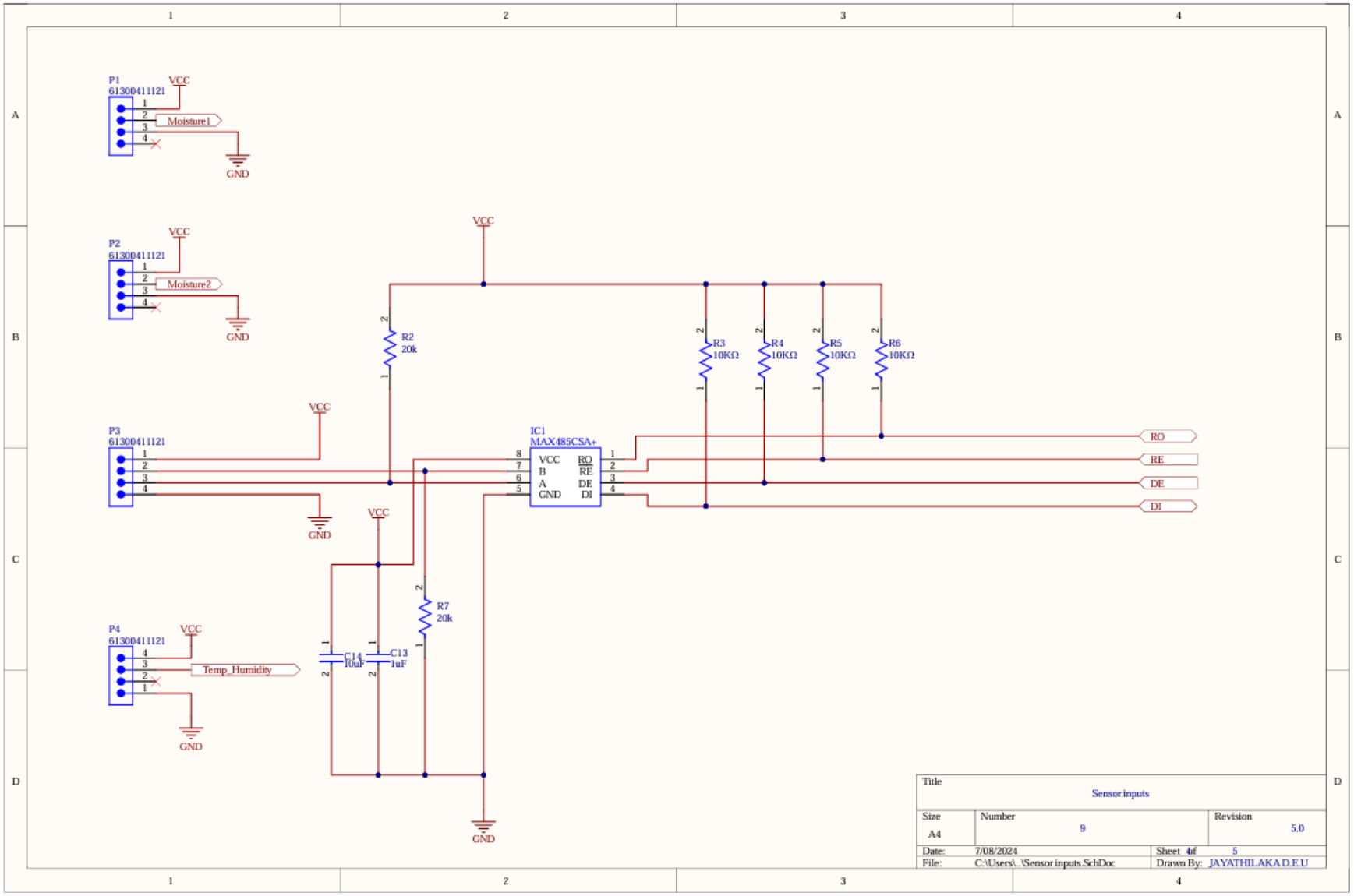


Figure 2: Sensor inputs

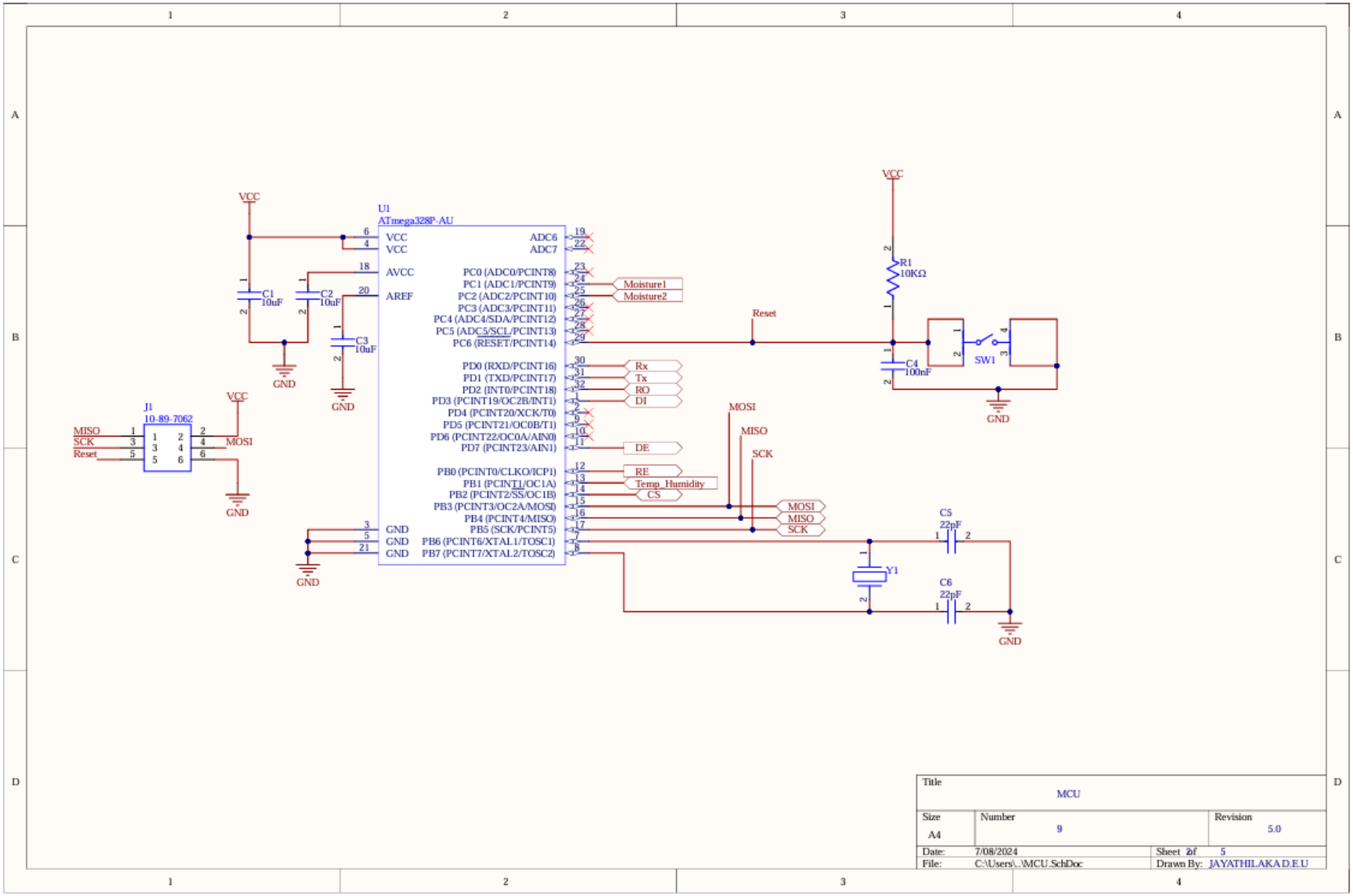


Figure 3: Microcontroller Unit

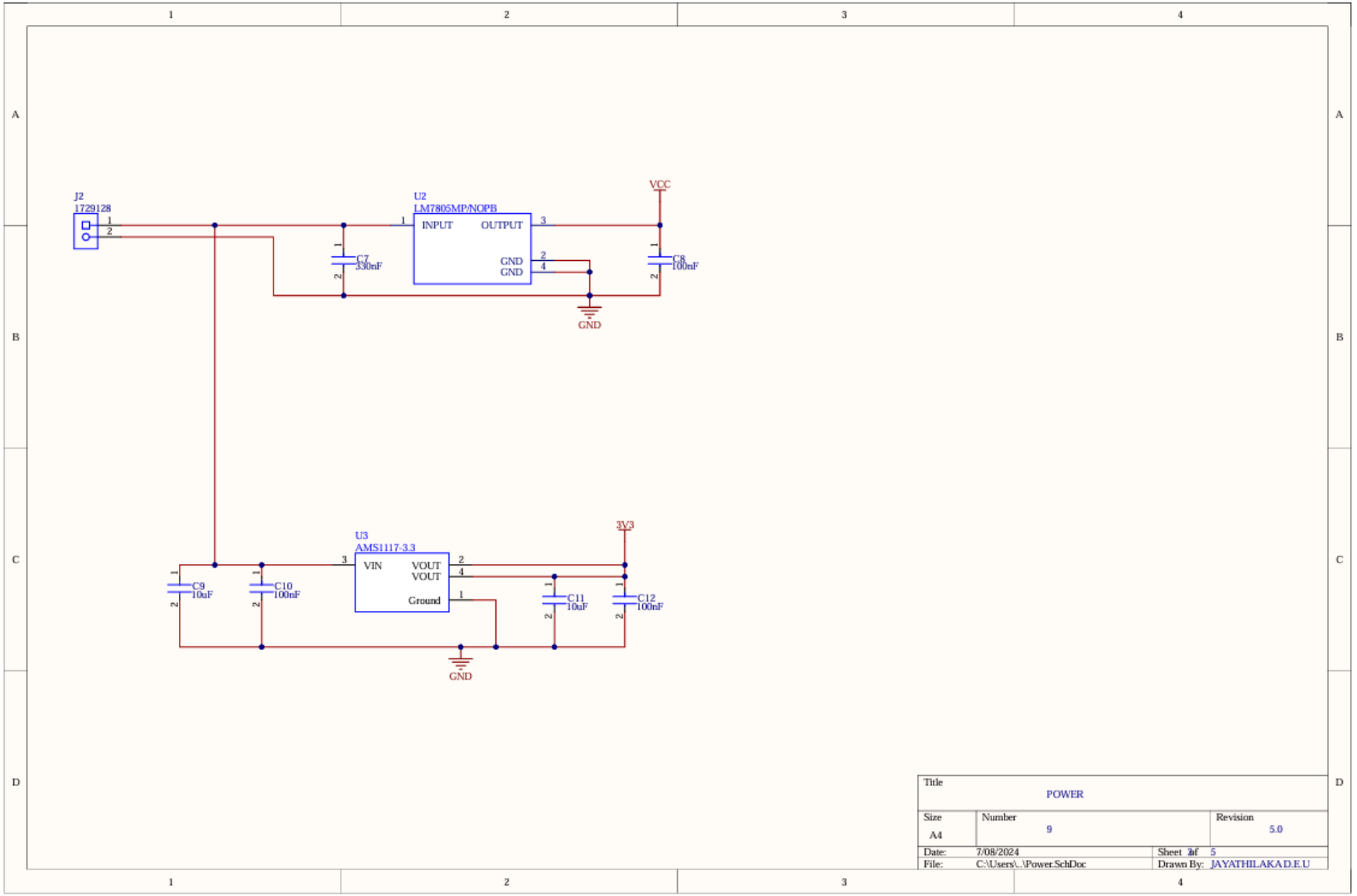


Figure 4: Power Circuit

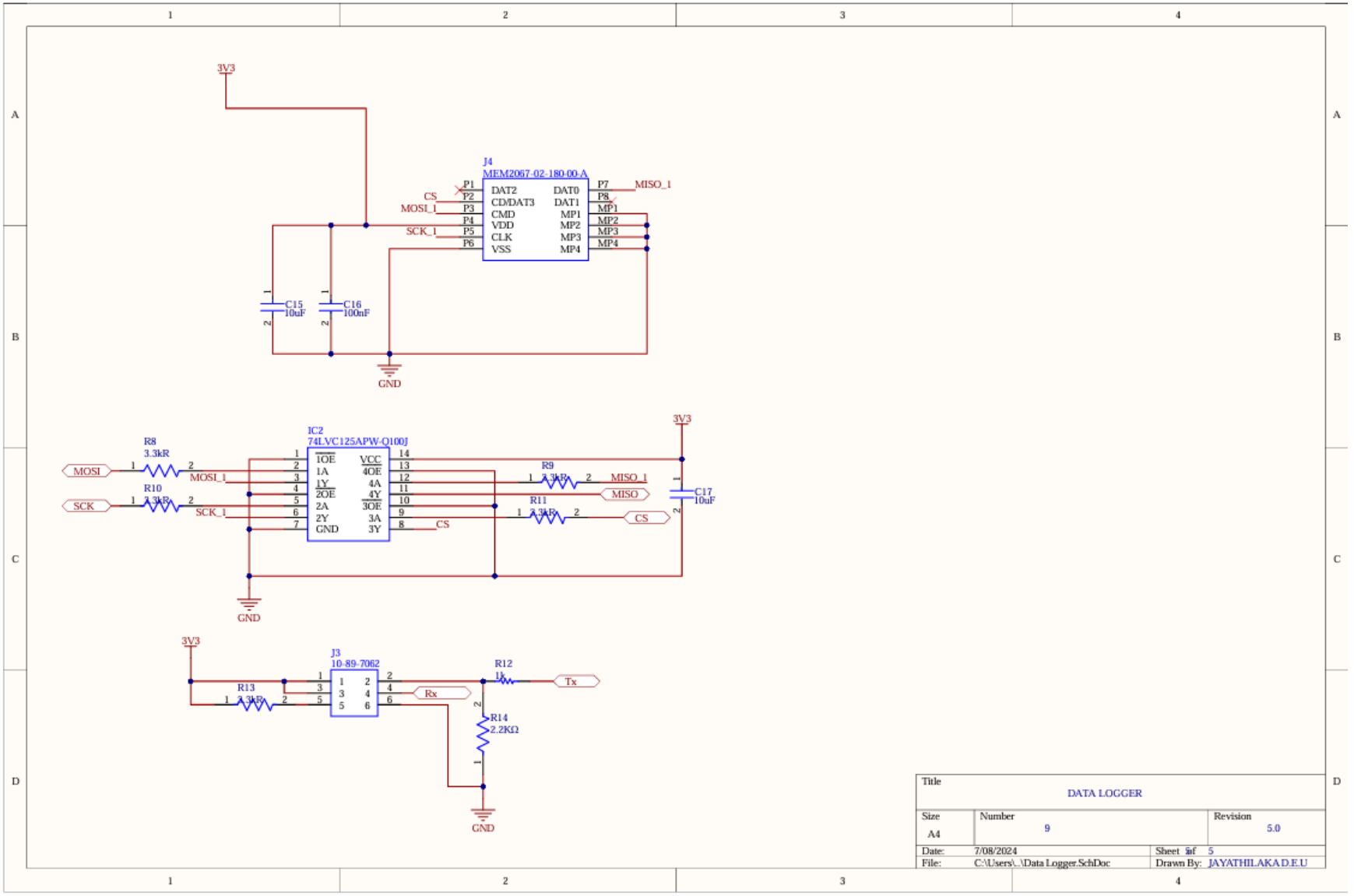


Figure 5: Data Logger

3 Printed Circuit Board

The following is the printed circuit board of the product. It is designed to be able to be manufactured by JLC PCB in China.

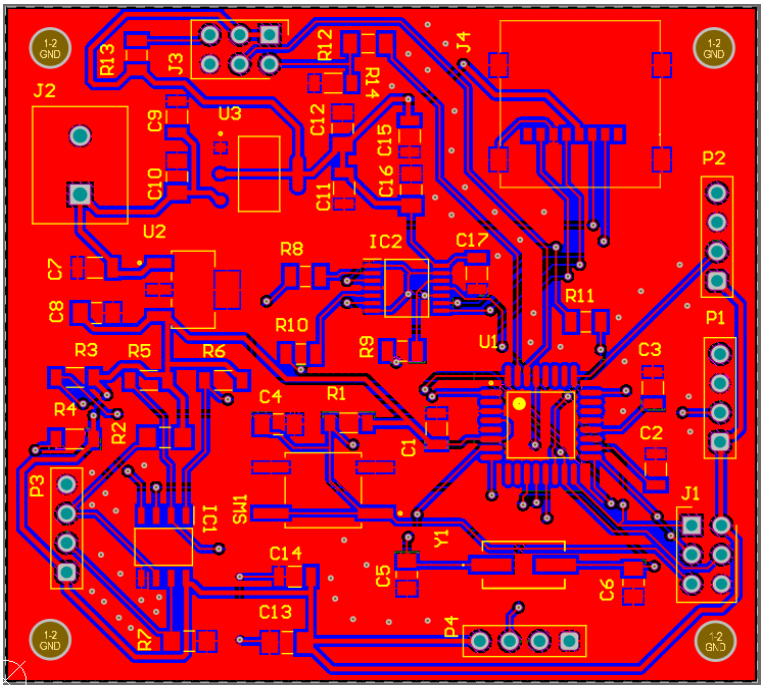


Figure 6: Top layer

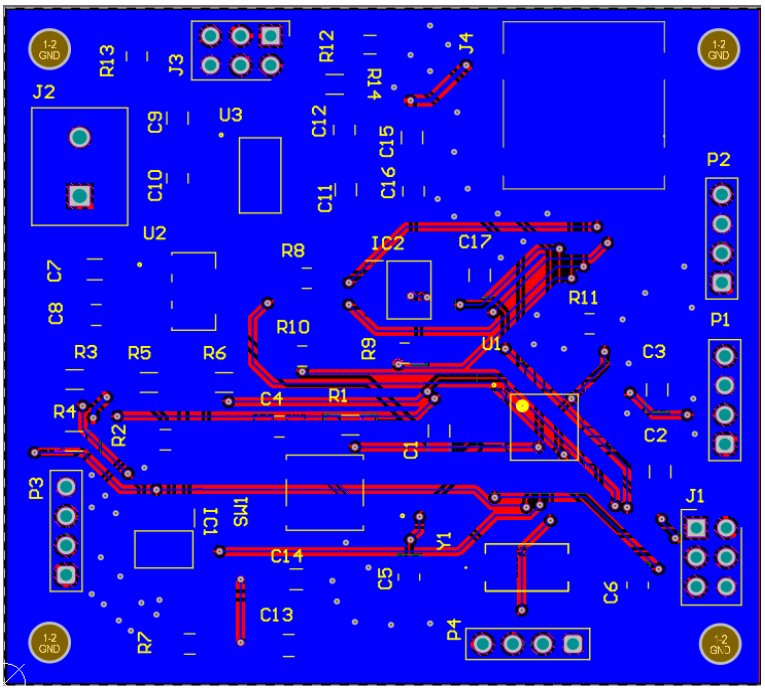


Figure 7: Bottom layer

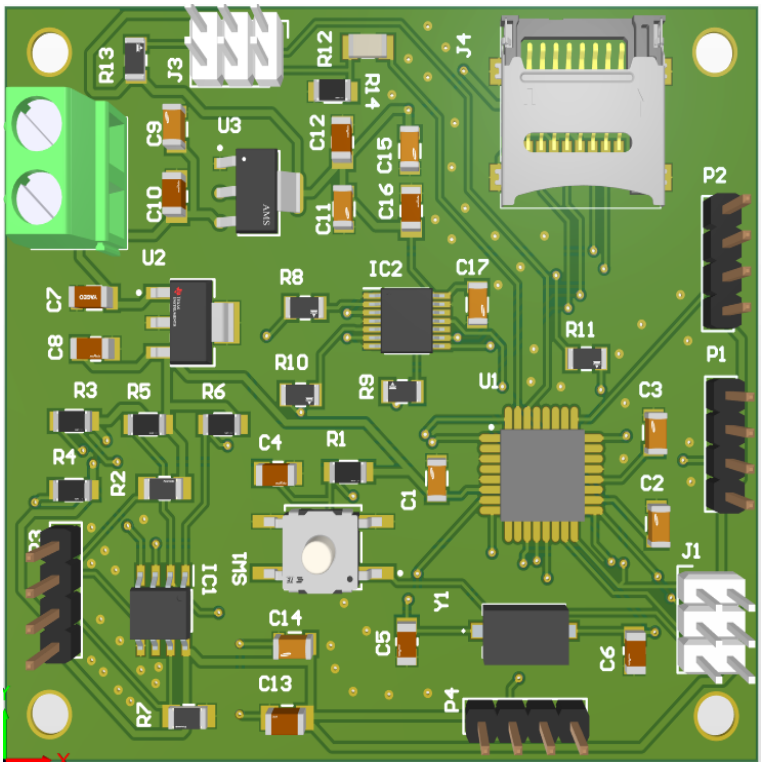


Figure 8: 3D view

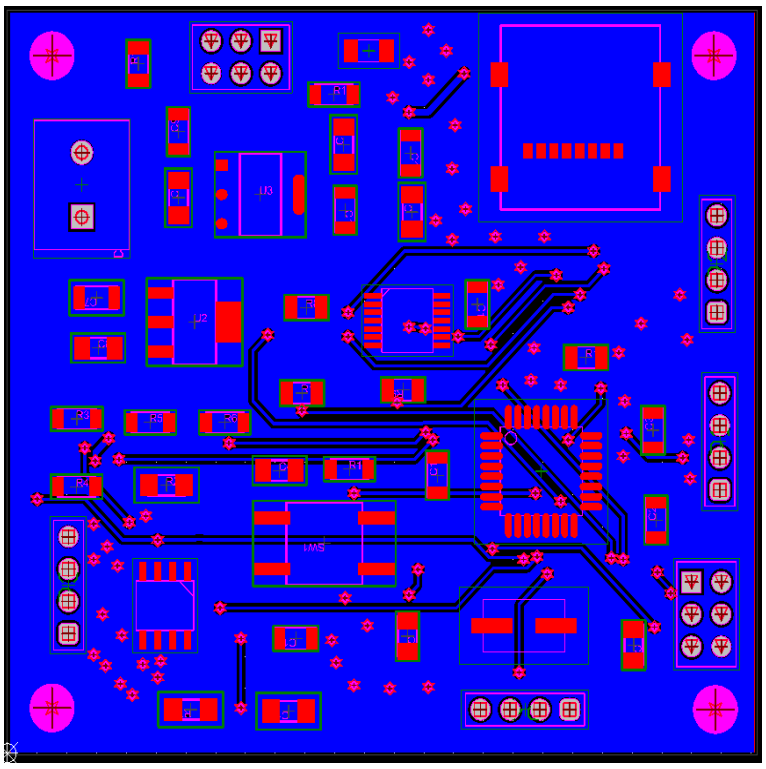


Figure 9: Gerber file

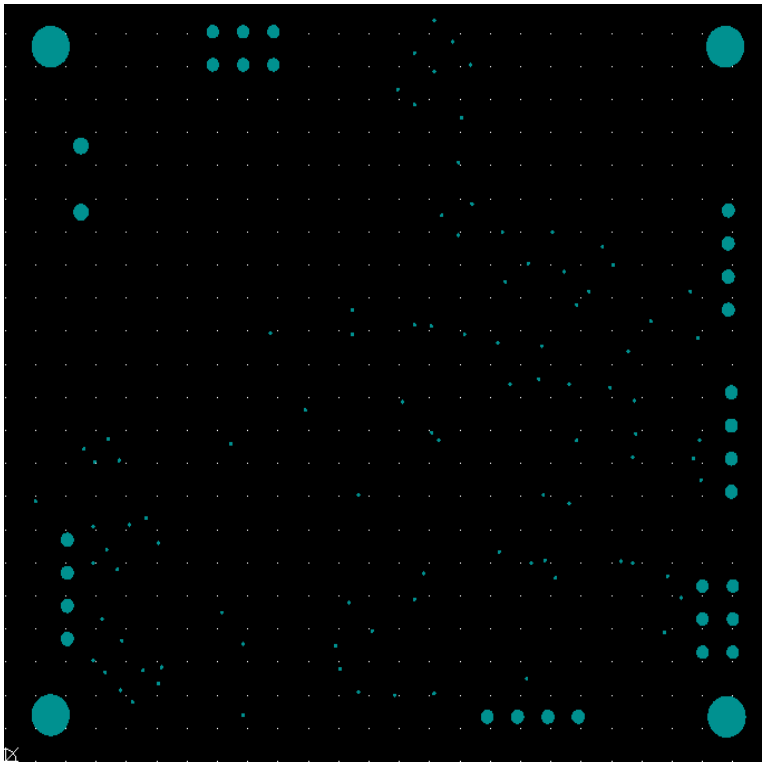


Figure 10: NC Drill file

4 Photographs of the PCB

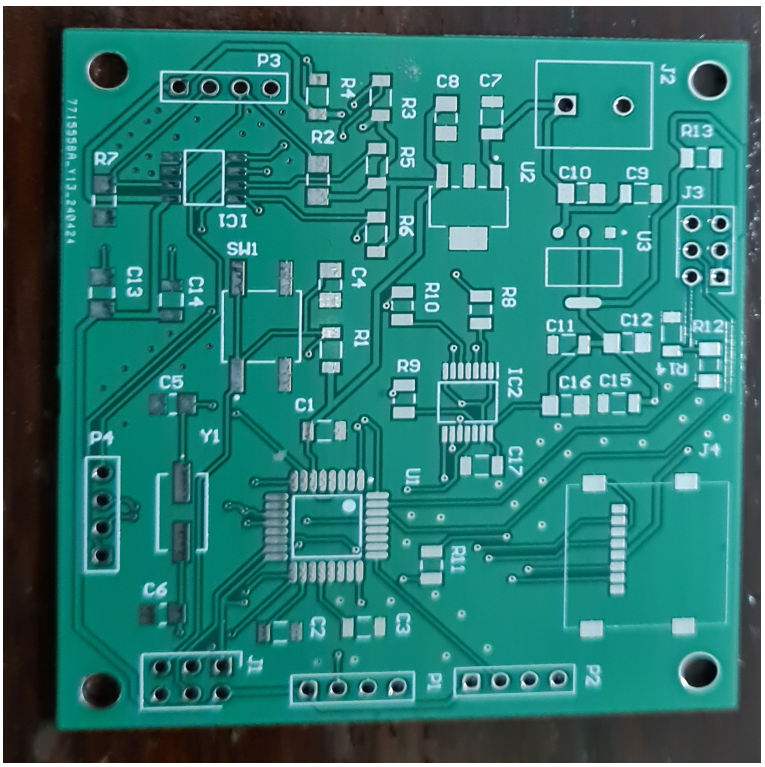


Figure 11: Front view of bare PCB

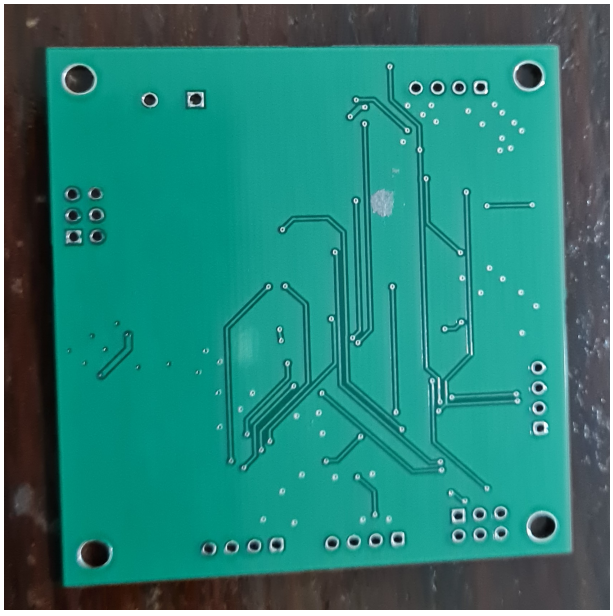


Figure 12: Back view of bare PCB

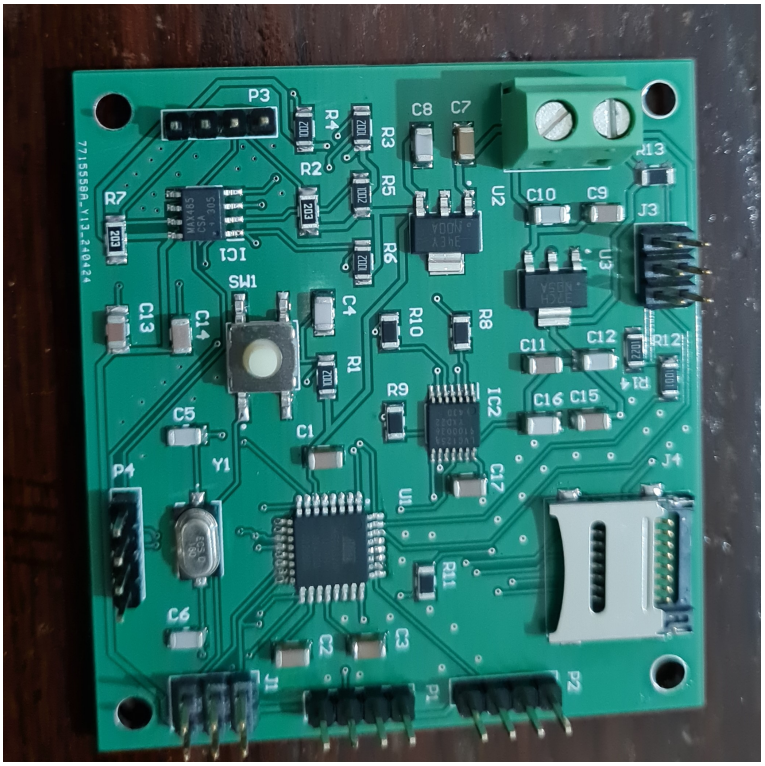


Figure 13: Front view of soldered PCB

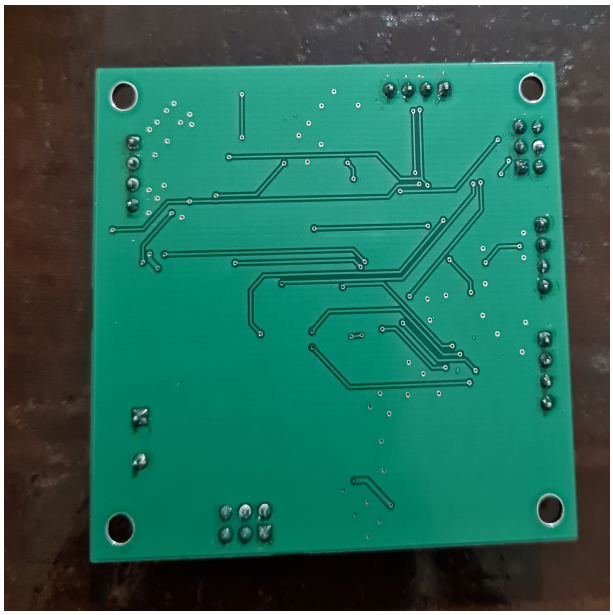


Figure 14: Back view of soldered PCB

5 Enclosure

The enclosure will be designed to be portable, lightweight, and compact with a smarter look. The manufacturing materials for the enclosure will be selected considering the durability and rough use. In this way, the enclosure will be designed to be aesthetically pleasing and easy to assemble.

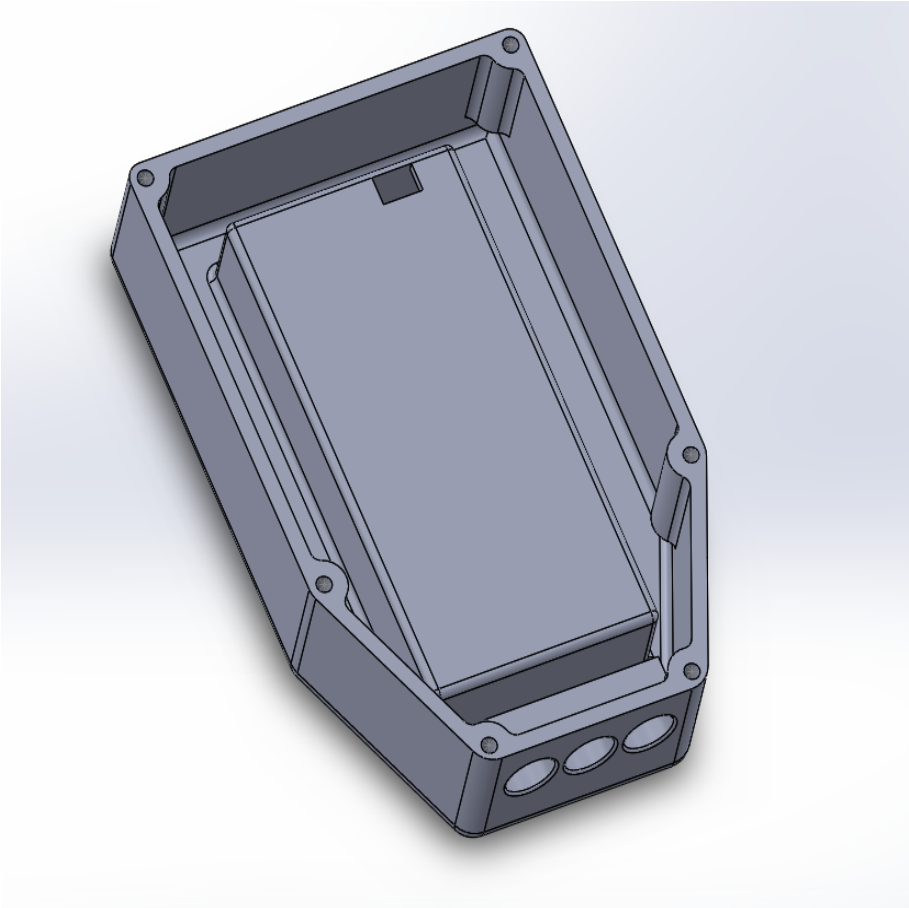


Figure 15: Top lid inside

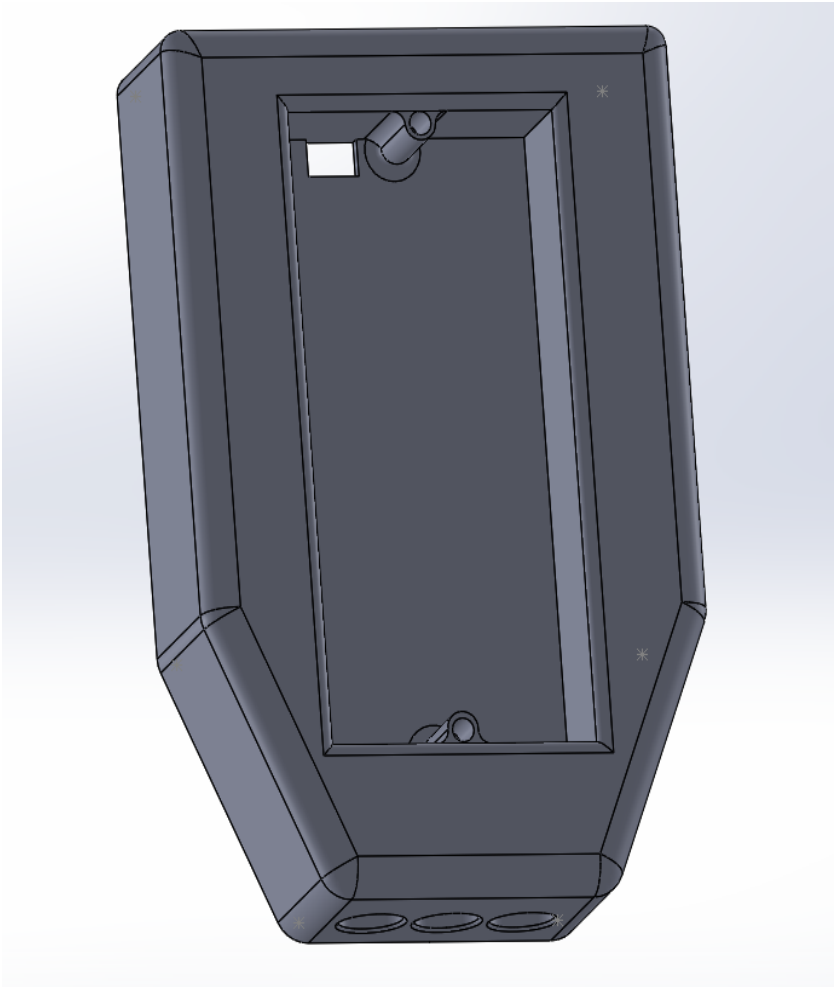


Figure 16: Top lid

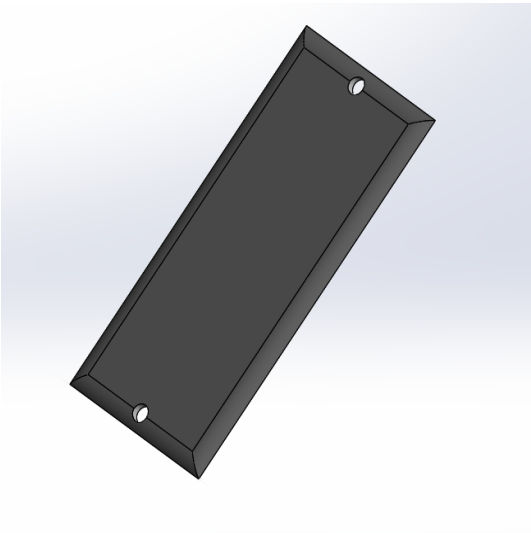


Figure 17: Battery lid

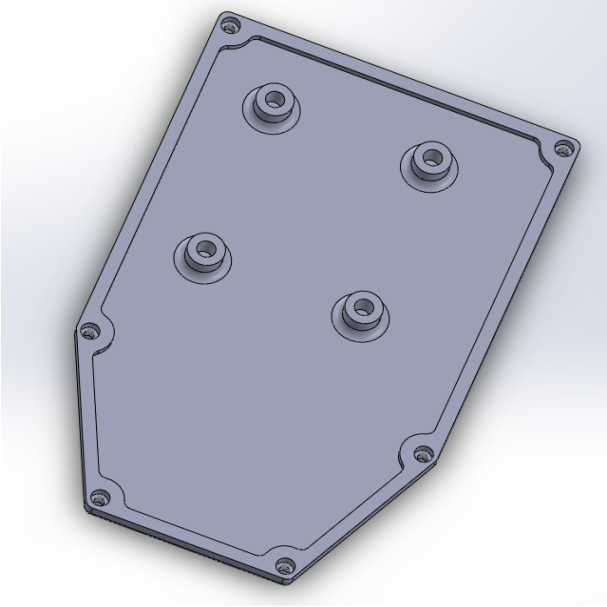
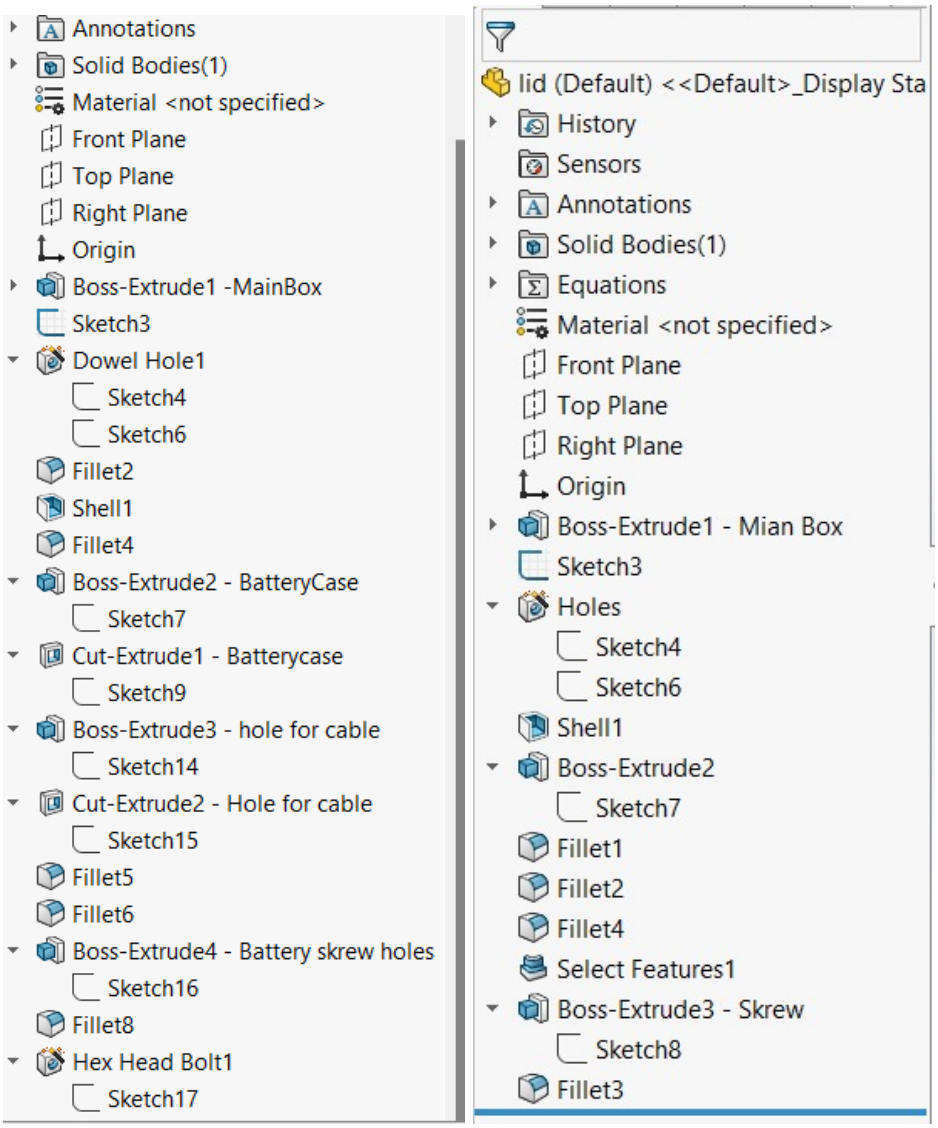


Figure 18: Bottom lid

Figure 19: After assembly



(a) (b)

Figure 20: Model trees

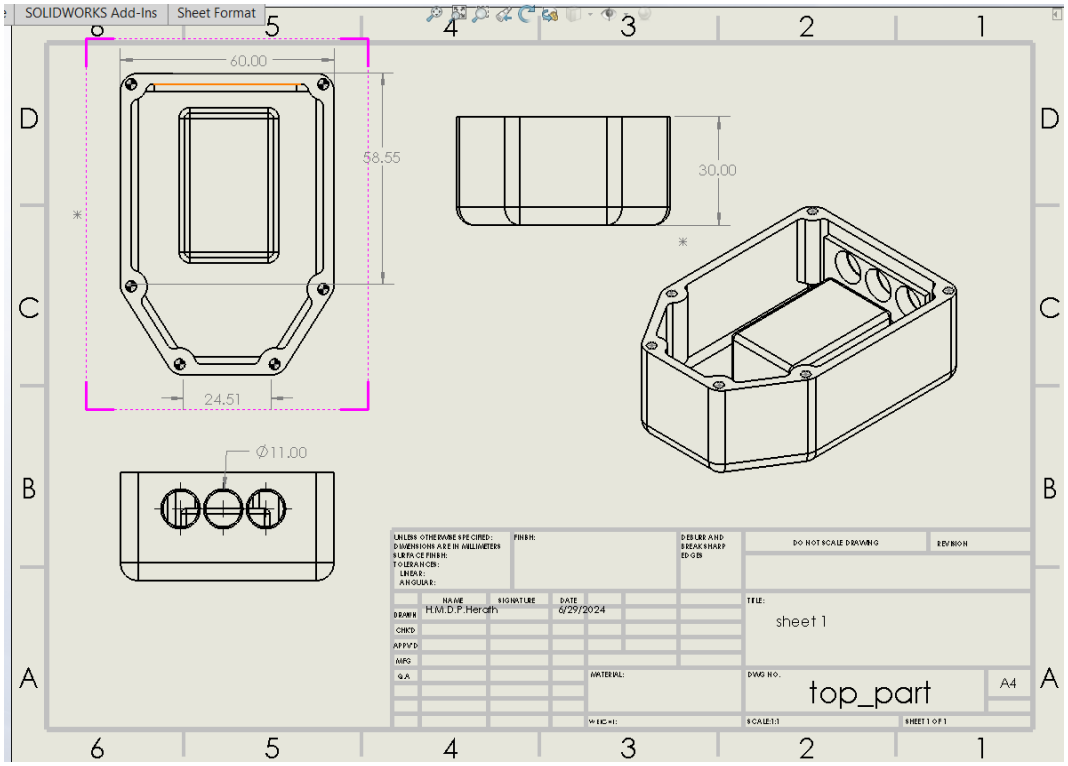


Figure 21

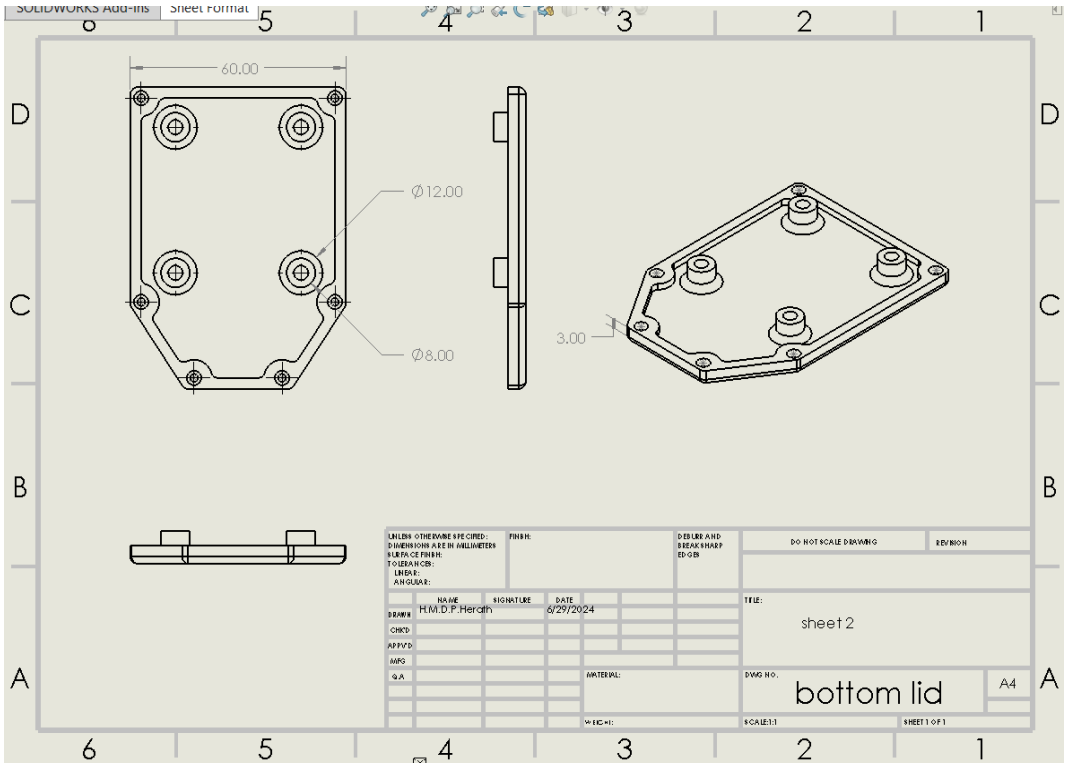


Figure 22

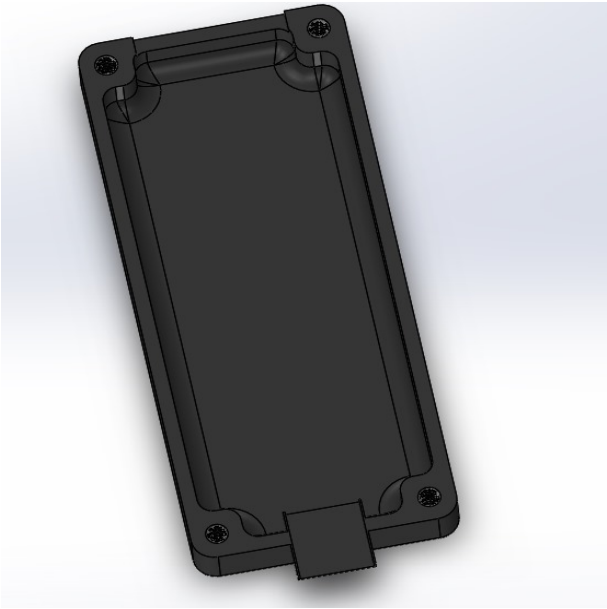


Figure 23: Sensor top

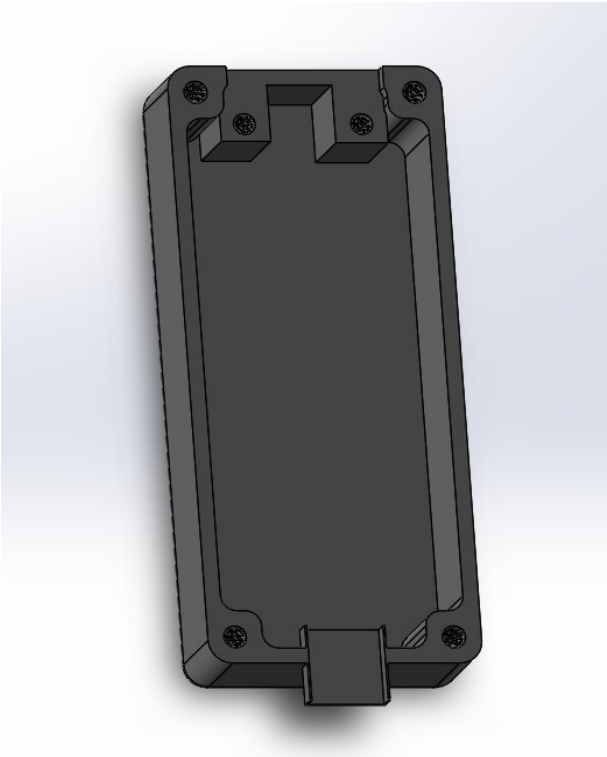


Figure 24: Sensor bottom

Figure 25: Assembly

Figure 26: Overall view of product

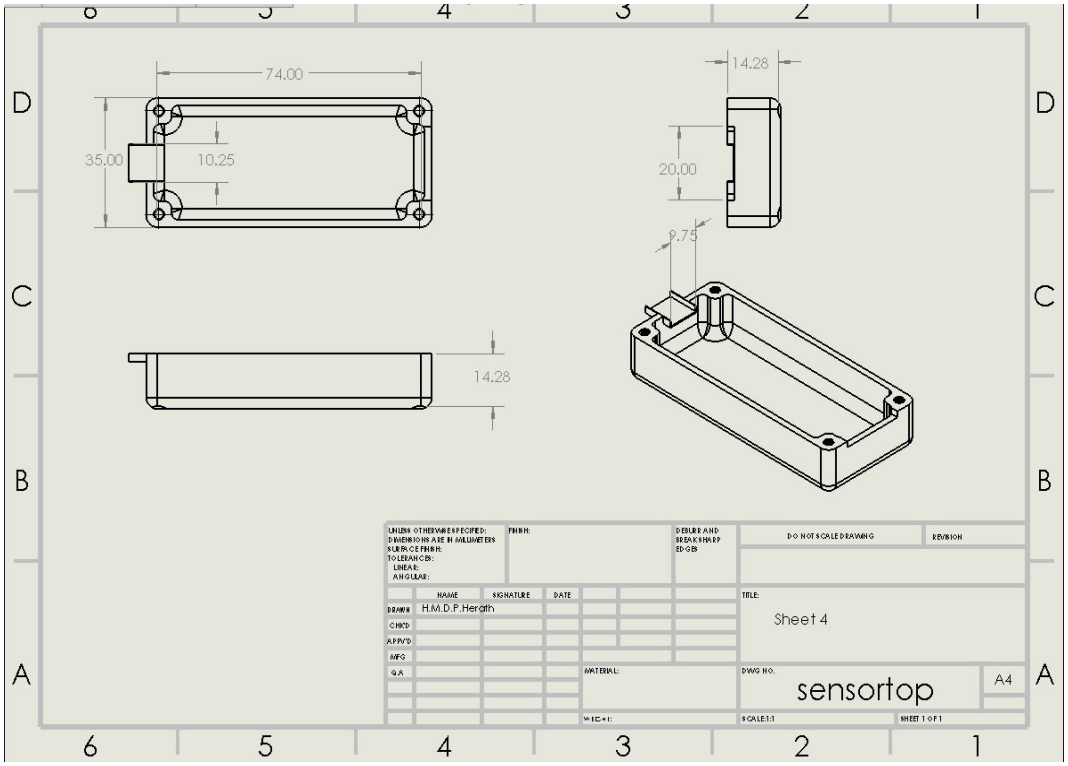


Figure 27: Sensor top

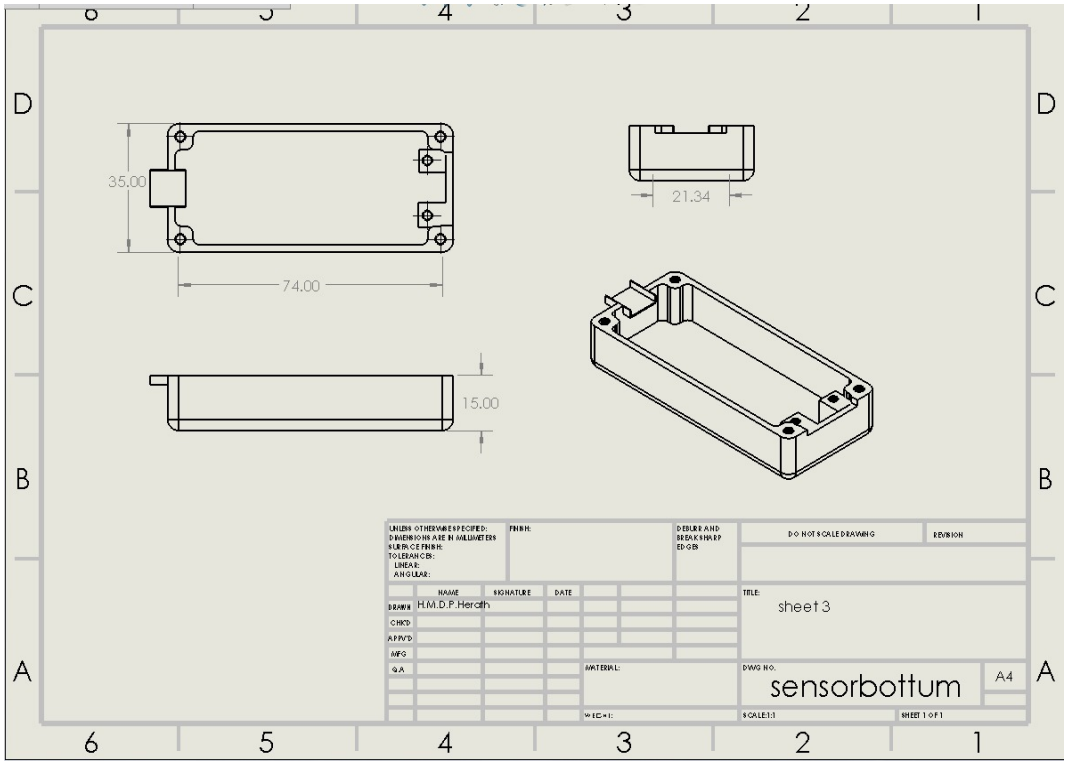
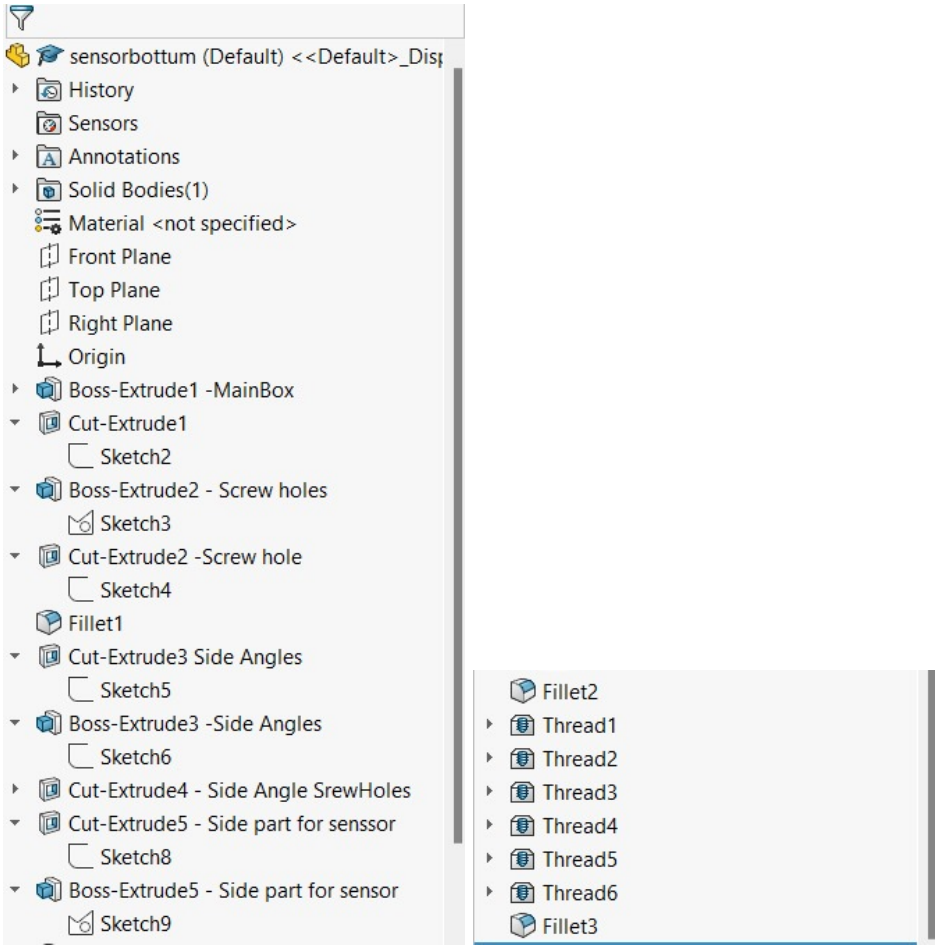


Figure 28: Sensor bottom



(a) (b)

Figure 29: Model trees of sensor bottom part

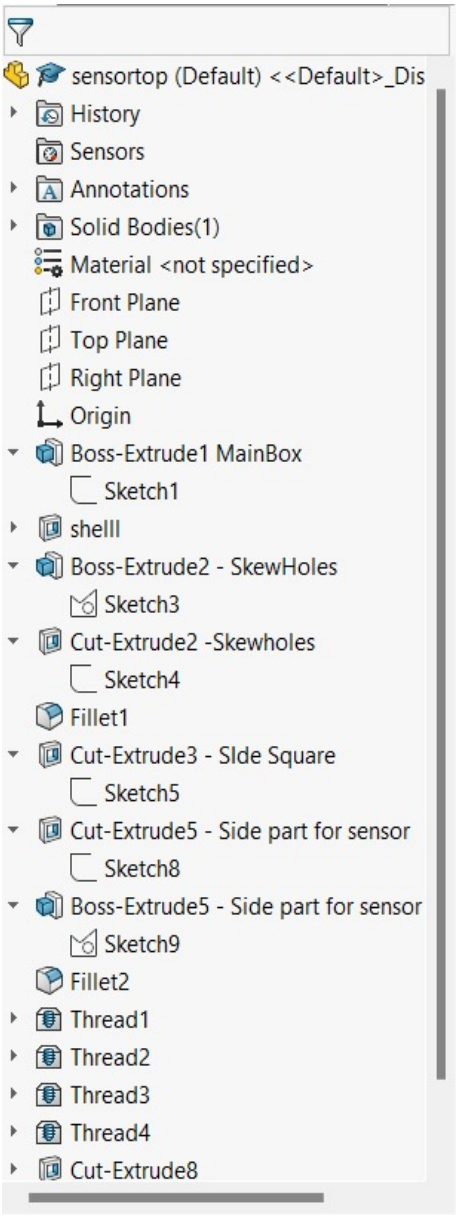


Figure 30: Sensor top model tree

The following is the modified enclosure design according to third conceptual design.

6 Photographs of Physicallly Built Enclosre

Figure 31: Top Part - Isometric View

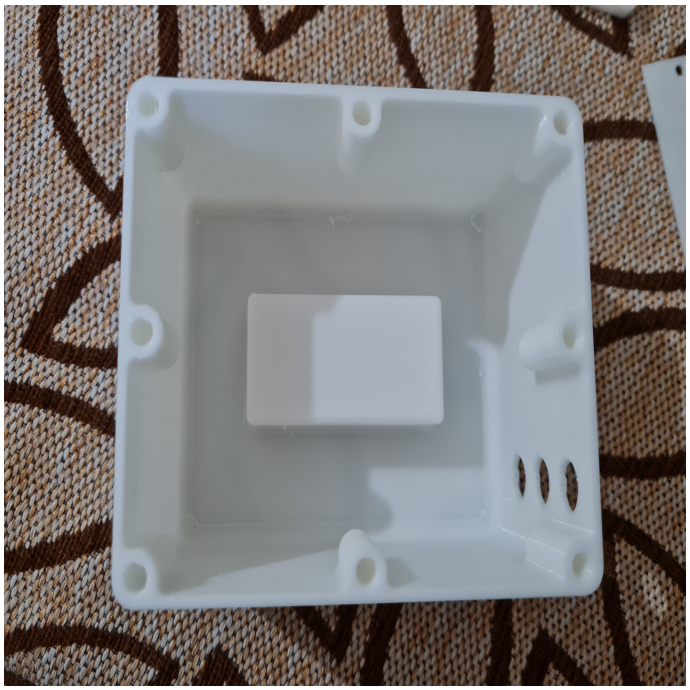


Figure 32: Top Part - Bottom View

Figure 33: Bottom lid

Figure 34: Printed enclosure

This physically built enclosure is from the previous solidworks design. The CAD design was changed after a discussion with the lecturer to align with the conceptual design chosen.

7 Detailed Programming Information

C Code for The Micro-controller

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <avr/wdt.h>
#include <util/delay.h>
#include <string.h>
#include <stdbool.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <math.h>

#define DHT_PIN PB5
#define CS_PIN PB0
#define MOISTURE_SENSOR1 PA0
#define MOISTURE_SENSOR2 PA1
#define RX_PIN PD0
#define TX_PIN PD1
#define RE PB1
#define DE PB2

#define SLEEP_INTERVAL 1800  // 30 minutes in 8-second intervals
#define SEND_INTERVAL 1350   // 3 hours in 8-second intervals
#define MODBUS_SERIAL_BAUD 4800
#define MODBUS_SERIAL_UBRR F_CPU/16/MODBUS_SERIAL_BAUD-1
#define SD_SECTOR_SIZE 512

volatile uint8_t watchdogEvent = 0;
volatile unsigned int sleepCounter = 0;

float moisture_val = NAN;
float humidity = NAN;
float air_temp = NAN;
float soil_temp = NAN;
float conductivity = NAN;
unsigned long lastWakeTime = 0;
unsigned long lastSendTime = 0;
uint8_t c=0,I_RH,D_RH,I_Temp,D_Temp,CheckSum;

void spi_init() {
    DDRB |= (1 << PB3) | (1 << PB5) | (1 << PB2);
    SPCR = (1 << SPE) | (1 << MSTR) | (1 << SPR0);
    DDRB &= ~(1 << PB4);
}

uint8_t spi_transfer(uint8_t data) {
    SPDR = data;
    while (!(SPSR & (1 << SPIF)));
    return SPDR;
}

void sd_command(uint8_t cmd, uint32_t arg, uint8_t crc) {
    spi_transfer(cmd | 0x40);
    spi_transfer(arg >> 24);
    spi_transfer(arg >> 16);
    spi_transfer(arg >> 8);
    spi_transfer(arg);
    spi_transfer(crc);
}

uint8_t sd_response() {
    uint8_t response;
```

```
    for (uint8_t i = 0; i < 8; i++) {
        response = spi_transfer(0xFF);
        if (response != 0xFF) return response;
    }
    return response;
}

void sd_initialize() {
    PORTB |= (1 << CS_PIN);
    for (uint8_t i = 0; i < 10; i++) spi_transfer(0xFF);

    PORTB &= ~(1 << CS_PIN);
    sd_command(0x00, 0x00000000, 0x95);
    while (sd_response() != 0x01);

    PORTB |= (1 << CS_PIN);
    spi_transfer(0xFF);

    PORTB &= ~(1 << CS_PIN);
    sd_command(0x08, 0x000001AA, 0x87);
    while (sd_response() != 0x01) {
        PORTB |= (1 << CS_PIN);
        spi_transfer(0xFF);
        PORTB &= ~(1 << CS_PIN);
        sd_command(0x08, 0x000001AA, 0x87);
    }

    PORTB |= (1 << CS_PIN);
    spi_transfer(0xFF);
}

void sd_read_sector(uint32_t sector, uint8_t *buffer) {
    PORTB &= ~(1 << CS_PIN);
    sd_command(0x51, sector * SD_SECTOR_SIZE, 0xFF);
    while (sd_response() != 0x00);

    for (uint16_t i = 0; i < SD_SECTOR_SIZE; i++) {
        buffer[i] = spi_transfer(0xFF);
    }

    PORTB |= (1 << CS_PIN);
}

bool seekLastLine(uint8_t *buffer, size_t size, uint32_t *sector, uint32_t *offset) {
    uint32_t currentSector = 0;
    size_t bufferSize = size < SD_SECTOR_SIZE ? size : SD_SECTOR_SIZE;

    // Get file size and sector count. This needs to be implemented based on the file
     structure.
    uint32_t fileSize = 0; // Placeholder: this needs to be calculated or provided

    while (currentSector * SD_SECTOR_SIZE < fileSize) {
        sd_read_sector(currentSector, buffer);
        for (int i = SD_SECTOR_SIZE - 1; i >= 0; --i) {
            if (buffer[i] == '\n') {
                *sector = currentSector;
                *offset = i + 1;
                return true;
            }
        }
        ++currentSector;
    }
    return false;
}

char* readLastLine(void) {
    uint32_t sector;
    uint32_t offset;
```

```
    static char buffer[SD_SECTOR_SIZE + 1]; // Use static buffer to return the string

    if (!seekLastLine(buffer, sizeof(buffer), &sector, &offset)) {
        return NULL; // Return NULL if seeking failed
    }

    // Read the sector containing the last line
    sd_read_sector(sector, buffer);

    // Determine the length of the last line
    size_t lineLen = SD_SECTOR_SIZE - offset;
    if (lineLen >= sizeof(buffer)) {
        lineLen = sizeof(buffer) - 1;
    }
    memcpy(buffer, buffer + offset, lineLen);
    buffer[lineLen] = '\0'; // Null-terminate the string

    return buffer; // Return pointer to the buffer containing the last line
}

void format_string(char* buffer, size_t size, const char* val1, const char* val2,
    const char* val3, const char* val4, const char* val5) {
    if (size == 0) return;

    // Manually build the formatted string
    char* p = buffer;

    // Add "Soil Moisture: "
    const char* prefix1 = "Soil Moisture: ";
    while (*prefix1 && (p - buffer < size - 1)) {
        *p++ = *prefix1++;
    }

    // Add value 1
    while (*val1 && (p - buffer < size - 1)) {
        *p++ = *val1++;
    }

    // Add ", Air Temperature: "
    const char* prefix2 = ", Air Temperature: ";
    while (*prefix2 && (p - buffer < size - 1)) {
        *p++ = *prefix2++;
    }

    // Add value 2
    while (*val2 && (p - buffer < size - 1)) {
        *p++ = *val2++;
    }

    // Add ", Soil Temperature: "
    const char* prefix3 = ", Soil Temperature: ";
    while (*prefix3 && (p - buffer < size - 1)) {
        *p++ = *prefix3++;
    }

    // Add value 3
    while (*val3 && (p - buffer < size - 1)) {
        *p++ = *val3++;
    }

    // Add ", Humidity: "
    const char* prefix4 = ", Humidity: ";
    while (*prefix4 && (p - buffer < size - 1)) {
        *p++ = *prefix4++;
    }

    // Add value 4
    while (*val4 && (p - buffer < size - 1)) {
```

```
        *p++ = *val4++;
    }

    // Add ", Conductivity: "
    const char* prefix5 = ", Conductivity: ";
    while (*prefix5 && (p - buffer < size - 1)) {
        *p++ = *prefix5++;
    }

    // Add value 5
    while (*val5 && (p - buffer < size - 1)) {
        *p++ = *val5++;
    }

    *p = '\0'; // Null-terminate the buffer
}

void USART_Init(unsigned int ubrr) {
    UBRR0H = (unsigned char)(ubrr >> 8);
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1 << RXEN0) | (1 << TXEN0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void USART_Transmit(unsigned char data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void USART_Transmit_String(const char* str) {
    while (*str) {
        USART_Transmit(*str++);
    }
}

void DHT11_Request() {
    DDRB |= (1 << DHT_PIN);
    PORTB &= ~(1 << DHT_PIN);
    _delay_ms(20);
    PORTB |= (1 << DHT_PIN);
}

void DHT11_Response() {
    DDRB &= ~(1 << DHT_PIN);
    while (PINB & (1 << DHT_PIN));
    while ((PINB & (1 << DHT_PIN)) == 0);
    while (PINB & (1 << DHT_PIN));
}

uint8_t DHT11_Receive_Data() {
    uint8_t c = 0;
    for (int q = 0; q < 8; q++) {
        while ((PINB & (1 << DHT_PIN)) == 0);
        _delay_us(30);
        if (PINB & (1 << DHT_PIN)) c = (c << 1) | (0x01);
        else c = (c << 1);
        while (PINB & (1 << DHT_PIN));
    }
    return c;
}

void wdt_init(void) {
    cli();
    MCUSR &= ~(1 << WDRF);
    WDTCSR |= (1 << WDCE) | (1 << WDE);
    WDTCSR = (1 << WDIE) | (1 << WDP3) | (1 << WDP0);
    sei();
}
```

```
266
267 ISR(WDT_vect) {
268     watchdogEvent = 1;
269 }
270
271 int main(void) {
272     wdt_init();
273     USART_Init(MODBUS_SERIAL_UBRR);
274     DDRB |= (1 << RE) | (1 << DE);
275     PORTB &= ~(1 << RE);
276     PORTB &= ~(1 << DE);
277
278     while (1) {
279         if (watchdogEvent) {
280             watchdogEvent = 0;
281             sleepCounter++;
282
283             if (sleepCounter >= SLEEP_INTERVAL) {
284                 sleepCounter = 0;
285
286                 DHT11_Request();
287                 DHT11_Response();
288                 I_RH = DHT11_Receive_Data();
289                 D_RH = DHT11_Receive_Data();
290                 I_Temp = DHT11_Receive_Data();
291                 D_Temp = DHT11_Receive_Data();
292                 CheckSum = DHT11_Receive_Data();
293
294                 if ((I_RH + D_RH + I_Temp + D_Temp) == CheckSum) {
295                     humidity = I_RH + (float)D_RH / 10.0;
296                     air_temp = I_Temp + (float)D_Temp / 10.0;
297                 }
298
299                 char buffer[256];
300                 format_string(buffer, sizeof(buffer), "Val1", "Val2", "Val3", "Val4",
     "Val5");
301                 USART_Transmit_String(buffer);
302                 USART_Transmit('\n');
303             }
304
305             set_sleep_mode(SLEEP_MODE_PWR_DOWN);
306             sleep_mode();
307         }
308     }
309 }
```

ESP8266 Code to Communicate with Firebase

```
1  #include <string.h>
2  #include "freertos/FreeRTOS.h"
3  #include "freertos/task.h"
4  #include "freertos/event_groups.h"
5  #include "esp_system.h"
6  #include "esp_wifi.h"
7  #include "esp_event.h"
8  #include "esp_log.h"
9  #include "nvs_flash.h"
10 #include "lwip/sockets.h"
11
12 #define WIFI_SSID      "Farm_Assistor"
13 #define WIFI_PASS      "edr1234"
14 #define MAX_RETRY      5
15 #define FIREBASE_HOST  "Farm_assistor"
16 #define FIREBASE_AUTH  "Fram_assistor_users"
17
18 static EventGroupHandle_t s_wifi_event_group;
19 #define WIFI_CONNECTED_BIT BIT0
20 #define WIFI_FAIL_BIT      BIT1
```

```
21
22 static const char *TAG = "wifi station";
23 static int s_retry_num = 0;
24
25 static void event_handler(void* arg, esp_event_base_t event_base, int32_t event_id,
       void* event_data) {
26     if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
27         esp_wifi_connect();
28     } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
29         if (s_retry_num < MAX_RETRY) {
30             esp_wifi_connect();
31             s_retry_num++;
32             ESP_LOGI(TAG, "retry to connect to the AP");
33         } else {
34             xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
35         }
36         ESP_LOGI(TAG,"connect to the AP fail");
37     } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
38         ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
39         ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
40         s_retry_num = 0;
41         xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
42     }
43 }
44
45 void wifi_init_sta(void) {
46     s_wifi_event_group = xEventGroupCreate();
47     esp_netif_init();
48     esp_event_loop_create_default();
49     esp_netif_create_default_wifi_sta();
50
51     wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
52     esp_wifi_init(&cfg);
53
54     esp_event_handler_instance_t instance_any_id;
55     esp_event_handler_instance_t instance_got_ip;
56     esp_event_handler_instance_register(WIFI_EVENT, ESP_EVENT_ANY_ID, &event_handler,
        NULL, &instance_any_id);
57     esp_event_handler_instance_register(IP_EVENT, IP_EVENT_STA_GOT_IP, &event_handler
       , NULL, &instance_got_ip);
58
59     wifi_config_t wifi_config = {
60         .sta = {
61             .ssid = WIFI_SSID,
62             .password = WIFI_PASS,
63             .threshold.authmode = WIFI_AUTH_WPA2_PSK,
64         },
65     };
66     esp_wifi_set_mode(WIFI_MODE_STA);
67     esp_wifi_set_config(WIFI_IF_STA, &wifi_config);
68     esp_wifi_start();
69
70     EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group, WIFI_CONNECTED_BIT |
       WIFI_FAIL_BIT, pdFALSE, pdFALSE, portMAX_DELAY);
71     if (bits & WIFI_CONNECTED_BIT) {
72         ESP_LOGI(TAG, "connected to ap SSID:%s password:%s", WIFI_SSID, WIFI_PASS);
73     } else if (bits & WIFI_FAIL_BIT) {
74         ESP_LOGI(TAG, "Failed to connect to SSID:%s, password:%s", WIFI_SSID,
       WIFI_PASS);
75     } else {
76         ESP_LOGE(TAG, "UNEXPECTED EVENT");
77     }
78
79     esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP,
       instance_got_ip);
80     esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID,
       instance_any_id);
81     vEventGroupDelete(s_wifi_event_group);
```

```
}

void send_data_to_firebase(const char* path, const char* data) {
    int sock;
    struct sockaddr_in dest_addr;
    dest_addr.sin_addr.s_addr = inet_addr(FIREBASE_HOST);
    dest_addr.sin_family = AF_INET;
    dest_addr.sin_port = htons(80);

    sock = socket(AF_INET, SOCK_STREAM, IPPROTO_IP);
    if (sock < 0) {
        ESP_LOGE(TAG, "Unable to create socket: errno %d", errno);
        return;
    }

    int err = connect(sock, (struct sockaddr *)&dest_addr, sizeof(dest_addr));
    if (err != 0) {
        ESP_LOGE(TAG, "Socket unable to connect: errno %d", errno);
        close(sock);
        return;
    }

    char request[512];
    snprintf(request, sizeof(request),
             "PUT /%s.json?auth=%s HTTP/1.1\r\n"
             "Host: %s\r\n"
             "Content-Type: application/json\r\n"
             "Content-Length: %d\r\n"
             "\r\n"
             "{\"value\": \"%s\"}\r\n",
             path, FIREBASE_AUTH, FIREBASE_HOST, strlen(data) + 12, data);

    err = send(sock, request, strlen(request), 0);
    if (err < 0) {
        ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
    }
    close(sock);
}

void uart_task(void *pvParameter) {
    char rx_buffer[128];
    while (1) {
        int len = uart_read_bytes(UART_NUM_1, (uint8_t *)rx_buffer, sizeof(rx_buffer)
     - 1, 20 / portTICK_RATE_MS);
        if (len > 0) {
            rx_buffer[len] = 0;
            ESP_LOGI(TAG, "Received: %s", rx_buffer);
            send_data_to_firebase("sensor/data", rx_buffer);
        }
        vTaskDelay(10000 / portTICK_RATE_MS);
    }
}

void app_main(void) {
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");
    wifi_init_sta();

    const uart_config_t uart_config = {
        .baud_rate = 9600,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
```

```
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
    };
    uart_param_config(UART_NUM_1, &uart_config);
    uart_set_pin(UART_NUM_1, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE,
    UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
    uart_driver_install(UART_NUM_1, 2048, 0, 0, NULL, 0);

    xTaskCreate(uart_task, "uart_task", 2048, NULL, 10, NULL);
}
```

Listing 1: ESP32 WiFi and Firebase Integration Code

Note : All the references used for coding purposes are listed in the reference section.

8 Mobile Application

8.0.1 Architecture Overview

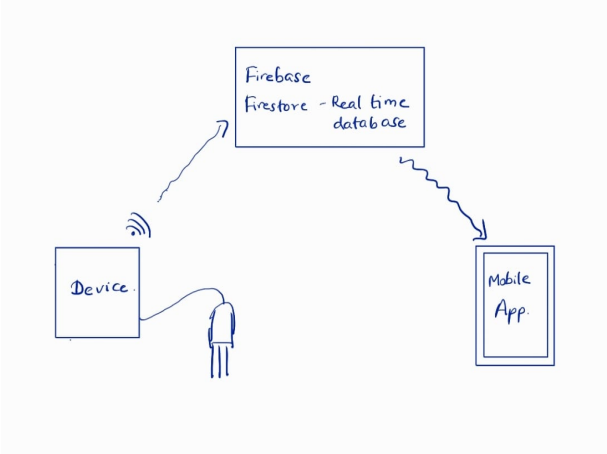
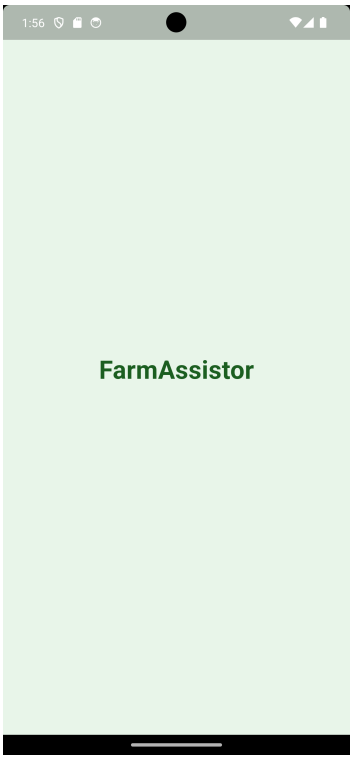


Figure 35: Architecture overview

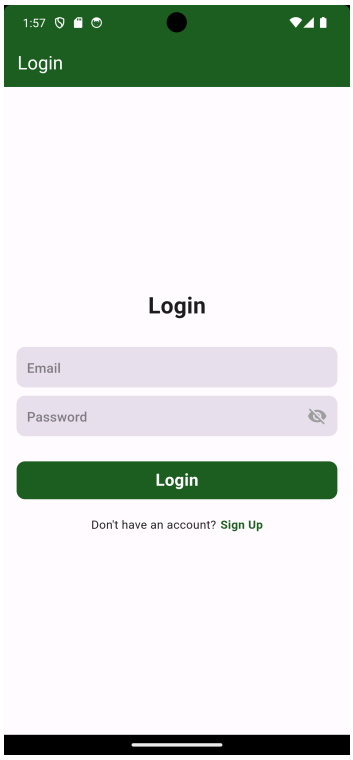
For our app development, we selected Firebase, a platform by Google to host the real-time database and authentication system. We chose Flutter, a framework based on the Dart language, for developing the user interface. Detailed guidelines for setting up required environments is given in the appendix comprehensive design details section.

8.0.2 Final Application Structure

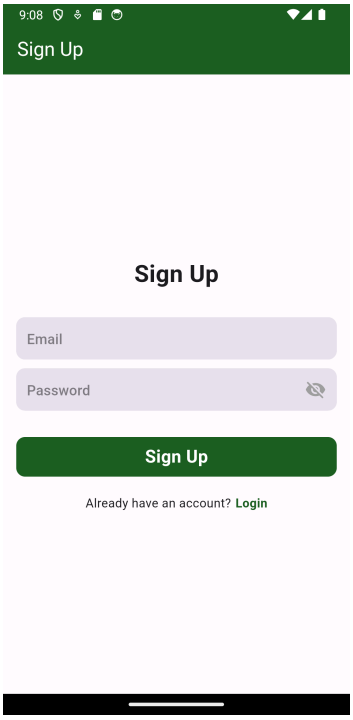
We have designed the final application structure to satisfy the identified requirements in the most efficient manner.



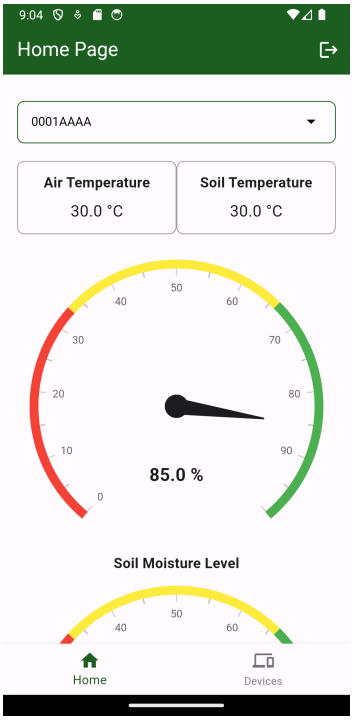
(a) Splash Screen



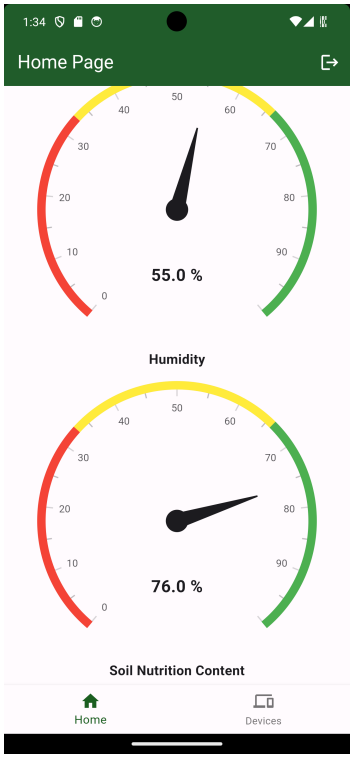
(b) Login Page



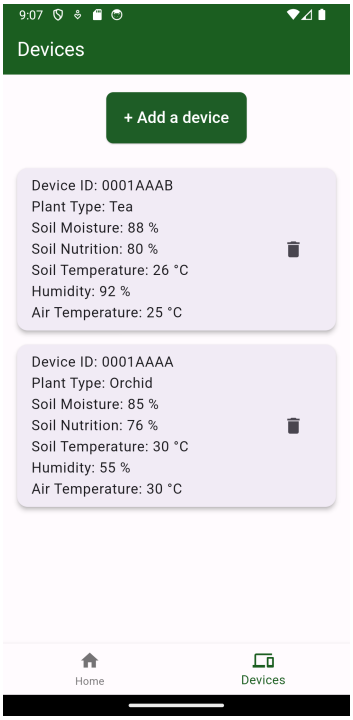
(c) Sign Up Page



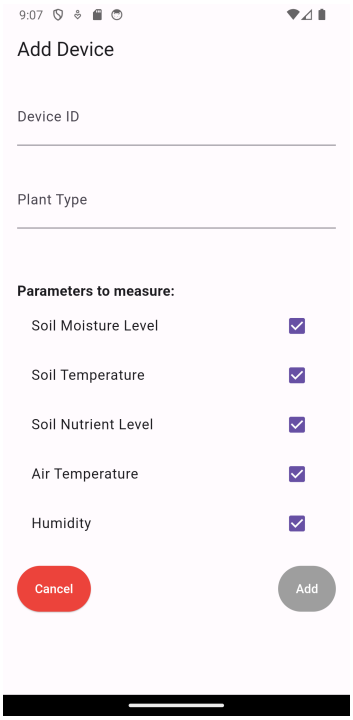
(d) Home Page



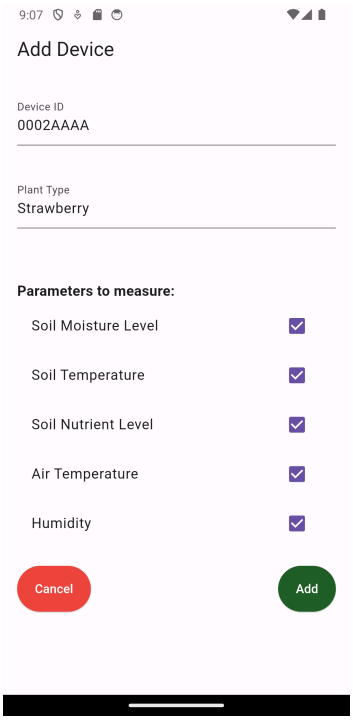
(e) Home Page



(f) Devices Page



(a) Add Device unfilled



(b) Add Device filled

Figure 37: Application Structure

9 Comprehensive Design details

9.1 Component Selection

As the first step we have decided on the sensors to be used, below are the sensors considered in this process.

- **RS485 Soil Sensor**
  One of the main reason for considering this sensor is it's RS485 output signal, This is an industrially used standard for sensor communication. Also it facilitate easier scalability of the product due to it's modbus protocol. Another reason is it's ability to measure conductivity of the soil which is quite rare in the sensors available in the market.
  After deciding on the sensor we considered few manufactures as follows,

  ComWinTop
  jxct
  Seeed Studio

  After evaluating the above products we have decided to use the sensor by ComWinTop due to the cost effectiveness and user ratings. They also provide satisfactory after sales services to help setup the sensor.

- **Capacitive Soil Moisture Sensor**
  Though this sensor only measure the soil mosture level, it was considered as a potential sensor due to it's low cost and high availability. Moreover soil moisture is the most important

parameter to be measured. Though it is not very reliable it will provide satisfactory results if it is well calibrated.

- **Resistive Soil Moisture Sensor**
  These sensors have a similar functionality to the capacitive sensor but there principle of operation is different. This sensor is not very reliable as there electrodes tend to corrode resulting non durable product

After careful consideration of above factors we have selected RS485 Soil sensor as our primary sensor and capacitive sensor as a sensor so that we can demonstrate the ability of the product to measure soil conditions of multiple places at the same time. To connect this sensor to the micro

(a) Capacitive Soil Sensor (b) Resistive Soil Sensor (c) RS485 Soil Sensor

Figure 38: Soil sensors

controller we have decided to use MAX485 IC which is commonly used as a transceiver for RS-485. This IC also has low power consumption making it ideal to be used for our application.

DHT22 air temperature and humidity sensor
Environmental conditions are critical for plant monitoring devices, so we have enhanced our product by incorporating the DHT22 sensor to measure temperature and humidity. The DHT22 was selected for its accuracy and reliability. It can measure temperatures from -40 to 80 degrees Celsius and humidity levels from 0 to 100% RH. The sensor uses a capacitive humidity sensor and a thermistor to measure the surrounding air, providing a digital signal on a single data line. This makes it straightforward to interface with microcontrollers, ensuring precise and efficient monitoring of environmental conditions.

When selecting the microcontroller we considered few options like ATMega328p, ESP32, ESP8266, RP2040. When selecting the best option from above we considered the following factors,

- Low power sleep mode
- Required number of analogue and digital pins to support the sensors and other functionalities.
- Availability - Microcontrller should be readily available to support possible industrial level production in the future.
- Reliability and robustness with proven track record.

Based on above criteria we chose ATMega328p as the microcontroller for the product.

To get the functionality of the datalogger we have decided to use a microSD card with a real time clock circuit. This will be simple to implement and power effective. We can easily read and write data using the microSD card. The measured sensor data will be written to the micrSD card at certain intervals. When required these written data can be easily transmitted to WiFi connectivity.

ESP-01 WiFi Module is chosen for the WiFi connectivity of the product. Main considerations when selecting the WiFi module was it's power requirements. As ESP-01 WiFi Module has very low sleep currents it can be used for this application.

Based on the components chosen we need both 5V and 3.3V supply in the circuit. to accomadate this we choose LM7805 (5V regulator) and AMS1117 (3.3V regulator). The product will be powered by two lithium ion rechargable batteries with combined voltage of 7.4V, which can ensure proper operation of both selected voltage regulators.

9.2 Power Calculations

After careful examination of the datasheets of the selected components, the following values were taken as the power and current requirements.

- **RS485 Sensor:**
  Current drawn : 20mA
  Voltage : 5V
- **Capacitive Sensor:**
  Current drawn : 5mA
  Voltage : 5V
- **Microcontroller:**
  Active mode :1.5 mA
  Sleep mode :1uA
- **Wi-Fi Module:**
  Active mode :80 mA
  Sleep mode :10uA
  Voltage :3.3V
- **Battery:**
  Nominal Voltage:3.7
  Capacity :2000mAh

Therefore maximum current drawn at any time will be around 120mA. By keeping a margin of saftey we calculated the trace width for 400 mA. It requires 0.221 mm trace width. Therefore as a standard practice we have used 0.254mm or 10 mil traces for signal lines and 0.5mm traces for power lines.

9.3 Circuit Analysis

Sensor Input Unit

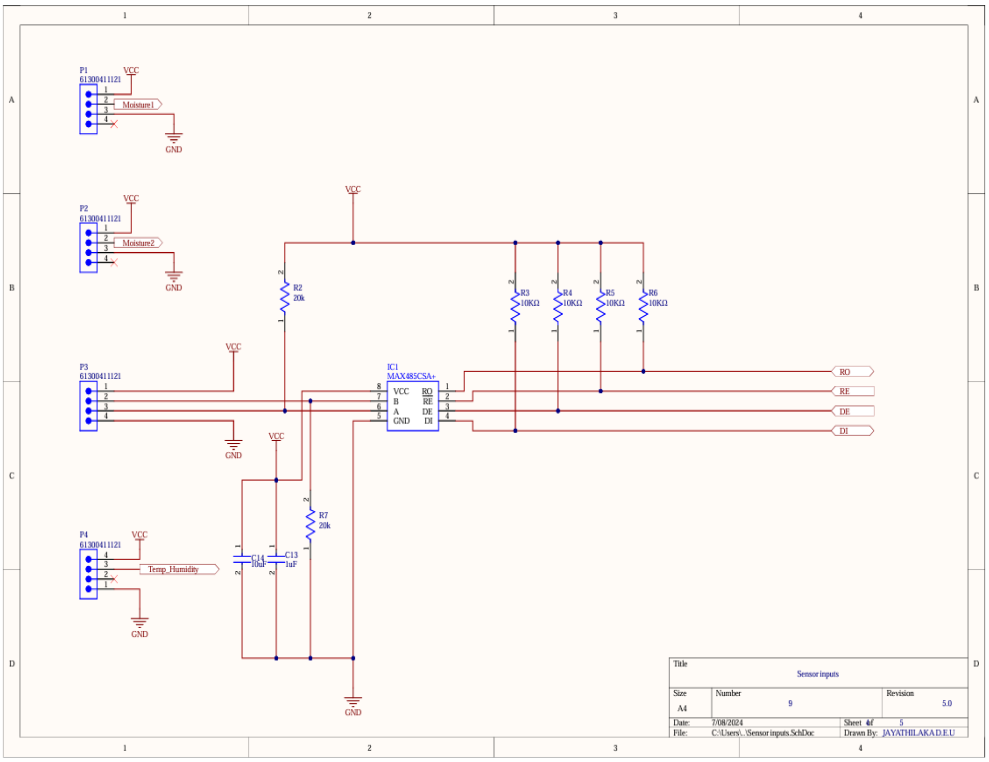


Figure 39: Sensor Connection

The sensor inputs diagram focuses on interfacing various sensors with the microcontroller. Soil moisture sensors (P1 and P2) and a temperature and humidity sensor (P4) are connected directly to the microcontroller. The RS485 output soil parameter measuring sensor is connected to a MAX485 RS-485 transceiver (IC1). The MAX485 chip facilitates robust communication over longer distances. Before interfacing with the ATmega328P microcontroller. Bypass capacitors (C13 and C14, 1uF each) are used to stabilize the circuit by filtering out noise from the power supply, ensuring the MAX485 chip receives clean and stable power. Pull-up resistors (R2 and R7) are used to maintain proper signal levels on the communication lines, ensuring accurate data transmission. Termination resistors (R3, R4, R5, R6) prevent signal reflections on the long communication lines, which can otherwise cause data corruption. This setup ensures reliable and accurate readings from the sensors, which are then transmitted to the microcontroller for processing.

Microcontroller Unit

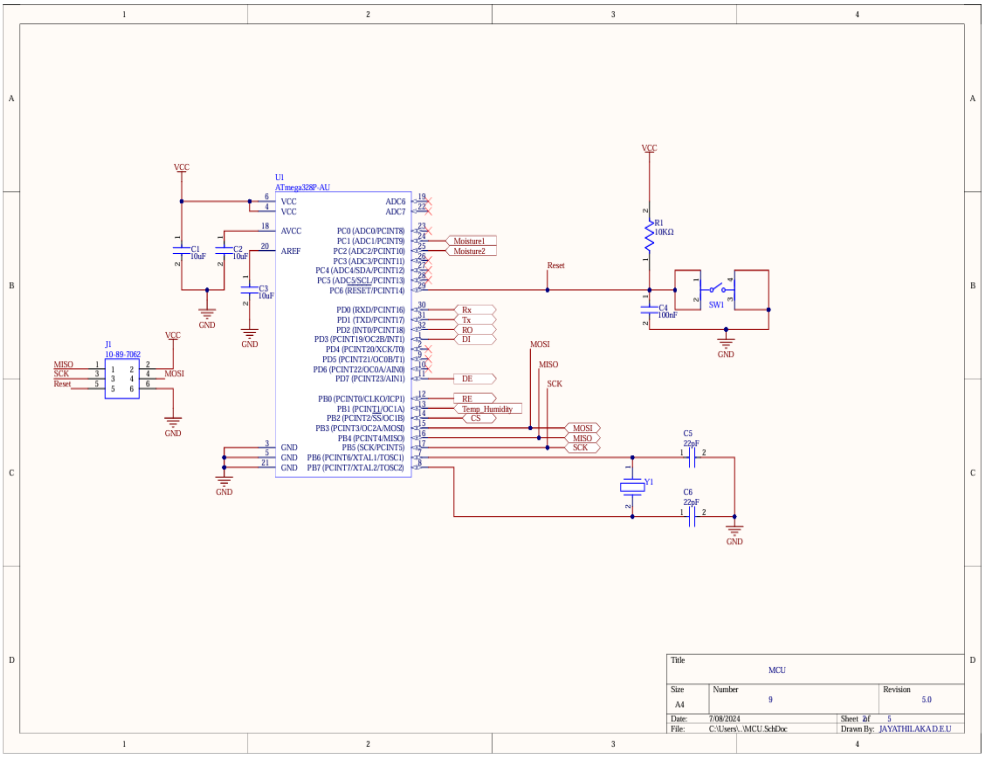


Figure 40: Microcontroller Unit

The MCU diagram details the setup of the ATmega328P-AU microcontroller, which serves as the central processing unit for the system. The microcontroller interfaces with various peripherals, including sensors and the microSD card. Power supply stability is ensured through multiple decoupling capacitors (10uF) connected between VCC and GND, filtering out noise and providing a stable voltage. A 16MHz crystal oscillator, with capacitors (C5 and C6, both 22pF), provides the necessary clock signal for the microcontroller, ensuring precise timing and operation. A reset circuit, consisting of a push-button switch (SW1), pull-up resistor (R1, 10kohm), and a capacitor (100nF), allows for manual resetting of the microcontroller. Additionally, a programming interface connector (J1) provides access to the necessary lines (MISO, MOSI, SCK, and Reset) for in-system programming, facilitating firmware updates and debugging.

Power Unit

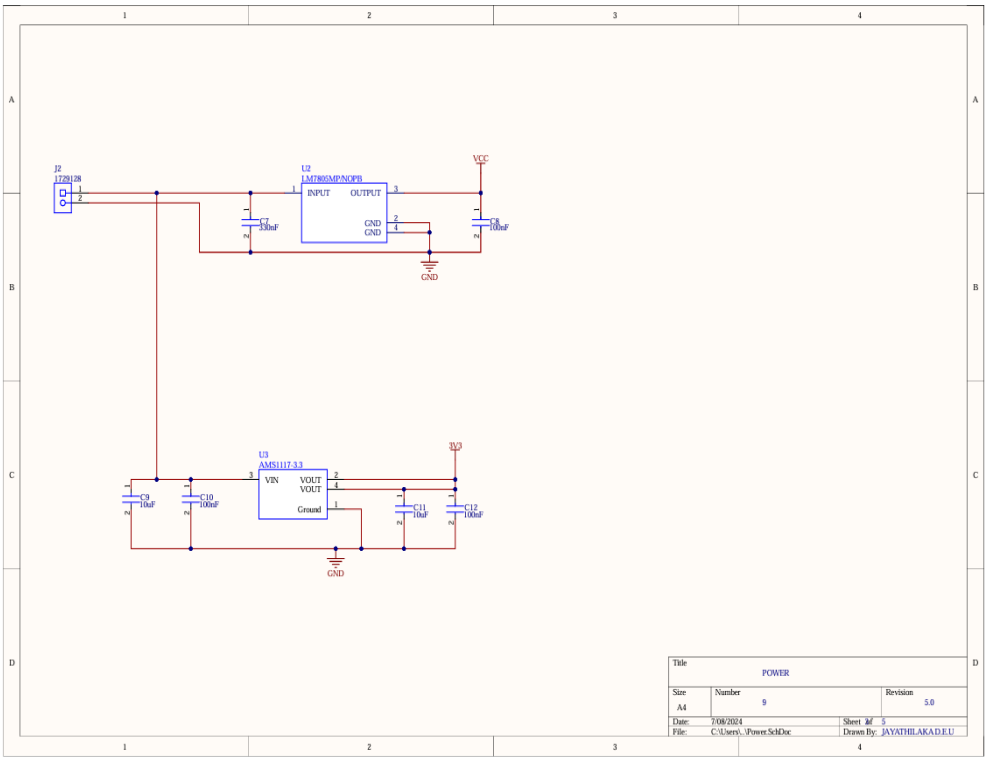


Figure 41: Power Unit

The power circuit involves two voltage regulators to provide the necessary voltages for different components. The LM7805 regulator is used to obtain a stable 5V supply, which is essential for the operation of the microcontroller and some of the sensors. The AMS1117 voltage regulator is used to provide a 3.3V supply, required by components like the microSD card and the WiFi module. These regulators ensure that each component receives the appropriate voltage level, enhancing the reliability and stability of the entire system.

DataLogger Unit

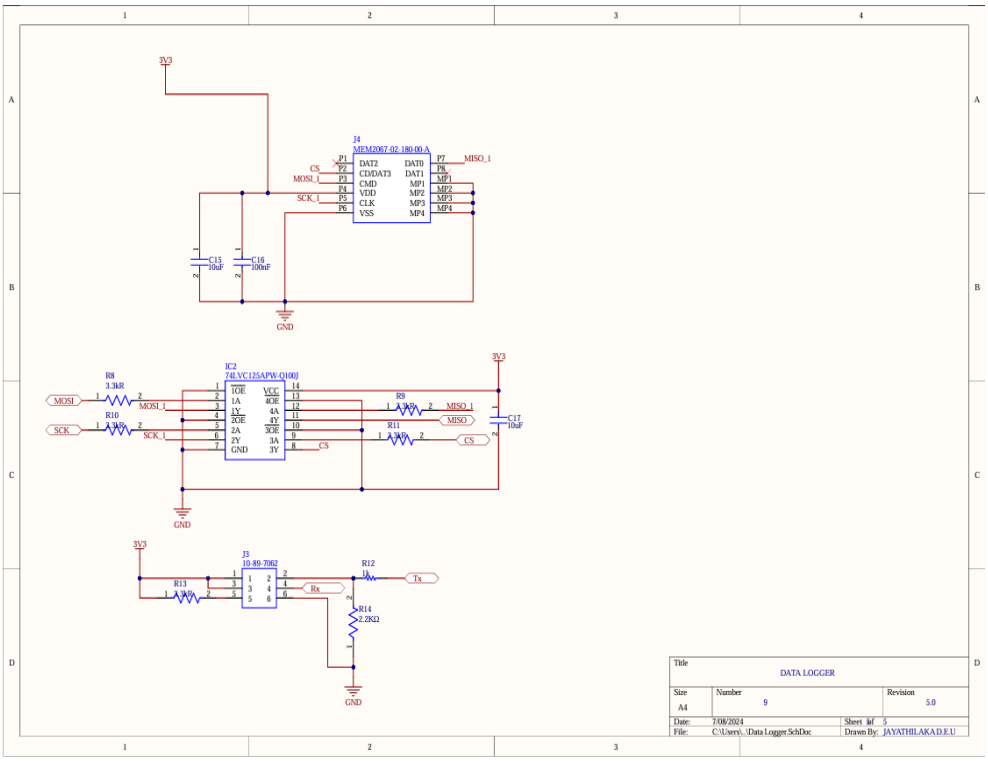


Figure 42: data logger Unit

The datalogger diagram illustrates a system designed for storing sensor data on a microSD card. In this unit, the microSD card is used to store data collected from sensors, allowing the system to minimize power consumption by reducing the frequency of activating the WiFi module, which is highly power-consuming.
At the heart of this system is the microSD card socket (J4), which interfaces with the microcontroller via SPI communication. The signals MOSI, MISO, SCK, and CS are routed through a 74LVC125A quad buffer/line driver (IC2), which ensures proper voltage level shifting between the microcontroller (operating at 5V) and the microSD card (operating at 3.3V). The buffer/line driver protects the microSD card from potentially damaging higher voltages and ensures reliable data transfer, effectively stabilizing the data written to the SD card.
Additionally, decoupling capacitors (C15 and C16, both 10uF) are placed near the power pins of the microSD card to stabilize the voltage supply and filter out noise, ensuring stable operation. The use of pull-up resistors (R8, R9, R10, R11) helps in maintaining signal integrity.
J3 is a header to which we can connect the WiFi module, allowing for data transmission to a remote database when required. By storing the data on the SD card first, the system can batch transmissions, significantly reducing the frequency of WiFi module activations and thus conserving power.

9.4 PCB Design Standards

Below listed are the main points considered during PCB design to ensure industry standard PCB which is efficient and reliable for the given purpose.

- **Planning and Documentation**
  - Clearly define the requirements and constraints of the PCB design.
  - Create a detailed schematic diagram to map out connections and components.
  - Document design decisions, constraints, and any special considerations.
- **Component Placement**
  - Strategically place components to minimize signal interference and optimize signal integrity.
  - Group related components together to simplify routing and improve thermal management.
  - Consider component height, orientation, and accessibility for assembly and maintenance.
- **Signal Integrity**
  - Maintain signal integrity by minimizing trace lengths, avoiding sharp corners, and using appropriate trace widths for high-speed signals.
  - Use ground planes and thicker power lines to reduce noise and provide stable power distribution.
- **Routing**
  - Route critical signals first, such as clock lines and high-speed data lines.
  - Follow good routing practices, such as avoiding crossing over split ground or power planes, and minimizing the use of vias.
- **Power Distribution**
  - Ensure proper power distribution by minimizing voltage drops and providing adequate decoupling capacitors near power pins of ICs.
  - Route power traces with sufficient width to handle the required current without overheating or voltage drop.
- **Design for Manufacturability (DFM)**
  - Design the PCB with manufacturability in mind to reduce manufacturing costs and improve yield.
  - Follow PCB fabrication guidelines provided by the manufacturer, including minimum trace width and spacing, and design clearances.
  - Ensure that the design meets assembly requirements, such as component placement tolerances and assembly process compatibility.
- **Review and Validation**
  - Conduct thorough design reviews to identify potential issues and validate the design against requirements and specifications.

9.5 Soldering

Figure 43: Bare PCB - Top View

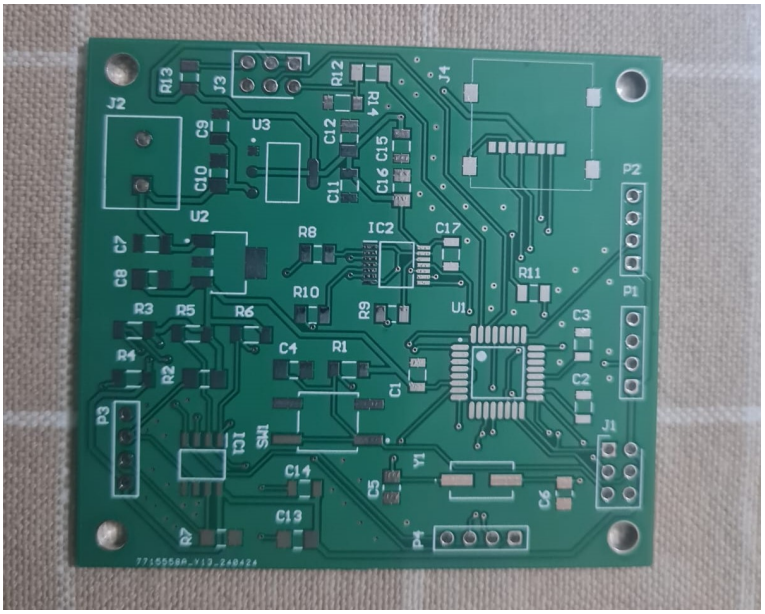


Figure 44: Bare PCB - Bottom View

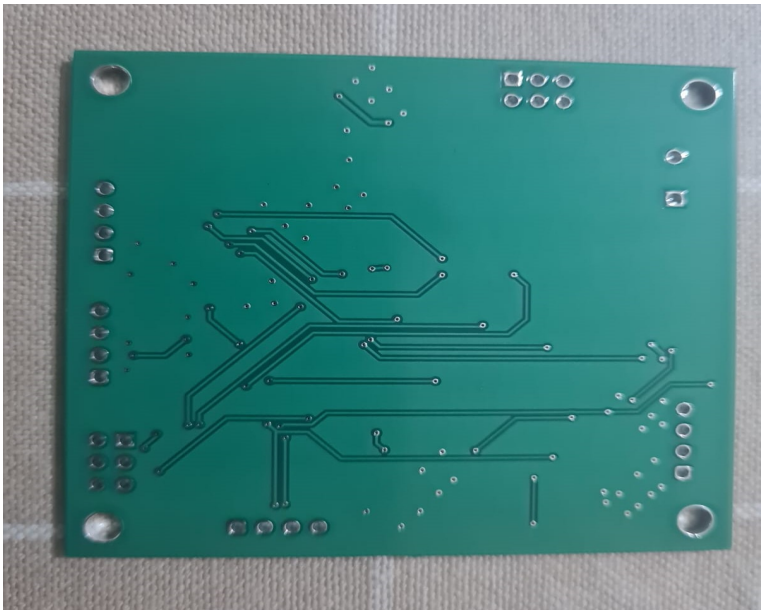
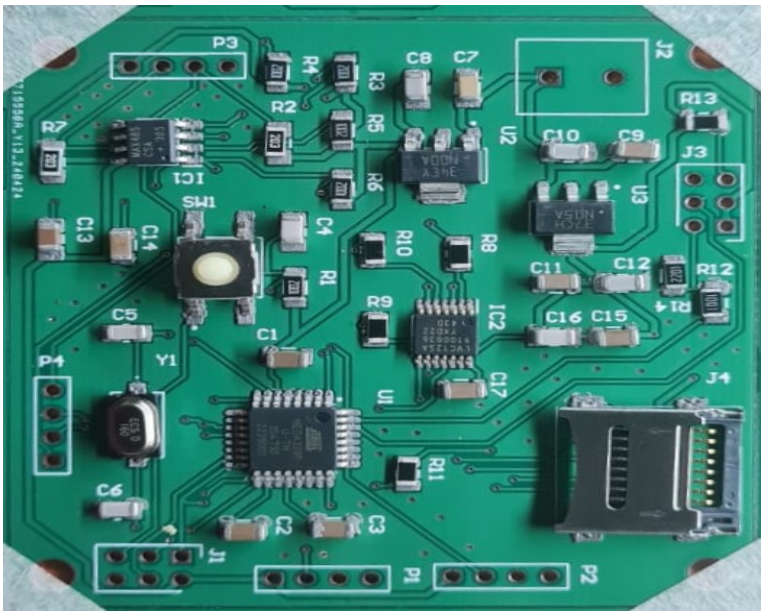


Figure 45: Soldered PCB



9.6 Bill of Material

All the components in the below table were purchased from mouser

Figure 46: BOM - Mouser Electronics

MOUSER ELECTRONICS

SHOPPING CART

This is not an invoice

17-Apr-24 11:06:28

	Mouser #	Mfr. #	Manufacturer	Order Qty.	Ext.: (USD)
1	187-CL31B106KBHNNNE	CL31B106KBHNNNE	Samsung Electro-Mechanics	10	$1.60
2	81-GRM31C5C1H104JA1L	GRM31C5C1H104JA01L	Murata	10	$2.46
3	81-GRM31A5C2J220JW1D	GRM31A5C2J220JW01D	Murata	3	$0.96
4	603-CC1206KKX70BB334	CC1206KKX7R0BB334	YAGEO	2	$0.64
5	581-12061C105K4T2A	12061C105K4T2A	KYOCERA AVX	2	$0.92
6	700-MAX485CSA	MAX485CSA+	Analog Devices Inc.	2	$8.46
7	771-74LVC125APWQ100J	74LVC125APW-Q100J	Nexperia	2	$0.84
8	538-10-89-7062	10-89-7062	Molex	2	$1.74
9	651-1729128	1729128	Phoenix Contact	1	$1.08
10	640-MEM20670218000A	MEM2067-02-180-00-A	Global Connector Technology (GCT)	2	$2.30
11	710-61300411121	61300411121	Wurth Elektronik	4	$0.76
12	603-RT1206BRD0710KL	RT1206BRD0710KL	YAGEO	10	$2.41
13	667-ERA-8AEB203V	ERA-8AEB203V	Panasonic	3	$0.96
14	71-CRCW1206-3.3K-E3	CRCW12063K30FKEA	Vishay	10	$0.31
15	506-1571634-2	1571634-2	TE Connectivity	1	$0.50
16	556-ATMEGA328P-AU	ATMEGA328P-AU	Microchip	2	$5.32
17	595-LM7805MP/NOPB	LM7805MP/NOPB	Texas Instruments	3	$5.19
18	926-LM1117MPX3.3NOPB	LM1117MPX-3.3/NOPB	Texas Instruments	2	$2.26
19	520-160-20-3XT	ECS-160-20-3X-TR	ECS	2	$0.64

By submitting your order you agree to these terms and conditions. $39.35

Prices are reflected at the date and time shown. $7.99

The Sensors were ordered from the selected manufactures specified using AliExpress online platform.

9.7 Platform and Framework Selection

When developing a mobile app, various platforms and frameworks were considered. Ultimately, Firebase was chosen for its open-source nature, scalability, authentication services, and real-time database capabilities. Here's an overview of other similar platforms and the reasons they were not selected, as well as a comparison between Flutter and React Native for UI design, with the reasons for choosing Flutter.

9.7.1 Alternative Platforms to Firebase

AWS Amplify

- **Pros:** Comprehensive set of features including authentication, storage, API, and analytics. Highly scalable and integrates well with other AWS services.
- **Cons:** Complex setup and configuration. Higher learning curve for new developers. Costs can escalate with increased usage.

Backendless

- **Pros:** Offers a broad range of features including user management, push notifications, and real-time databases.
- **Cons:** Less popular compared to Firebase, which means less community support and fewer resources for troubleshooting.

Parse

- **Pros:** Open-source, flexible, and supports various backends including self-hosting.
- **Cons:** Requires significant effort to manage and scale the infrastructure, less integrated compared to Firebase.

9.7.2 Reasons for Selecting Firebase

- **Ease of Integration:** Firebase provides seamless integration with other Google services and is relatively easy to set up.
- **Comprehensive Features:** Offers a wide range of built-in features such as authentication, cloud storage, and real-time databases, which reduces the need for additional services.
- **Scalability:** Firebase scales automatically to accommodate increased traffic and data usage, making it suitable for both small and large applications.
- **Community and Support:** Large user base and extensive documentation, ensuring ample resources for learning and troubleshooting.

9.7.3 Comparison between Flutter and React Native for UI Design

Performance

- **Flutter:** Uses the Dart language and compiles to native ARM code, which generally provides superior performance and faster runtime.
- **React Native:** Uses JavaScript and relies on a bridge to communicate with native modules, which can introduce performance bottlenecks.

Development Speed

- **Flutter:** Provides a rich set of pre-designed widgets and a hot-reload feature, which speeds up the development process.
- **React Native:** Also supports hot-reloading but relies more on third-party libraries, which can sometimes lead to compatibility issues.

UI Consistency

- **Flutter:** Ensures consistent UI across all platforms due to its own rendering engine, Skia.
- **React Native:** Leverages native components, which can lead to inconsistencies across different platforms.

Community and Support

- **Flutter:** Growing community with increasing adoption by developers and companies.
- **React Native:** Larger and more established community with abundant resources, but Flutter is quickly catching up.

9.7.4 Reasons for Selecting Flutter

- **Consistent UI and Performance:** Flutter's own rendering engine ensures consistent performance and appearance across platforms.
- **Rich Set of Widgets:** Flutter provides a comprehensive collection of pre-designed widgets that adhere to Material Design and Cupertino (iOS-style) standards, allowing for faster and more consistent UI development.
- **High Performance:** Compiles to native ARM code, providing superior performance compared to JavaScript-based frameworks like React Native.
- **Growing Ecosystem:** Flutter's ecosystem is rapidly expanding with strong community support and increasing adoption by developers.

9.7.5 Summary

Firebase was selected for its comprehensive feature set and scalability, while Flutter was chosen for its superior performance, consistent UI, and rapid development capabilities.

9.8 Mobile Application Development

To build our app using Flutter, we need to install Flutter, Android Studio, and Firebase. This process involves several detailed steps to ensure a smooth development environment setup.

Installing Flutter

- **Download Flutter SDK:** Visit the Flutter website and download the appropriate SDK for your operating system. Download and install the latest version of Flutter from the official website: Flutter Installation Guide.
- **Update Path:** Add the Flutter bin directory to your system's PATH environment variable. This allows you to run Flutter commands from any terminal. For example, on Windows, you would add `C:\path\to\flutter\bin` to your PATH.
- **Run Flutter Doctor:** Open a terminal and run the `flutter doctor` command. This command checks your environment and reports any missing dependencies you need to install.

Installing Android Studio

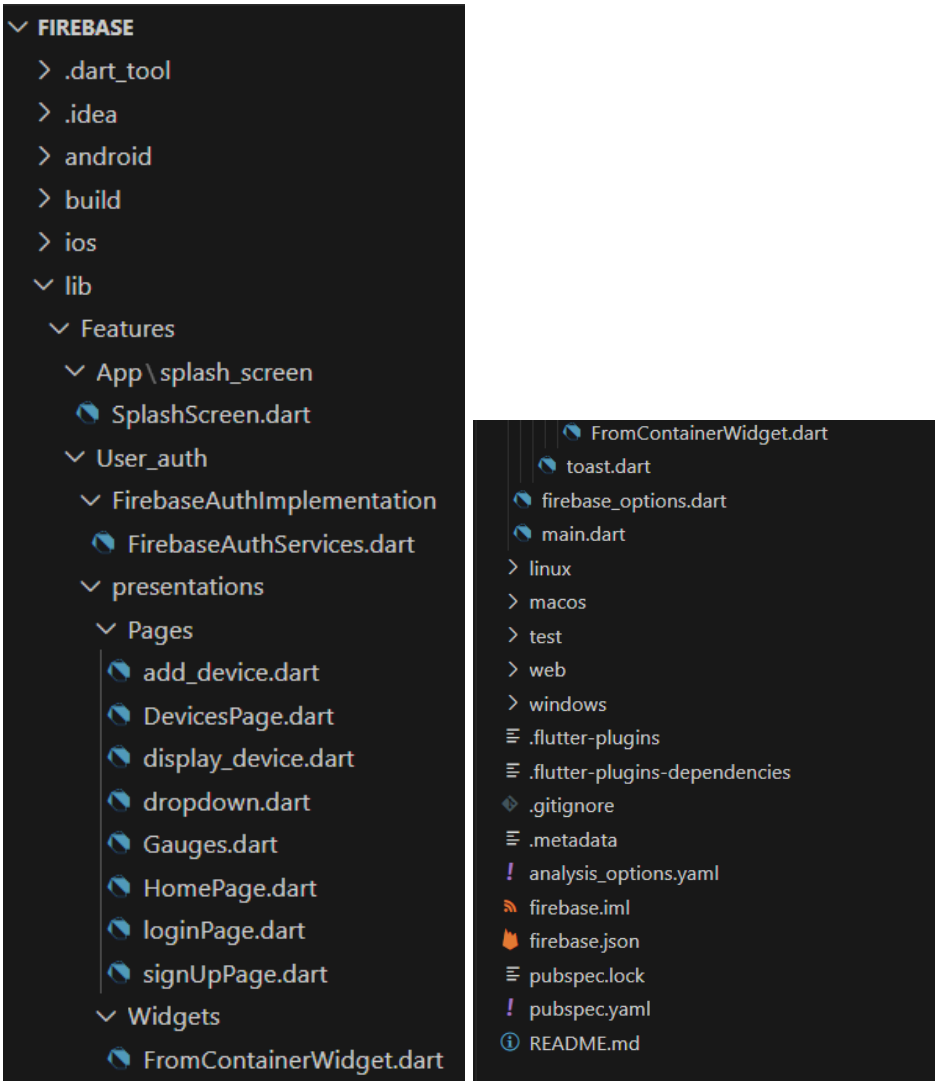
- **Download Android Studio:** Go to the Android Studio download page and download the installer for your operating system. Download and install the latest version of Android Studio from the official website: Android Studio.
- **Install Android Studio:** Follow the installer's instructions to complete the installation.
- **Set Up Android Studio:** Open Android Studio and follow the setup wizard to install the necessary SDK packages and tools.
- **Configure Virtual Device:** Set up an Android Virtual Device (AVD) within Android Studio. Use the AVD Manager to create a new virtual device, following the prompts.

Integrating Firebase

- **Create Firebase Project:** Go to the Firebase Console and create a new project, providing the necessary details like the project name. Visit the official website to access the Firebase Console: Firebase Console.
- **Add Firebase to Your App:** Follow Firebase's instructions to integrate it with your Android and iOS apps. Download the configuration files (`google-services.json` for Android and `GoogleService-Info.plist` for iOS) and place them in the appropriate directories in your Flutter project.
- **Configure Firebase:** Initialize Firebase in your Flutter app, typically in the `main.dart` file. Import the required Firebase packages and call `Firebase.initializeApp()` before running your app.

9.8.1 Mobile Application Code

Basic code structure



(a) (b)

Figure 47: Basic Code Structure of Application

In the basic structure we have defined special files `FireBaseAuthImplementation` and `firebase_options` to handle the user authentication and data retrieval from the firebase user authentication and firestore real time data base. The splash screen file contain the temporary screen visible for few seconds before loading the application. The pages folder contain pages of the application and files defined for separate functionalities of pages. FromContainerWidgets file has UI components related to login and sign up pages. Similarly toast file also contain UI components that are used across multiple pages. Main file is used to interconnect pages and initiate the overall structure.

Main.dart File

The Flutter application initializes with Firebase for user authentication and data management. The `main()` function ensures Firebase services are initialized using `Firebase.initializeApp()` before launching the app. The `MyApp` class serves as the root widget, utilizing `MaterialApp` to define routes for main screens such as SplashScreen, LoginPage, SignUpPage, and HomePage. These routes manage navigation based on user interactions and authentication status.

```dart
import 'package:firebase/Features/App/splash_screen/SplashScreen.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/HomePage.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/loginPage.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/signUpPage.dart';
import 'package:firebase/firebase_options.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';

Future<void> main() async {
  WidgetsFlutterBinding.ensureInitialized();
  if (kIsWeb) {
    await Firebase.initializeApp(
      options: FirebaseOptions(
        apiKey: "YOUR_API_KEY",
        appId: "YOUR_APP_ID",
        messagingSenderId: "YOUR_MESSAGING_SENDER_ID",
        projectId: "YOUR_PROJECT_ID",
      ),
    );
  }

  await Firebase.initializeApp(
    options: DefaultFirebaseOptions.currentPlatform,
  );
  runApp(const MyApp());
}

class MyApp extends StatelessWidget {
  const MyApp({Key? key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: "FarmAssistor",
      routes: {
        '/': (context) => SplashScreen(
          child: loginPage(),
        ),
        '/login': (context) => loginPage(),
        '/signUp': (context) => signUpPage(),
        '/home': (context) => HomePage(),
      },
    );
  }
}
```

Firebase Options File

The `DefaultFirebaseOptions` class offers default Firebase setup options for various platforms, automatically choosing the correct configuration based on whether the platform is web, Android, or Windows. This ensures that Firebase services such as authentication and data storage start up correctly. Each platform-specific setup includes details like API keys, app IDs, and project identifiers.

```dart
// File generated by FlutterFire CLI.
// ignore_for_file: type=lint
import 'package:firebase_core/firebase_core.dart' show FirebaseOptions;
```

```dart
import 'package:flutter/foundation.dart'
    show defaultTargetPlatform, kIsWeb, TargetPlatform;

/// Default [FirebaseOptions] for use with your Firebase apps.
///
/// Example:
/// ```dart
/// import 'firebase_options.dart';
/// // ...
/// await Firebase.initializeApp(
///   options: DefaultFirebaseOptions.currentPlatform,
/// );
/// ```
class DefaultFirebaseOptions {
  static FirebaseOptions get currentPlatform {
    if (kIsWeb) {
      return web;
    }
    switch (defaultTargetPlatform) {
      case TargetPlatform.android:
        return android;
      case TargetPlatform.iOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for ios - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.macOS:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for macos - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      case TargetPlatform.windows:
        return windows;
      case TargetPlatform.linux:
        throw UnsupportedError(
          'DefaultFirebaseOptions have not been configured for linux - '
          'you can reconfigure this by running the FlutterFire CLI again.',
        );
      default:
        throw UnsupportedError(
          'DefaultFirebaseOptions are not supported for this platform.',
        );
    }
  }

  static const FirebaseOptions web = FirebaseOptions(
    apiKey: 'AIzaSyBU7ejJbl_aCI4YOJSW1MOG-7eq_uKJAB8',
    appId: '1:506199705329:web:2ac8023fd48ca01fd2cb50',
    messagingSenderId: '506199705329',
    projectId: 'farmassistor',
    authDomain: 'farmassistor.firebaseapp.com',
    storageBucket: 'farmassistor.appspot.com',
  );

  static const FirebaseOptions android = FirebaseOptions(
    apiKey: 'AIzaSyAsPQAeCBmiiM4u9D6WYyrRERhVqb9dc8c',
    appId: '1:506199705329:android:911723d4651b4a8bd2cb50',
    messagingSenderId: '506199705329',
    projectId: 'farmassistor',
    storageBucket: 'farmassistor.appspot.com',
  );

  static const FirebaseOptions windows = FirebaseOptions(
    apiKey: 'AIzaSyBU7ejJbl_aCI4YOJSW1MOG-7eq_uKJAB8',
    appId: '1:506199705329:web:90a79b6b1f9fad49d2cb50',
    messagingSenderId: '506199705329',
    projectId: 'farmassistor',
    authDomain: 'farmassistor.firebaseapp.com',
```

```
    storageBucket: 'farmassistor.appspot.com',
  );
}
```

Firebase Services File

The `FirebaseAuthService` class handles interactions with Firebase Authentication and Firestore in a Flutter application. It initializes FirebaseAuth and FirebaseFirestore instances for user authentication and database operations. Key methods include:

- `signUpWithEmailAndPassword`: Registers new users and stores their details in Firestore.
- `signInWithEmailAndPassword`: Logs in existing users.
- `saveDeviceInformation`: Stores device-specific data like ID and parameters.

```dart
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:flutter/material.dart';

class FirebaseAuthService {
  FirebaseAuth _auth = FirebaseAuth.instance;
  FirebaseFirestore _firestore = FirebaseFirestore.instance;

  static FirebaseAuthService get instance => FirebaseAuthService();

  String? getCurrentUserId() {
    return _auth.currentUser?.uid;
  }

  Future<void> saveDeviceInformation(String userId, String deviceId, String plantType
    , Map<String, dynamic> deviceInfo) async {
    try {
      await _firestore.collection('users').doc(userId).collection('devices').add({
        'deviceId': deviceId,
        'plantType': plantType,
        'soilMoistureChecked': deviceInfo['soilMoistureChecked'] ?? false,
        'soilTemperatureChecked': deviceInfo['soilTemperatureChecked'] ?? false,
        'soilNutrientChecked': deviceInfo['soilNutrientChecked'] ?? false,
        'airTemperatureChecked': deviceInfo['airTemperatureChecked'] ?? false,
        'humidityChecked': deviceInfo['humidityChecked'] ?? false,
        'createdAt': FieldValue.serverTimestamp(),
      });
    } catch (e) {
      print('Error saving device: $e');
      // Handle error, show error dialog, etc.
    }
  }

  Future<User?> signUpWithEmailAndPassword(String email, String password) async {
    try {
      UserCredential credential = await _auth.createUserWithEmailAndPassword(email:
    email, password: password);
      User? user = credential.user;
      if (user != null) {
        await _firestore.collection('users').doc(user.uid).set({
          'email': email,
          'createdAt': FieldValue.serverTimestamp(),
        });
      }
      return user;
    } on FirebaseAuthException catch (e) {
      print("Some error occurred");
      if (e.code == 'email-already-in-use') {
        showToast(message: 'The email address is already in use.');
      } else {
```

```
        showToast(message: 'An error occurred: ${e.code}');
      }
      return null;
    }
  }

  Future<User?> signInWithEmailAndPassword(String email, String password) async {
    try {
      UserCredential credential = await _auth.signInWithEmailAndPassword(email: email
    , password: password);
      return credential.user;
    } on FirebaseAuthException catch (e) {
      print("Some error occurred");
      if (e.code == 'user-not-found' || e.code == 'wrong-password') {
        showToast(message: 'Invalid email or password.');
      } else {
        showToast(message: 'An error occurred: ${e.code}');
      }
      return null;
    }
  }

  User? getCurrentUser() {
    return _auth.currentUser;
  }

  Future<List<Map<String, dynamic>>> getDevices(String userId) async {
    try {
      QuerySnapshot querySnapshot = await _firestore
          .collection('users')
          .doc(userId)
          .collection('devices')
          .get();

      // Convert QuerySnapshot to List<Map<String, dynamic>>
      List<Map<String, dynamic>> devicesList = querySnapshot.docs.map((doc) => doc.
    data() as Map<String, dynamic>).toList();

      return devicesList;
    } catch (e) {
      print('Error fetching devices: $e');
      return []; // Return an empty list on error
    }
  }

  void showErrorDialog(BuildContext context, String message) {
    showDialog(
      context: context,
      builder: (BuildContext context) {
        return AlertDialog(
          title: Text('Error'),
          content: Text(message),
          actions: [
            TextButton(
              onPressed: () => Navigator.of(context).pop(),
              child: Text('OK'),
            ),
          ],
        );
      },
    );
  }

  void showToast({required String message}) {
    print(message); // Placeholder for actual toast notification
  }
}
```

Splash Screen File

The `SplashScreen` widget serves as an introductory screen for the FarmAssistor application. It extends `StatefulWidget` to handle state changes throughout its lifecycle. In its `initState()` method, a 3-second delay is implemented using `Future.delayed()`. After this delay, the widget navigates to the designated child widget using `Navigator.pushAndRemoveUntil()`. This strategy ensures a smooth transition from the SplashScreen to the subsequent screen, typically the login or home screen of the application.

```
import 'package:flutter/material.dart';

class SplashScreen extends StatefulWidget {
  final Widget? child;
  const SplashScreen({Key? key, this.child});

  @override
  State<SplashScreen> createState() => _SplashScreenState();
}

class _SplashScreenState extends State<SplashScreen> {
  @override
  void initState() {
    Future.delayed(
      Duration(seconds: 3),
      () {
        Navigator.pushAndRemoveUntil(context, MaterialPageRoute(builder: (context) =>
     widget.child!), (route) => false);
      }
    );
    super.initState();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      backgroundColor: Colors.green.shade50,
      body: Center(
        child: Text(
          "FarmAssistor",
          style: TextStyle(
            color: Colors.green.shade900,
            fontWeight: FontWeight.bold,
            fontSize: 30.0,
          ),
        ),
      ),
    );
  }
}
```

Login Screen File

The `loginPage` class implements a user login interface using Firebase Authentication. Its UI includes an app bar, text fields for email and password input, and a login button that triggers the `_signIn` method. This method attempts to authenticate the user with `_auth.signInWithEmailAndPassword` using provided credentials. During authentication, a loading spinner (`CircularProgressIndicator`) indicates progress. Upon successful login, the user navigates to the home page. The "Sign Up" text is clickable, allowing users to navigate to the `signUpPage` for registration.

```
import 'package:firebase/Features/User_auth/FirebaseAuthImplementation/
    FirebaseAuthServices.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/signUpPage.dart';
import 'package:firebase/Features/User_auth/presentations/Widgets/FromContainerWidget
    .dart';
import 'package:firebase_auth/firebase_auth.dart';
```

```dart
import 'package:flutter/material.dart';

class loginPage extends StatefulWidget {
  const loginPage({super.key});

  @override
  State<loginPage> createState() => _loginPageState();
}

class _loginPageState extends State<loginPage> {
  bool _isSigning = false;
  final FirebaseAuthService _auth = FirebaseAuthService();

  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green.shade900,
        title: Text(
          "Login",
          style: TextStyle(color: Colors.white),
        ),
        automaticallyImplyLeading: false,
      ),
      body: Center(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 15),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                "Login",
                style: TextStyle(fontSize: 27, fontWeight: FontWeight.bold),
              ),
              SizedBox(height: 30),
              FormContainerWidget(
                controller: _emailController,
                hintText: "Email",
                isPasswordField: false,
              ),
              SizedBox(height: 10),
              FormContainerWidget(
                controller: _passwordController,
                hintText: "Password",
                isPasswordField: true,
              ),
              SizedBox(height: 30),
              GestureDetector(
                onTap: _signIn,
                child: Container(
                  width: double.infinity,
                  height: 45,
                  decoration: BoxDecoration(
                    color: Colors.green.shade900,
                    borderRadius: BorderRadius.circular(10),
                  ),
                  child: Center(
                    child: _isSigning
```

```
                        ? CircularProgressIndicator(color: Colors.white)
                        : Text(
                            "Login",
                            style: TextStyle(
                                color: Colors.white,
                                fontWeight: FontWeight.bold,
                                fontSize: 20),
                          ),
                    ),
                  ),
                ),
                SizedBox(height: 20),
                Row(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    Text("Don't have an account?"),
                    SizedBox(width: 5),
                    GestureDetector(
                      onTap: () {
                        Navigator.pushAndRemoveUntil(
                          context,
                          MaterialPageRoute(builder: (context) => signUpPage()),
                          (route) => false,
                        );
                      },
                      child: Text(
                        "Sign Up",
                        style: TextStyle(
                          color: Colors.green.shade900,
                          fontWeight: FontWeight.bold,
                        ),
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
      );
  }

  void _signIn() async {
    setState(() {
      _isSigning = true;
    });

    String email = _emailController.text;
    String password = _passwordController.text;

    User? user = await _auth.signInWithEmailAndPassword(email, password);

    setState(() {
      _isSigning = false;
    });

    if (user != null) {
      print("User is successfully signed in");
      Navigator.pushNamed(context, "/home");
    } else {
      print("Some error happened");
    }
  }
}
```

Sign Up Screen File

The `signUpPage` class implements a user registration interface using Firebase Authentication. Its straightforward UI includes an app bar, text fields for entering email and password, and a "Sign Up" button that triggers the `_signUp` method. This method attempts to create a new user account using `_auth.signUpWithEmailAndPassword` with the provided credentials. During registration, a loading spinner (`CircularProgressIndicator`) is displayed while awaiting the process. Upon successful registration, the user is navigated to the home page. The "Login" text is clickable, allowing existing users to navigate to the `loginPage` for signing in.

```dart
import 'package:firebase/Features/User_auth/FirebaseAuthImplementation/
    FirebaseAuthServices.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/loginPage.dart';
import 'package:firebase/Features/User_auth/presentations/Widgets/FromContainerWidget
    .dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

class signUpPage extends StatefulWidget {
  const signUpPage({super.key});

  @override
  State<signUpPage> createState() => _signUpPageState();
}

class _signUpPageState extends State<signUpPage> {
  final FirebaseAuthService _auth = FirebaseAuthService();

  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();
  bool isSigningUp = false;

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        backgroundColor: Colors.green.shade900,
        title: Text(
          "Sign Up",
          style: TextStyle(color: Colors.white),
        ),
        automaticallyImplyLeading: false,
      ),
      body: Center(
        child: Padding(
          padding: const EdgeInsets.symmetric(horizontal: 15),
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                "Sign Up",
                style: TextStyle(fontSize: 27, fontWeight: FontWeight.bold),
              ),
              SizedBox(height: 30),
              FormContainerWidget(
                controller: _emailController,
                hintText: "Email",
                isPasswordField: false,
              ),
              SizedBox(height: 10),
              FormContainerWidget(
                controller: _passwordController,
```

```
                hintText: "Password",
                isPasswordField: true,
              ),
              SizedBox(height: 30),
              GestureDetector(
                onTap: _signUp,
                child: Container(
                  width: double.infinity,
                  height: 45,
                  decoration: BoxDecoration(
                    color: Colors.green.shade900,
                    borderRadius: BorderRadius.circular(10),
                  ),
                  child: Center(
                    child: isSigningUp
                        ? CircularProgressIndicator(color: Colors.white)
                        : Text(
                            "Sign Up",
                            style: TextStyle(
                                color: Colors.white,
                                fontWeight: FontWeight.bold,
                                fontSize: 20),
                          ),
                  ),
                ),
              ),
              SizedBox(height: 20),
              Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Text("Already have an account?"),
                  SizedBox(width: 5),
                  GestureDetector(
                    onTap: () {
                      Navigator.pushAndRemoveUntil(
                        context,
                        MaterialPageRoute(builder: (context) => loginPage()),
                        (route) => false,
                      );
                    },
                    child: Text(
                      "Login",
                      style: TextStyle(
                        color: Colors.green.shade900,
                        fontWeight: FontWeight.bold,
                      ),
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
    );
  }

  void _signUp() async {
    setState(() {
      isSigningUp = true;
    });

    String email = _emailController.text;
    String password = _passwordController.text;

    User? user = await _auth.signUpWithEmailAndPassword(email, password);

    setState(() {
```

```
      isSigningUp = false;
    });

    if (user != null) {
      print("User is successfully created");
      Navigator.pushNamed(context, "/home");
    } else {
      print("Some error happened");
    }
  }
}
```

Home Page File

The `HomePage` class manages a dynamic interface based on selected devices retrieved from Firebase Firestore. During initialization in `initState`, it handles user authentication and fetches device data. The `_fetchDevices` method retrieves devices associated with the current user, updating the UI state with initial device data. `_updateDataForSelectedDevice` fetches specific device data from Firestore based on the selected device's ID, updating local state variables (such as soilMoisture and humidity) and triggering UI updates. The `build` method constructs a `Scaffold` containing a `BottomNavigationBar` for navigation and a dynamic body that alternates between displaying device data and a list of devices based on user interaction. This setup ensures real-time updates of environmental data through gauges and temperature displays, responsive to user selections.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:firebase/Features/User_auth/FirebaseAuthImplementation/
    FirebaseAuthServices.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/dropdown.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/Gauges.dart';
import 'package:firebase/Features/User_auth/presentations/Pages/DevicesPage.dart';

class HomePage extends StatefulWidget {
  const HomePage({Key? key}) : super(key: key);

  @override
  _HomePageState createState() => _HomePageState();
}

class _HomePageState extends State<HomePage> {
  int _selectedIndex = 0;
  Map<String, dynamic>? selectedDevice;
  double soilMoisture = 0.0;
  double humidity = 0.0;
  double soilNutrition = 0.0;
  double airTemperature = 0.0;
  double soilTemperature = 0.0;

  final FirebaseAuthService _authService = FirebaseAuthService();

  @override
  void initState() {
    super.initState();
    _fetchDevices();
  }

  Future<void> _fetchDevices() async {
    String? userId = _authService.getCurrentUserId();
    if (userId != null) {
      try {
        List<Map<String, dynamic>> devices = await _authService.getDevices(userId);
        if (devices.isNotEmpty) {
          setState(() {
            selectedDevice = devices.first;
            _updateDataForSelectedDevice(selectedDevice!); // Fetch initial data
          });
```

```
      }
    } catch (e) {
      print('Error fetching devices: $e');
    }
  }
}

void _onDeviceChanged(Map<String, dynamic> newDevice) {
  setState(() {
    selectedDevice = newDevice;
    _updateDataForSelectedDevice(newDevice);
  });
}

Future<void> _updateDataForSelectedDevice(Map<String, dynamic> device) async {
  try {
    // Simulating data fetching from Firebase based on device ID
    // Replace with actual Firebase logic based on your data structure
    // Example: DocumentReference docRef = FirebaseFirestore.instance.collection('devices').doc(device['deviceId']);
    // DocumentSnapshot snapshot = await docRef.get();

    // Example data retrieval (replace with actual Firebase data retrieval)
    Map<String, dynamic> data = {
      'soilMoisture': 0.85,
      'humidity': 55.0,
      'soilNutrient': 0.76,
      'airTemperature': 30,
      'soilTemperature': 30,
    };

    setState(() {
      soilMoisture = data['soilMoisture'];
      humidity = data['humidity'];
      soilNutrition = data['soilNutrient'];
      airTemperature = data['airTemperature'];
      soilTemperature = data['soilTemperature'];
    });
  } catch (e) {
    print('Error fetching device data: $e');
  }
}

void _onItemTapped(int index) {
  setState(() {
    _selectedIndex = index;
  });
}

@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: _selectedIndex == 0
        ? AppBar(
            backgroundColor: Colors.green.shade900,
            title: Text(
              "Home Page",
              style: TextStyle(color: Colors.white),
            ),
            actions: [
              IconButton(
                icon: Icon(Icons.logout, color: Colors.white),
                onPressed: () {
                  FirebaseAuth.instance.signOut();
                  Navigator.pushNamed(context, "/login");
                },
              ),
            ],
```

```
            automaticallyImplyLeading: false, // Removes the back button
          )
        : null,
    body: _selectedIndex == 0
        ? SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                SizedBox(height: 30),
                Padding(
                  padding: const EdgeInsets.symmetric(horizontal: 16.0),
                  child: PlantTypeDropdown(
                    onPlantTypeChanged: _onDeviceChanged,
                  ),
                ),
                SizedBox(height: 20),
                Container(
                  padding: EdgeInsets.symmetric(horizontal: 16.0),
                  child: Column(
                    crossAxisAlignment: CrossAxisAlignment.stretch,
                    children: [
                      Row(
                        mainAxisAlignment: MainAxisAlignment.spaceBetween,
                        children: [
                          _buildTemperatureBox('Air Temperature', airTemperature),
                          _buildTemperatureBox('Soil Temperature', soilTemperature)
,
                        ],
                      ),
                      SizedBox(height: 20),
                      SoilMoistureGauge(value: soilMoisture),
                      SizedBox(height: 20),
                      HumidityGauge(value: humidity),
                      SizedBox(height: 20),
                      SoilNutritionGauge(value: soilNutrition),
                      SizedBox(height: 20),
                    ],
                  ),
                ),
              ],
            ),
          )
        : DevicesPage(),
    bottomNavigationBar: BottomNavigationBar(
      items: const <BottomNavigationBarItem>[
        BottomNavigationBarItem(
          icon: Icon(Icons.home),
          label: 'Home',
        ),
        BottomNavigationBarItem(
          icon: Icon(Icons.devices),
          label: 'Devices',
        ),
      ],
      currentIndex: _selectedIndex,
      selectedItemColor: Colors.green.shade900,
      onTap: _onItemTapped,
    ),
  );
}

Widget _buildTemperatureBox(String title, double temperature) {
  return Expanded(
    child: Container(
      padding: EdgeInsets.all(12.0),
      decoration: BoxDecoration(
        border: Border.all(color: Colors.grey),
        borderRadius: BorderRadius.circular(8.0),
```

```
        ),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            Text(
              title,
              style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
            ),
            SizedBox(height: 8.0),
            Text(
              '${temperature.toStringAsFixed(1)}  C ',
              style: TextStyle(fontSize: 18),
            ),
          ],
        ),
      ),
    );
  }
}
```

Devices Page File

The `DevicesPage` class in Flutter manages a screen that displays a list of devices linked to the current user. It utilizes Firebase authentication (`FirebaseAuth`) and Firestore (`FirebaseFirestore`) for user login and data storage respectively. Upon initialization (`initState`), it retrieves the user's devices from Firestore using `_fetchDevices`. The UI includes an `AppBar` with a title ("Devices") and a button ("+ Add a device") for navigating to the `AddDeviceScreen`. Each device is listed using `ListView.builder`, where a `Card` displays details like Device ID, Plant Type, and checkboxes for parameters such as Soil Moisture, Soil Temperature, Soil Nutrition, Air Temperature, and Humidity. Each device entry includes a delete button (`IconButton`) that triggers a confirmation dialog (`AlertDialog`) before deleting the device using `_deleteDevice`, which removes the corresponding Firestore document. The `_showAddDeviceScreen` method manages navigation to `AddDeviceScreen`, updating the device list upon adding a new device. Overall, `DevicesPage` enables Create, Read, Update, Delete operations for managing user devices in Firestore.

```
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'add_device.dart';

class DevicesPage extends StatefulWidget {
  const DevicesPage({Key? key}) : super(key: key);

  @override
  _DevicesPageState createState() => _DevicesPageState();
}

class _DevicesPageState extends State<DevicesPage> {
  final FirebaseAuth _auth = FirebaseAuth.instance;
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  late User? _currentUser;
  late String _userId;
  List<Map<String, dynamic>> _devices = [];

  @override
  void initState() {
    super.initState();
    _currentUser = _auth.currentUser;
    if (_currentUser != null) {
      _userId = _currentUser!.uid;
      _fetchDevices();
    }
  }
```

```
30
31  Future<void> _fetchDevices() async {
32    try {
33      QuerySnapshot querySnapshot = await _firestore
34          .collection('users')
35          .doc(_userId)
36          .collection('devices')
37          .get();
38
39      setState(() {
40        _devices = querySnapshot.docs
41            .map((doc) => doc.data() as Map<String, dynamic>)
42            .toList();
43      });
44    } catch (e) {
45      print('Error fetching devices: $e');
46    }
47  }
48
49  Future<void> _deleteDevice(String deviceId) async {
50    try {
51      await _firestore
52          .collection('users')
53          .doc(_userId)
54          .collection('devices')
55          .where('deviceId', isEqualTo: deviceId)
56          .get()
57          .then((QuerySnapshot querySnapshot) {
58        querySnapshot.docs.forEach((doc) {
59          doc.reference.delete();
60        });
61      });
62      await _fetchDevices();
63    } catch (e) {
64      print('Error deleting device: $e');
65    }
66  }
67
68  @override
69  Widget build(BuildContext context) {
70    return Scaffold(
71      appBar: AppBar(
72        backgroundColor: Colors.green.shade900,
73        title: Text("Devices", style: TextStyle(color: Colors.white)),
74        automaticallyImplyLeading: false,
75      ),
76      body: Column(
77        crossAxisAlignment: CrossAxisAlignment.stretch,
78        children: [
79          Padding(
80            padding: const EdgeInsets.symmetric(vertical: 20),
81            child: Center(
82              child: ElevatedButton(
83                onPressed: () {
84                  _showAddDeviceScreen(context);
85                },
86                style: ButtonStyle(
87                  backgroundColor: MaterialStateProperty.all<Color>(
88                    Colors.green.shade900,
89                  ),
90                  foregroundColor: MaterialStateProperty.all<Color>(
91                    Colors.white,
92                  ),
93                  padding: MaterialStateProperty.all<EdgeInsetsGeometry>(
94                    EdgeInsets.symmetric(horizontal: 20, vertical: 16),
95                  ),
96                  shape: MaterialStateProperty.all<OutlinedBorder>(
97                    RoundedRectangleBorder(
```

```
                    borderRadius: BorderRadius.circular(8),
                  ),
                ),
              ),
              child: Text(
                "+ Add a device",
                style: TextStyle(fontSize: 18),
              ),
            ),
          ),
        ),
        Expanded(
          child: ListView.builder(
            itemCount: _devices.length,
            itemBuilder: (context, index) {
              final device = _devices[index];
              String deviceId = device['deviceId'];
              String plantType = device['plantType'];
              int soilMoistureChecked = device['soilMoistureChecked'] ?? 0;
              int soilTemperatureChecked = device['soilTemperatureChecked'] ?? 0;
              int soilNutrientChecked = device['soilNutrientChecked'] ?? 0;
              int airTemperatureChecked = device['airTemperatureChecked'] ?? 0;
              int humidityChecked = device['humidityChecked'] ?? 0;

              return Card(
                margin: EdgeInsets.symmetric(vertical: 8, horizontal: 16),
                elevation: 4,
                child: ListTile(
                  title: Column(
                    crossAxisAlignment: CrossAxisAlignment.start,
                    children: [
                      Text('Device ID: $deviceId'),
                      Text('Plant Type: $plantType'),
                      Text('Soil Moisture: $soilMoistureChecked %'),
                      Text('Soil Nutrition: $soilNutrientChecked %'),
                      Text('Soil Temperature: $soilTemperatureChecked  C '),
                      Text('Humidity: $humidityChecked %'),
                      Text('Air Temperature: $airTemperatureChecked  C '),
                    ],
                  ),
                  trailing: IconButton(
                    icon: Icon(Icons.delete),
                    onPressed: () {
                      showDialog(
                        context: context,
                        builder: (BuildContext context) {
                          return AlertDialog(
                            title: Text('Delete Device'),
                            content: Text(
                                'Are you sure you want to delete device $deviceId?'
),
                            actions: [
                              TextButton(
                                onPressed: () {
                                  Navigator.of(context).pop();
                                },
                                child: Text('Cancel'),
                              ),
                              TextButton(
                                onPressed: () {
                                  Navigator.of(context).pop();
                                  _deleteDevice(deviceId);
                                },
                                child: Text('Delete'),
                              ),
                            ],
                          );
                        },
```

```
                        );
                      },
                    ),
                  ),
                );
              },
            ),
          ),
        ],
      ),
    );
  }

  Future<void> _showAddDeviceScreen(BuildContext context) async {
    final result = await Navigator.push(
      context,
      MaterialPageRoute(builder: (context) => AddDeviceScreen()),
    );

    if (result != null) {
      String deviceId = result['deviceId'];
      String plantType = result['plantType'];
      int soilMoistureChecked = result['soilMoistureChecked'];
      int soilTemperatureChecked = result['soilTemperatureChecked'];
      int soilNutrientChecked = result['soilNutrientChecked'];
      int airTemperatureChecked = result['airTemperatureChecked'];
      int humidityChecked = result['humidityChecked'];

      try {
        await _firestore
            .collection('users')
            .doc(_userId)
            .collection('devices')
            .add({
          'deviceId': deviceId,
          'plantType': plantType,
          'soilMoistureChecked': soilMoistureChecked,
          'soilTemperatureChecked': soilTemperatureChecked,
          'soilNutrientChecked': soilNutrientChecked,
          'airTemperatureChecked': airTemperatureChecked,
          'humidityChecked': humidityChecked,
        });

        await _fetchDevices();
      } catch (e) {
        print('Error adding device: $e');
      }
    }
  }
}
```

Add Device Page File

The `AddDeviceScreen` class works as a screen designed for adding a new device configuration. It features `TextFields` for entering device ID and plant type, incorporating validation mechanisms to prevent empty submissions (tracked by `deviceIdError` and `plantTypeError`). Checkboxes (`CheckboxListTile`) are provided for selecting parameters to monitor, such as soil moisture, soil temperature, soil nutrient, air temperature, and humidity. The state of each checkbox is managed using boolean variables (`soilMoistureChecked`, `soilTemperatureChecked`, etc.), while the `updateAddButtonEnabledState` method dynamically enables or disables the "Add" button (`ElevatedButton`) based on the validation of text fields and the selection of at least one parameter. Additionally, the `_convertBoolToNumeric` method facilitates the conversion of boolean values to numeric (0 or 1) for storage or processing purposes. Overall, this screen ensures user inputs are validated and parameter selections are confirmed before allowing the addition of a new

device configuration.

```dart
import 'package:flutter/material.dart';

class AddDeviceScreen extends StatefulWidget {
  const AddDeviceScreen({Key? key}) : super(key: key);

  @override
  _AddDeviceScreenState createState() => _AddDeviceScreenState();
}

class _AddDeviceScreenState extends State<AddDeviceScreen> {
  bool soilMoistureChecked = true;
  bool soilTemperatureChecked = true;
  bool soilNutrientChecked = true;
  bool airTemperatureChecked = true;
  bool humidityChecked = true;

  final TextEditingController deviceIdController = TextEditingController();
  final TextEditingController plantTypeController = TextEditingController();

  String? deviceIdError;
  String? plantTypeError;

  bool isAddButtonEnabled = false;

  @override
  void initState() {
    super.initState();
    updateAddButtonEnabledState();
  }

  void updateAddButtonEnabledState() {
    bool isAtLeastOneCheckboxChecked =
        soilMoistureChecked ||
        soilTemperatureChecked ||
        soilNutrientChecked ||
        airTemperatureChecked ||
        humidityChecked;
    bool isFieldsNotEmpty =
        deviceIdController.text.isNotEmpty &&
        plantTypeController.text.isNotEmpty;
    setState(() {
      isAddButtonEnabled = isAtLeastOneCheckboxChecked && isFieldsNotEmpty;
    });
  }

  int _convertBoolToNumeric(bool value) {
    return value ? 1 : 0;
  }

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Add Device'),
        automaticallyImplyLeading: false,
      ),
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16.0),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: <Widget>[
            TextField(
              controller: deviceIdController,
              onChanged: (value) {
                setState(() {
                  deviceIdError =
```

```
                    value.isEmpty ? 'Device ID cannot be empty' : null;
                updateAddButtonEnabledState();
              });
            },
            decoration: InputDecoration(
              labelText: 'Device ID',
              errorText: deviceIdError,
            ),
          ),
          SizedBox(height: 30),
          TextField(
            controller: plantTypeController,
            onChanged: (value) {
              setState(() {
                plantTypeError =
                    value.isEmpty ? 'Plant Type cannot be empty' : null;
                updateAddButtonEnabledState();
              });
            },
            decoration: InputDecoration(
              labelText: 'Plant Type',
              errorText: plantTypeError,
            ),
          ),
          SizedBox(height: 60),
          Text(
            'Parameters to measure:',
            style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
          ),
          CheckboxListTile(
            title: Text('Soil Moisture Level'),
            value: soilMoistureChecked,
            onChanged: (value) {
              setState(() {
                soilMoistureChecked = value ?? false;
                updateAddButtonEnabledState();
              });
            },
          ),
          CheckboxListTile(
            title: Text('Soil Temperature'),
            value: soilTemperatureChecked,
            onChanged: (value) {
              setState(() {
                soilTemperatureChecked = value ?? false;
                updateAddButtonEnabledState();
              });
            },
          ),
          CheckboxListTile(
            title: Text('Soil Nutrient Level'),
            value: soilNutrientChecked,
            onChanged: (value) {
              setState(() {
                soilNutrientChecked = value ?? false;
                updateAddButtonEnabledState();
              });
            },
          ),
          CheckboxListTile(
            title: Text('Air Temperature'),
            value: airTemperatureChecked,
            onChanged: (value) {
              setState(() {
                airTemperatureChecked = value ?? false;
                updateAddButtonEnabledState();
              });
            },
```

```
            ),
            CheckboxListTile(
              title: Text('Humidity'),
              value: humidityChecked,
              onChanged: (value) {
                setState(() {
                  humidityChecked = value ?? false;
                  updateAddButtonEnabledState();
                });
              },
            ),
            SizedBox(height: 20),
            Row(
              mainAxisAlignment: MainAxisAlignment.spaceBetween,
              children: <Widget>[
                ElevatedButton(
                  onPressed: () {
                    Navigator.pop(context);
                  },
                  style: ButtonStyle(
                    backgroundColor:
                        MaterialStateProperty.all<Color>(Colors.red),
                    foregroundColor:
                        MaterialStateProperty.all<Color>(Colors.white),
                    padding: MaterialStateProperty.all<EdgeInsetsGeometry>(
                      EdgeInsets.symmetric(horizontal: 20, vertical: 16),
                    ),
                  ),
                  child: Text('Cancel'),
                ),
                ElevatedButton(
                  onPressed: isAddButtonEnabled
                      ? () {
                          Navigator.pop(context, {
                            'deviceId': deviceIdController.text,
                            'plantType': plantTypeController.text,
                            'soilMoistureChecked':
                                _convertBoolToNumeric(soilMoistureChecked),
                            'soilTemperatureChecked':
                                _convertBoolToNumeric(soilTemperatureChecked),
                            'soilNutrientChecked':
                                _convertBoolToNumeric(soilNutrientChecked),
                            'airTemperatureChecked':
                                _convertBoolToNumeric(airTemperatureChecked),
                            'humidityChecked':
                                _convertBoolToNumeric(humidityChecked),
                          });
                        }
                      : null,
                  style: ButtonStyle(
                    backgroundColor:
                        MaterialStateProperty.resolveWith<Color>(
                            (Set<MaterialState> states) {
                      if (states.contains(MaterialState.disabled)) {
                        return Colors.grey;
                      }
                      return Colors.green.shade900;
                    }),
                    foregroundColor:
                        MaterialStateProperty.all<Color>(Colors.white),
                    padding: MaterialStateProperty.all<EdgeInsetsGeometry>(
                      EdgeInsets.symmetric(horizontal: 20, vertical: 16),
                    ),
                  ),
                  child: Text('Add'),
                ),
              ],
            ),
```

```
          ],
        ),
      ),
    );
  }
}
```

Gauges File

These widgets use the Syncfusion Flutter Gauges library to display circular gauges for soil moisture, soil nutrition, and humidity levels. Each gauge is customized with specific ranges and annotations to visualize the respective data effectively.

```dart
import 'package:flutter/material.dart';
import 'package:syncfusion_flutter_gauges/gauges.dart';

class SoilMoistureGauge extends StatelessWidget {
  final double value;

  const SoilMoistureGauge({Key? key, required this.value}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return CircularGauge(
      value: value * 100,
      title: 'Soil Moisture Level',
      unit: '%',
      ranges: [
        GaugeRange(startValue: 0, endValue: 33, color: Colors.red),
        GaugeRange(startValue: 33, endValue: 66, color: Colors.yellow),
        GaugeRange(startValue: 66, endValue: 100, color: Colors.green),
      ],
    );
  }
}

class SoilNutritionGauge extends StatelessWidget {
  final double value;

  const SoilNutritionGauge({Key? key, required this.value}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return CircularGauge(
      value: value * 100,
      title: 'Soil Nutrition Content',
      unit: '%',
      ranges: [
        GaugeRange(startValue: 0, endValue: 33, color: Colors.red),
        GaugeRange(startValue: 33, endValue: 66, color: Colors.yellow),
        GaugeRange(startValue: 66, endValue: 100, color: Colors.green),
      ],
    );
  }
}

class HumidityGauge extends StatelessWidget {
  final double value;

  const HumidityGauge({Key? key, required this.value}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return CircularGauge(
      value: value,
      title: 'Humidity',
      unit: '%',
```

```
      ranges: [
        GaugeRange(startValue: 0, endValue: 33, color: Colors.red),
        GaugeRange(startValue: 33, endValue: 66, color: Colors.yellow),
        GaugeRange(startValue: 66, endValue: 100, color: Colors.green),
      ],
    );
  }
}

class CircularGauge extends StatelessWidget {
  final double value;
  final String title;
  final String unit;
  final List<GaugeRange> ranges;

  const CircularGauge({
    Key? key,
    required this.value,
    required this.title,
    required this.unit,
    required this.ranges,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return SfRadialGauge(
      axes: <RadialAxis>[
        RadialAxis(
          minimum: 0,
          maximum: 100,
          pointers: <GaugePointer>[
            NeedlePointer(value: value),
          ],
          ranges: ranges,
          annotations: <GaugeAnnotation>[
            GaugeAnnotation(
              widget: Text(
                '${value.toStringAsFixed(1)} $unit',
                style: TextStyle(fontSize: 20, fontWeight: FontWeight.bold),
              ),
              angle: 90,
              positionFactor: 0.5,
            ),
            GaugeAnnotation(
              widget: Text(
                title,
                style: TextStyle(fontSize: 16, fontWeight: FontWeight.bold),
              ),
              angle: 90,
              positionFactor: 1.1,
            ),
          ],
        ),
      ],
    );
  }
}
```

Dropdown File

This widget integrates a dropdown menu to select a plant type based on available devices fetched from Firestore. It uses `FirebaseAuthService` to retrieve devices associated with the current user, updating the dropdown list accordingly. The selected device's details are passed back via the `onPlantTypeChanged` callback for further handling in the parent widget.

```
import 'package:flutter/material.dart';
import 'package:firebase/Features/User_auth/FirebaseAuthImplementation/
```

```dart
    FirebaseAuthServices.dart';

class PlantTypeDropdown extends StatefulWidget {
  final ValueChanged<Map<String, dynamic>> onPlantTypeChanged;

  const PlantTypeDropdown({Key? key, required this.onPlantTypeChanged}) : super(key:
    key);

  @override
  _PlantTypeDropdownState createState() => _PlantTypeDropdownState();
}

class _PlantTypeDropdownState extends State<PlantTypeDropdown> {
  FirebaseAuthService _authService = FirebaseAuthService();
  List<Map<String, dynamic>> devices = [];
  Map<String, dynamic>? selectedDevice;

  @override
  void initState() {
    super.initState();
    _fetchDevices();
  }

  Future<void> _fetchDevices() async {
    String? userId = _authService.getCurrentUserId();
    if (userId != null) {
      try {
        devices = await _authService.getDevices(userId);
        setState(() {
          selectedDevice = devices.isNotEmpty ? devices.first : null;
          if (selectedDevice != null) {
            widget.onPlantTypeChanged(selectedDevice!);
          }
        });
      } catch (e) {
        print('Error fetching devices: $e');
      }
    }
  }

  @override
  Widget build(BuildContext context) {
    return DropdownButtonFormField<Map<String, dynamic>>(
      value: selectedDevice,
      icon: Icon(Icons.arrow_drop_down),
      iconEnabledColor: Colors.black,
      style: TextStyle(color: Colors.black),
      dropdownColor: Color.fromARGB(255, 234, 249, 235),
      decoration: InputDecoration(
        contentPadding: EdgeInsets.symmetric(horizontal: 16.0, vertical: 12.0),
        enabledBorder: OutlineInputBorder(
          borderSide: BorderSide(color: Colors.green.shade900),
          borderRadius: BorderRadius.circular(8.0),
        ),
        focusedBorder: OutlineInputBorder(
          borderSide: BorderSide(color: Colors.green.shade900),
          borderRadius: BorderRadius.circular(8.0),
        ),
      ),
      items: devices.map((device) {
        return DropdownMenuItem<Map<String, dynamic>>(
          value: device,
          child: Text(device['deviceId'] ?? ''),
        );
      }).toList(),
      onChanged: (Map<String, dynamic>? newValue) {
        setState(() {
          selectedDevice = newValue;
```

```
          if (newValue != null) {
            widget.onPlantTypeChanged(newValue);
          }
        });
      },
    );
  }
}
```

Display Devices File

The `FirestoreService` class manages interactions with Firestore to add devices for specific users. It initializes a `FirebaseFirestore` instance `_firestore` to handle Firestore operations. The `addDevice` method within this class takes `userId` and `deviceInfo` (a map containing device details) as parameters. Inside a try block, it attempts to add a new document to the Firestore collection `users`. The document fields include `deviceId`, `plantType`, and various boolean flags (`soilMoistureChecked`, `soilTemperatureChecked`, `soilNutrientChecked`, `airTemperatureChecked`, `humidityChecked`), which are converted to numeric values (1 for true, 0 for false).

```
import 'package:cloud_firestore/cloud_firestore.dart';

class FirestoreService {
  final FirebaseFirestore _firestore = FirebaseFirestore.instance;

  Future<void> addDevice(String userId, Map<String, dynamic> deviceInfo) async {
    try {
      await _firestore.collection('users').doc(userId).collection('devices').add({
        'deviceId': deviceInfo['deviceId'],
        'plantType': deviceInfo['plantType'],
        'soilMoistureChecked': deviceInfo['soilMoistureChecked'] ? 1 : 0,
        'soilTemperatureChecked': deviceInfo['soilTemperatureChecked'] ? 1 : 0,
        'soilNutrientChecked': deviceInfo['soilNutrientChecked'] ? 1 : 0,
        'airTemperatureChecked': deviceInfo['airTemperatureChecked'] ? 1 : 0,
        'humidityChecked': deviceInfo['humidityChecked'] ? 1 : 0,
        'createdAt': FieldValue.serverTimestamp(),
      });
    } catch (e) {
      print("Error adding device: $e");
    }
  }
}
```

pubspec.yaml File

The `pubspec.yaml` file sets up a Flutter project named "FarmAssistor". It specifies the package's name and description, with a version number of 1.0.0. The environment settings mandate Dart SDK version 3.1.0. The dependencies section lists required packages, such as Flutter SDK, firebase_core for Firebase core functionalities, firebase_auth for authentication services, cloud_firestore for Firestore database access, and syncfusion_flutter_gauges for gauge visualizations. The dev_dependencies section includes linting and testing tools, specifically `flutter_lint` and `flutter_test`. Additionally, asset management includes a reference to the `cupertino_icons` package.

```
name: firebase
description: "A new Flutter project."
# The following line prevents the package from being accidentally published to
# pub.dev using `flutter pub publish`. This is preferred for private packages.
publish_to: 'none' # Remove this line if you wish to publish to pub.dev

# The following defines the version and build number for your application.
# A version number is three numbers separated by dots, like 1.2.43
# followed by an optional build number separated by a +.
# Both the version and the builder number may be overridden in flutter
```

```
# build by specifying --build-name and --build-number, respectively.
# In Android, build-name is used as versionName while build-number used as versionCode.
# Read more about Android versioning at https://developer.android.com/studio/publish/versioning
# In iOS, build-name is used as CFBundleShortVersionString while build-number is used as CFBundleVersion.
# Read more about iOS versioning at
# https://developer.apple.com/library/archive/documentation/General/Reference/InfoPlistKeyReference/Articles/CoreFoundationKeys.html
# In Windows, build-name is used as the major, minor, and patch parts
# of the product and file versions while build-number is used as the build suffix.
version: 1.0.0+1

environment:
  sdk: '>=3.3.4 <4.0.0'

# Dependencies specify other packages that your package needs in order to work.
# To automatically upgrade your package dependencies to the latest versions
# consider running `flutter pub upgrade --major-versions`. Alternatively,
# dependencies can be manually updated by changing the version numbers below to
# the latest version available on pub.dev. To see which dependencies have newer
# versions available, run `flutter pub outdated`.
dependencies:
  flutter:
    sdk: flutter


  # The following adds the Cupertino Icons font to your application.
  # Use with the CupertinoIcons class for iOS style icons.
  cupertino_icons: ^1.0.6
  firebase_core: ^3.1.0
  firebase_auth: ^5.1.0
  cloud_firestore: ^5.0.1
  firebase_storage: ^12.0.1
  fluttertoast: ^8.2.2
  syncfusion_flutter_gauges: ^20.2.49




dev_dependencies:
  flutter_test:
    sdk: flutter

  # The "flutter_lints" package below contains a set of recommended lints to
  # encourage good coding practices. The lint set provided by the package is
  # activated in the `analysis_options.yaml` file located at the root of your
  # package. See that file for information about deactivating specific lint
  # rules and activating additional ones.
  flutter_lints: ^3.0.0

# For information on the generic Dart part of this file, see the
# following page: https://dart.dev/tools/pub/pubspec

# The following section is specific to Flutter packages.
flutter:

  # The following line ensures that the Material Icons font is
  # included with your application, so that you can use the icons in
  # the material Icons class.
  uses-material-design: true

  # To add assets to your application, add an assets section, like this:
  # assets:
  #   - images/a_dot_burr.jpeg
  #   - images/a_dot_ham.jpeg

```

```
  # An image asset can refer to one or more resolution-specific "variants", see
  # https://flutter.dev/assets-and-images/#resolution-aware

  # For details regarding adding assets from package dependencies, see
  # https://flutter.dev/assets-and-images/#from-packages

  # To add custom fonts to your application, add a fonts section here,
  # in this "flutter" section. Each entry in this list should have a
  # "family" key with the font family name, and a "fonts" key with a
  # list giving the asset and other descriptors for the font. For
  # example:
  # fonts:
  #   - family: Schyler
  #     fonts:
  #       - asset: fonts/Schyler-Regular.ttf
  #       - asset: fonts/Schyler-Italic.ttf
  #         style: italic
  #   - family: Trajan Pro
  #     fonts:
  #       - asset: fonts/TrajanPro.ttf
  #       - asset: fonts/TrajanPro_Bold.ttf
  #         weight: 700
  #
  # For details regarding fonts from package dependencies,
  # see https://flutter.dev/custom-fonts/#from-packages
```

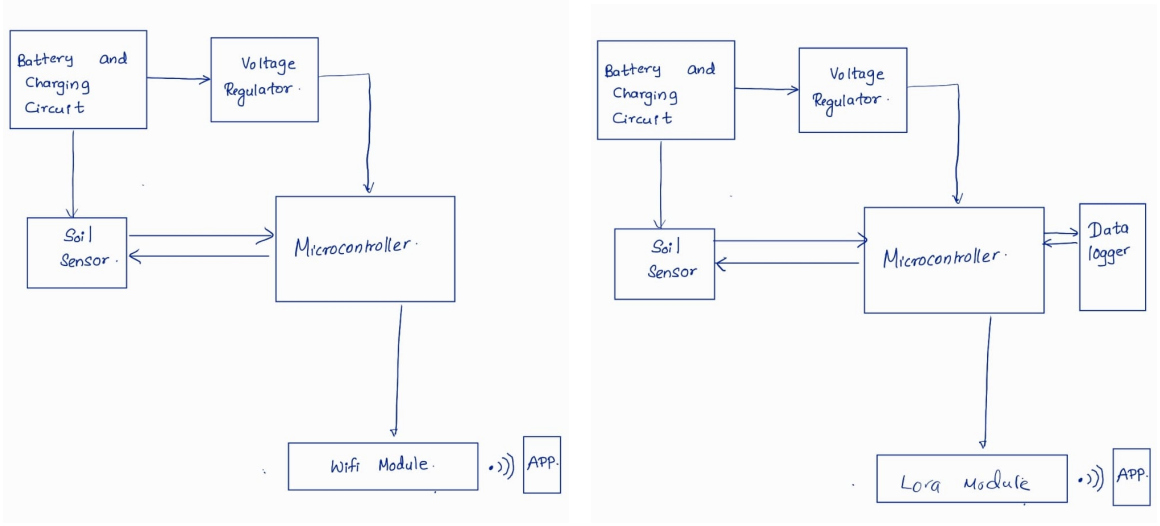
10 Daily Log Entries

1st Feb 2024
After examining the projects given and other possibilities, we have selected Farm Assistor as our project, which is a device that can monitor the soil condition in a farming environment and provide useful information to the farmer. Initially we have planned to measure soil moisture level, temperature, nitrogen, phosphorous, potassium content of the soil. We have proposed Farm Assistor as our first project idea with these concepts.

5th Feb 2024 - 11th Feb 2024
We have analyzed the project idea and identified the key scientific, mathematical, and engineering concepts we will be using in the course of project implementation.

12th Feb 2024 - 18th Feb 2024
A comprehensive study of the stakeholders and existing similar products was carried out. These findings are listed in the observe users section and stakeholder map section. Based on the observations we have chosen three parameters soil moisture level, temperature and conductivity to be the most important parameters relevant to soil health. We decided to move forward in the project with these three requirements.

4th March 2024 - 17th March 2024
During these two weeks we have explored numerourous possible conceptual designs for the product. Here are few conceptual designs which were discarded due to the lack of feasibility compared to the finally evaluated conceptual designs given in section conceptual designs.



(a) Rough Design 1 (b) Rough Design 2

Figure 48: Initial Conceptual Designs

Here the rough design 1 was rejected because i doesn't have a data logger. Since this product should be field deployable, we need to pay special attention to the effective power usage. Since there is no urgent need to provide real time information, we can put the device in sleep mode and take sensor readings at appropriate intervals. Then these readings can be stored and transmitted at appropriate time intervals, this function plays a crucial role as Wi-Fi module draws significant amount of current. Therefore we have rejected designs without a datalogger but needs Wi-Fi transmission.

In rough design 2 we have considered using LoRa as a replacement to Wi-Fi. This idea was rejected because we need to buy LoRa gateways which will increase the cost when implementing our product. Further we had some doubts regarding the existense of appropriate infrastructure in Sri Lanka to implement this design even in rural areas. Due to these concerns these two designs were rejected without further evaluations.
We have comprehensively evaluated the designs under conceptual design section and selected the best design. Evaluation information is specified under evaluation section.

18th March 2024 - 24th March 2024

During this week we have studied the selected conceptual designs in detail to design the required circuitry. As the first step we have evaluated possible implementations to the functional block diagram. After getting the general idea of the implementation we have selected the main components for the circuit. Details of the component selection is given under section 14.1.1 under comprehensive design details.
Due to time limitations and extended delivery time we ordered the selected sensor from it's manufacturer through AliExpress online platform.
And the availability of other components was checked using the Mouser website.

For our mobile app development, we chose Firebase because it's open-source, scalable, and provides strong authentication services and a real-time database. We considered other platforms like AWS Amplify and Backendless but found them too complex and lacking in community support. For the UI design, we went with Flutter over React Native due to its better performance, consistent UI across different platforms, and its extensive set of pre-designed widgets.

25th March 2024 - 31st March 2024
During this week we have come up with the initial schematic designs.

Figure 49: Main - Revision 1.0

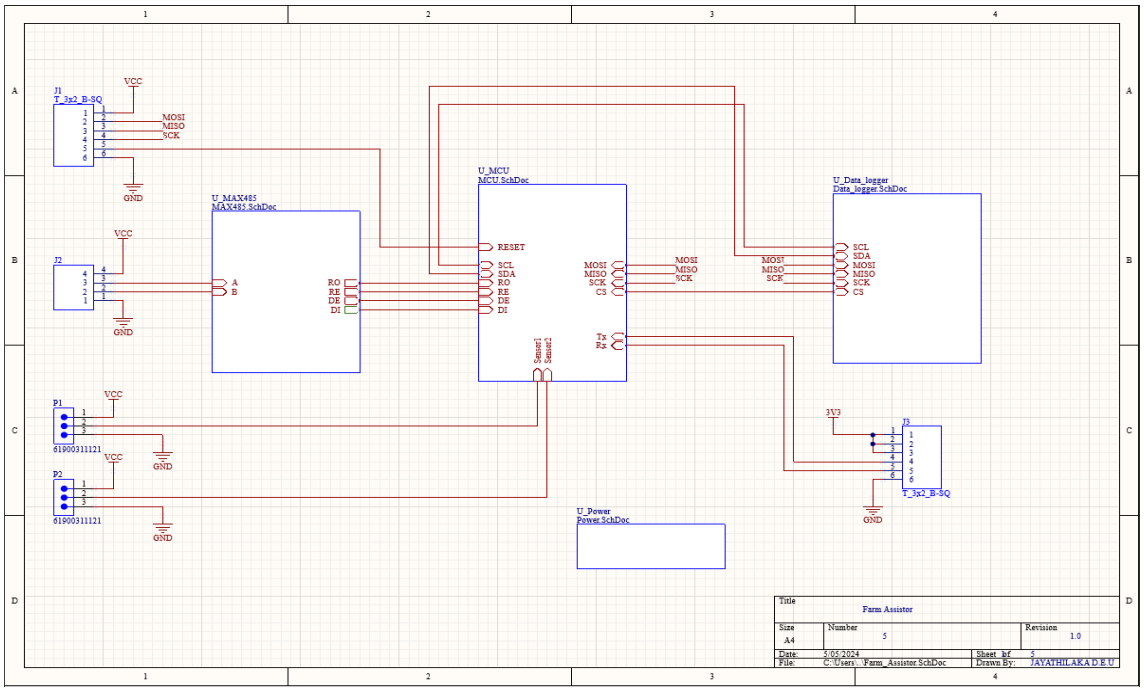


Figure 50: MCU - Revision 1.0

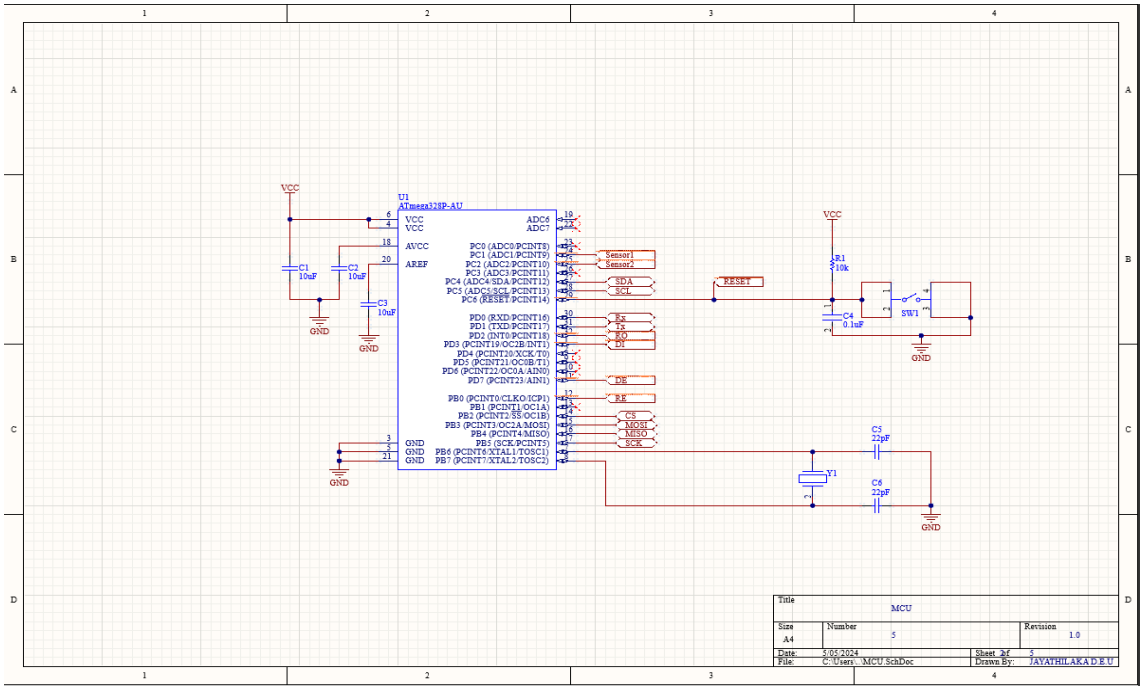


Figure 51: Power - Revision 1.0

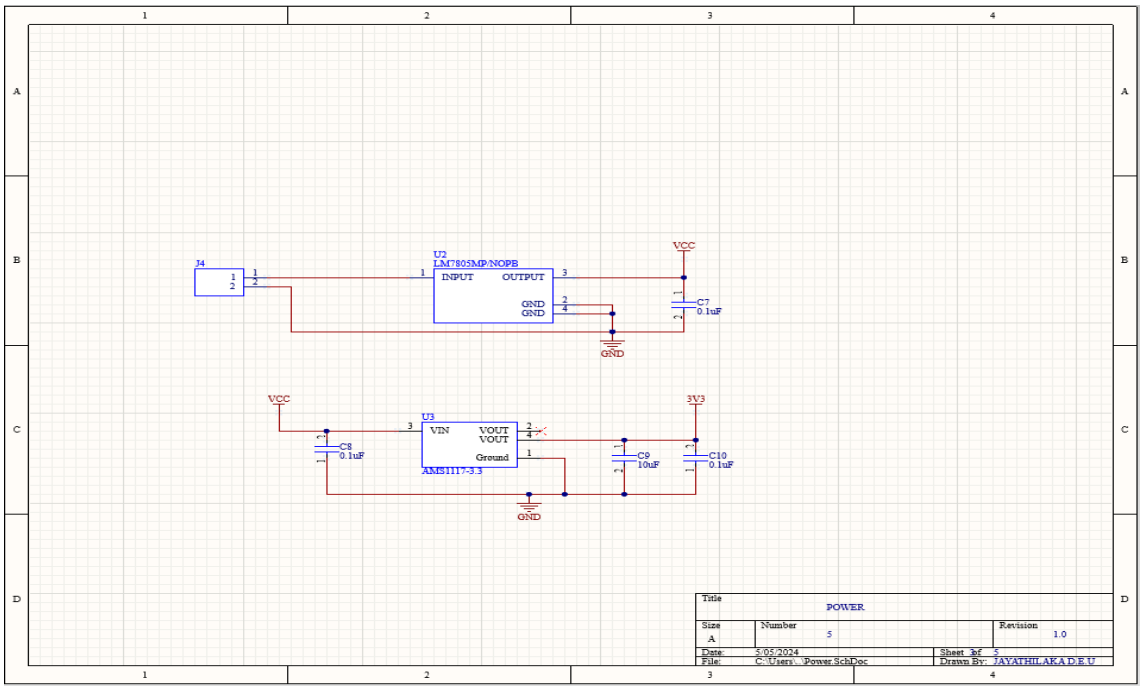


Figure 52: Sensor Inputs - Revision 1.0

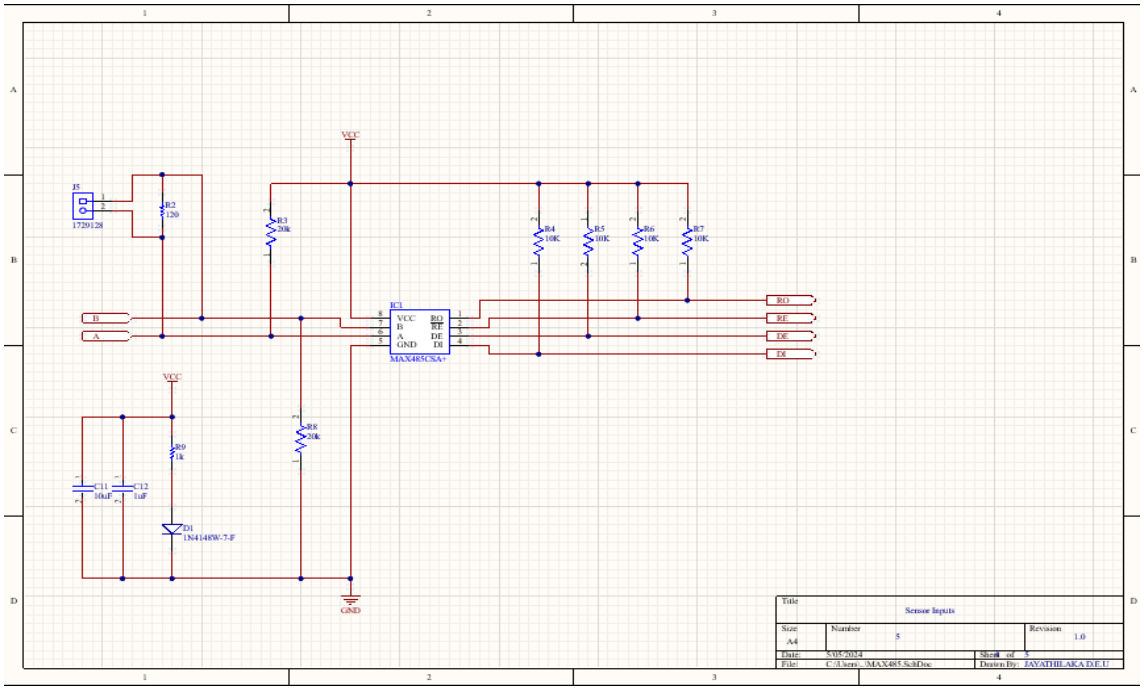
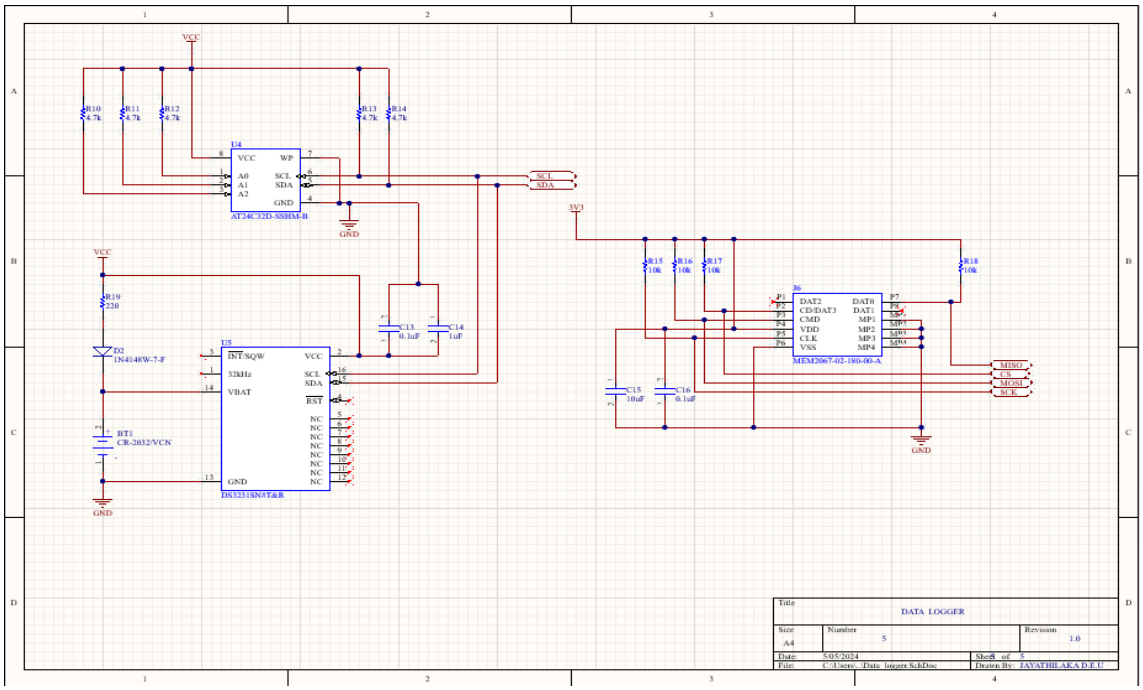
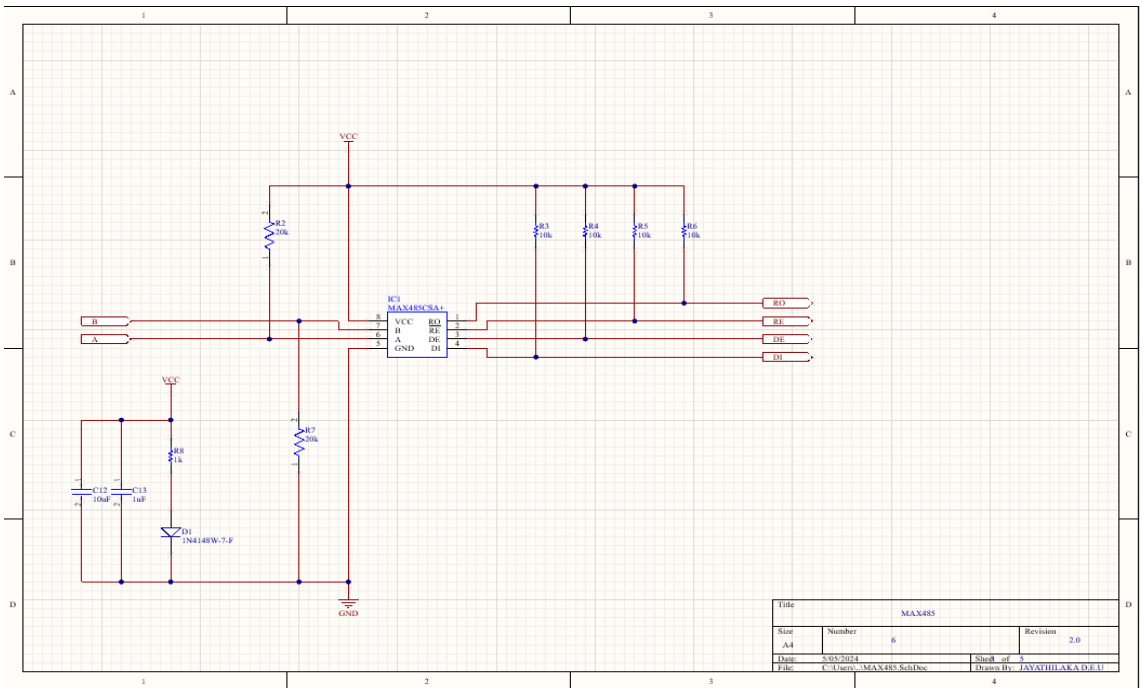


Figure 53: Datalogger - Revision 1.0



Above is the first schematic developed for the project
In the revision 2.0 only the sensor Inputs sheet was changed. We have changed the way MAX485 IC would take the inputs

Figure 54: Sensor Inputs - Revision 2.0



Keeping all other circuitry similar to rev 1.0 we have drawn the first PCB for the project

Figure 55: PCB - Top Layer

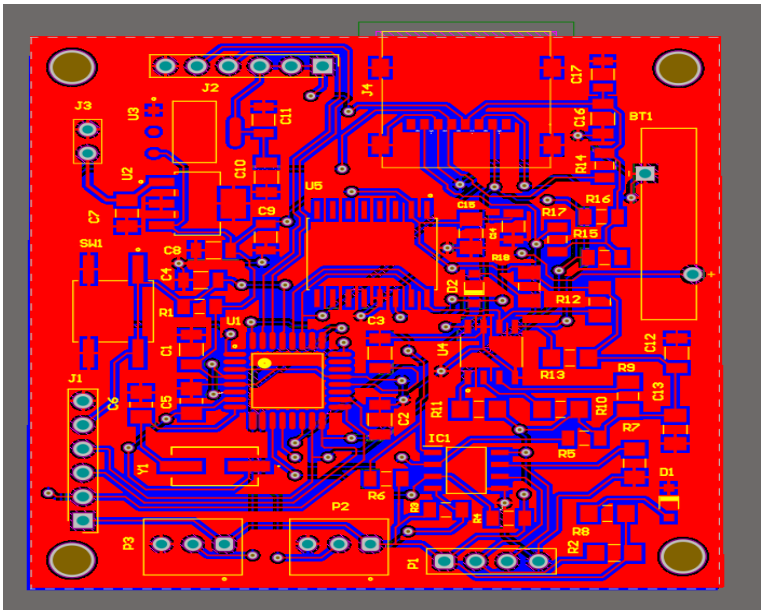


Figure 56: PCB - Bottom Layer

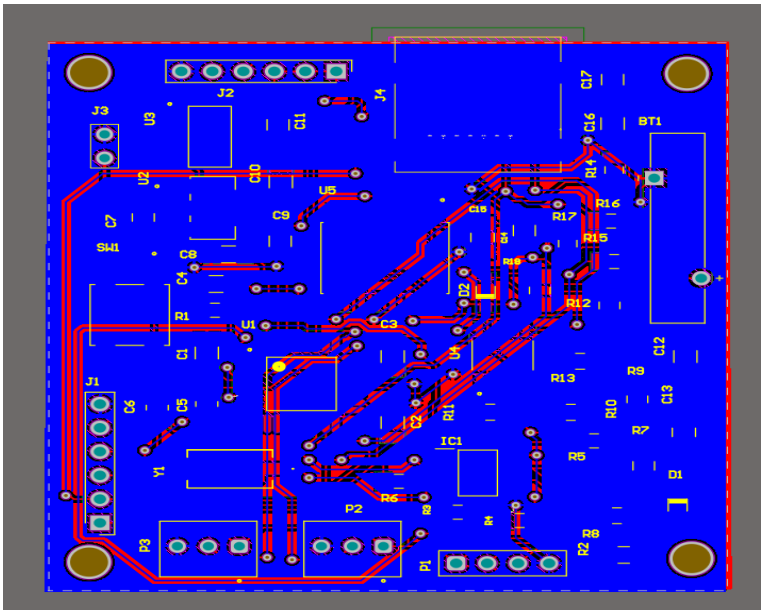
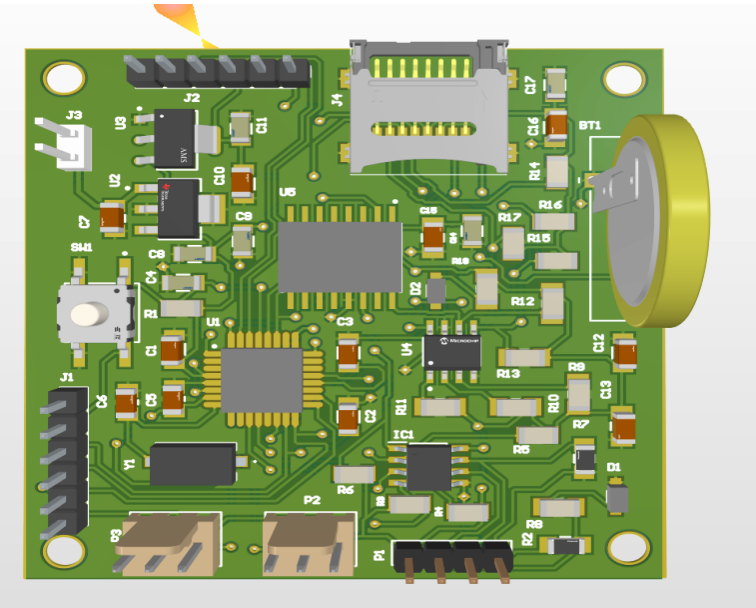


Figure 57: PCB - 3D view



This week we have set up necessary environments for the app development. This included installing VS Code, adding flutter to it's path and integrating the flutter project with the relevant firebase project.

01st April 2024 - 07th April 2024
We have improved the schematic design layout and introduced a line buffer to the datalogger to ensure reliable data transmission. Also the PCB was improved with neater routing and thicker power lines. PCB layout was also improved.

Figure 58: Main - Revision 3.0

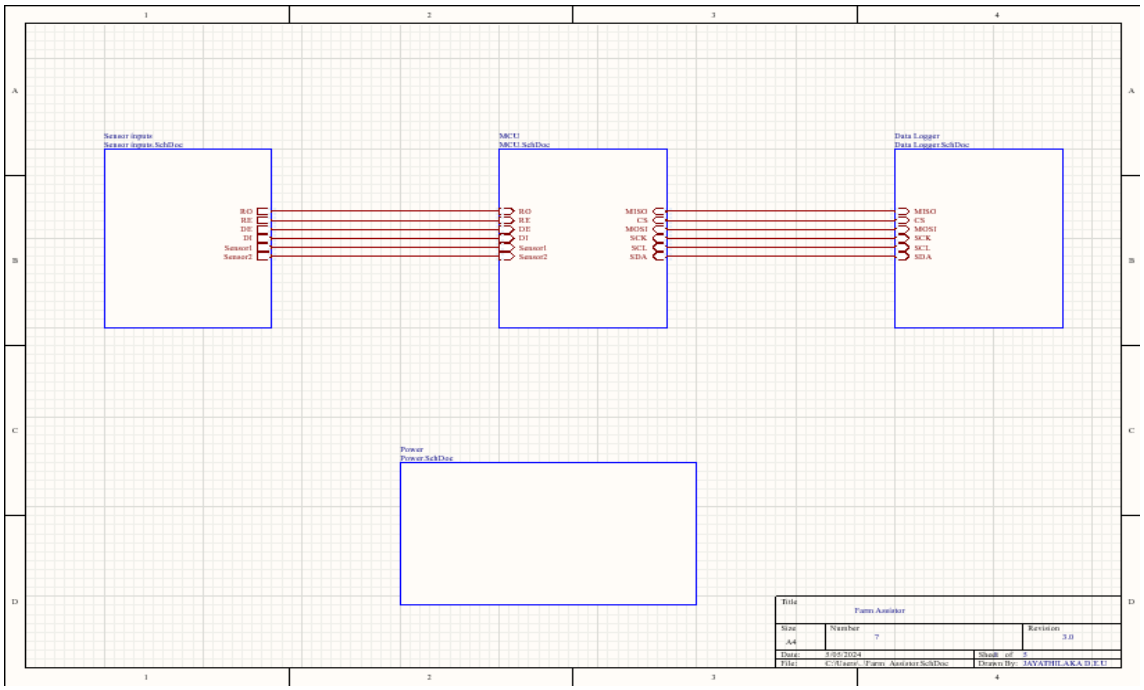


Figure 59: MCU - Revision 3.0

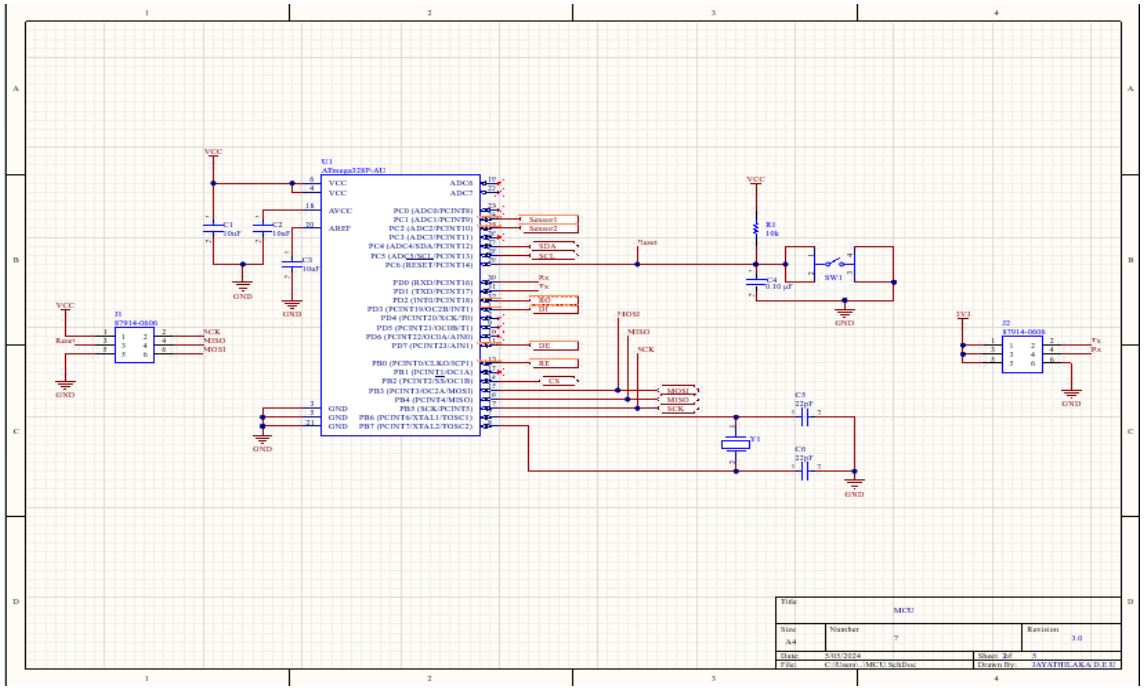


Figure 60: Power - Revision 3.0

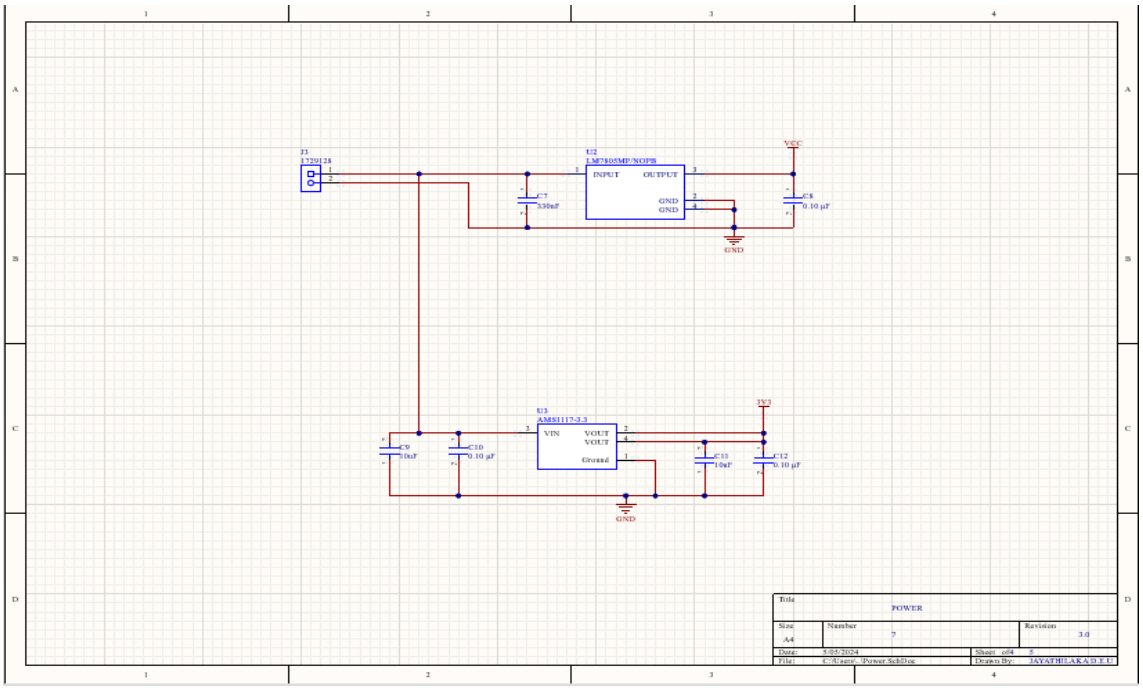


Figure 61: Sensor Inputs - Revision 3.0

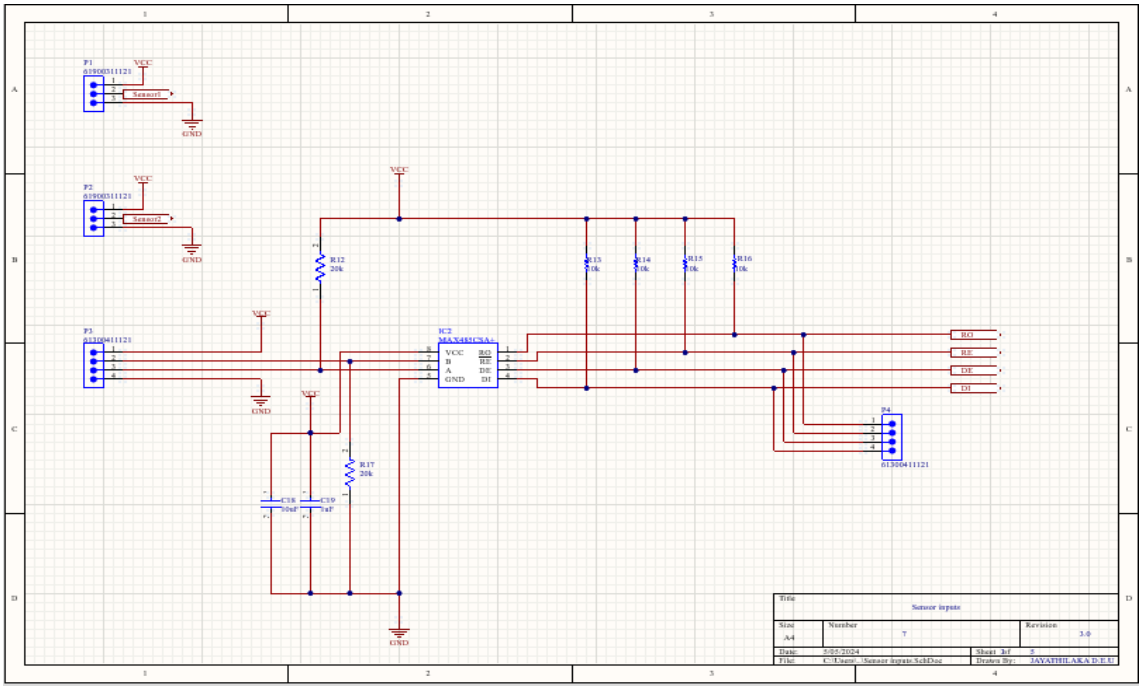


Figure 62: Datalogger - Revision 3.0

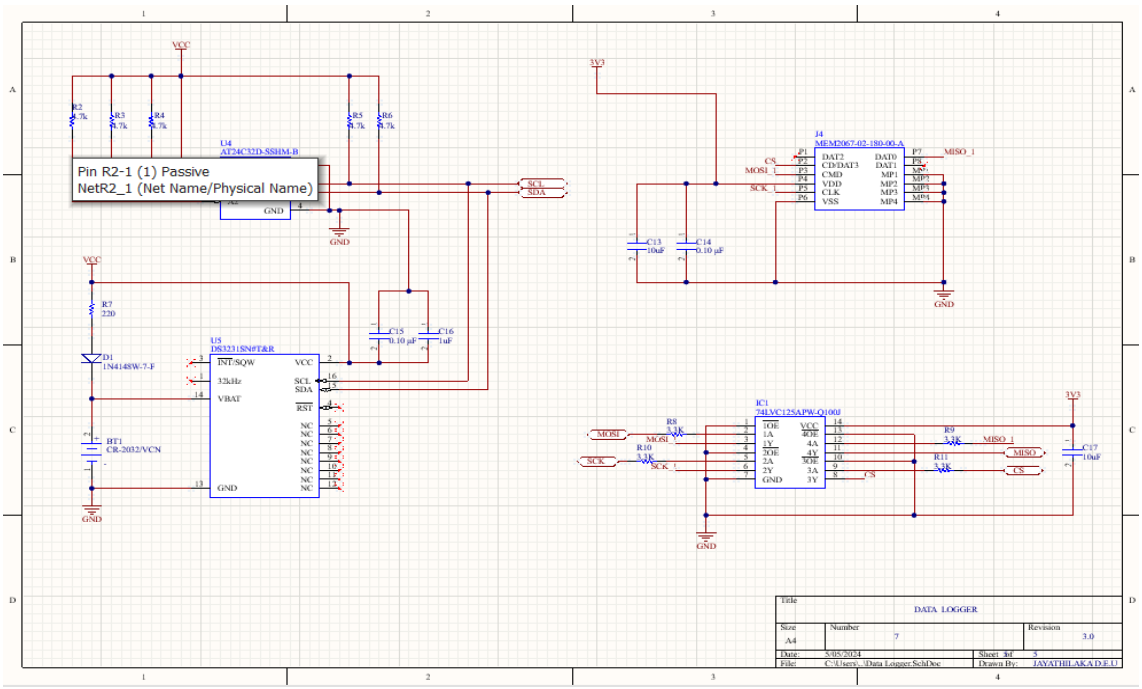


Figure 63: PCB - Top Layer

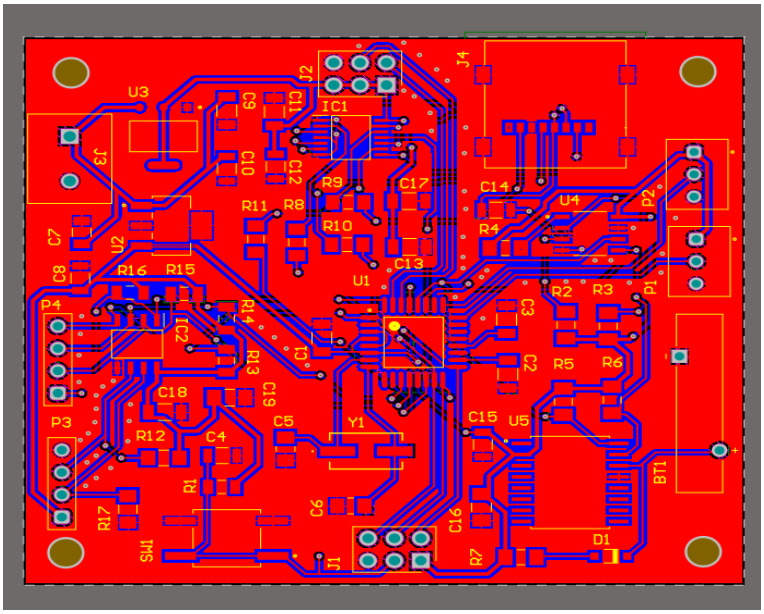


Figure 64: PCB - Bottom Layer

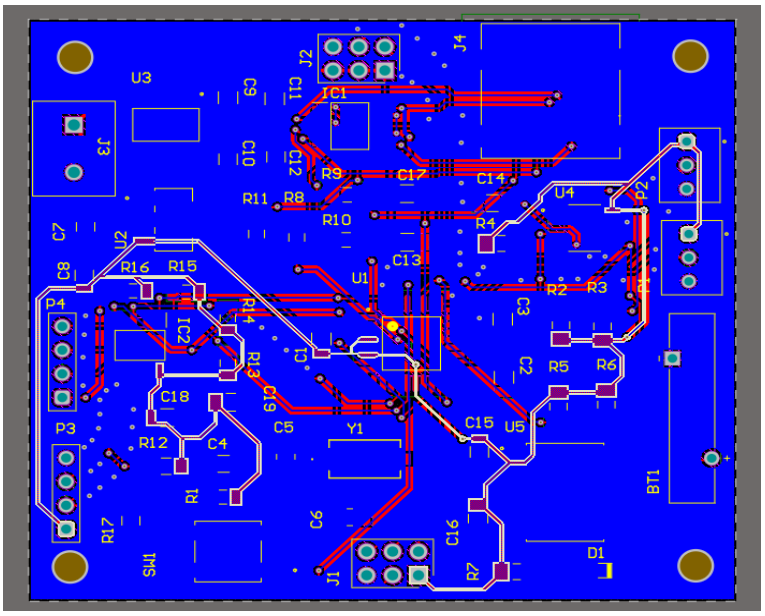
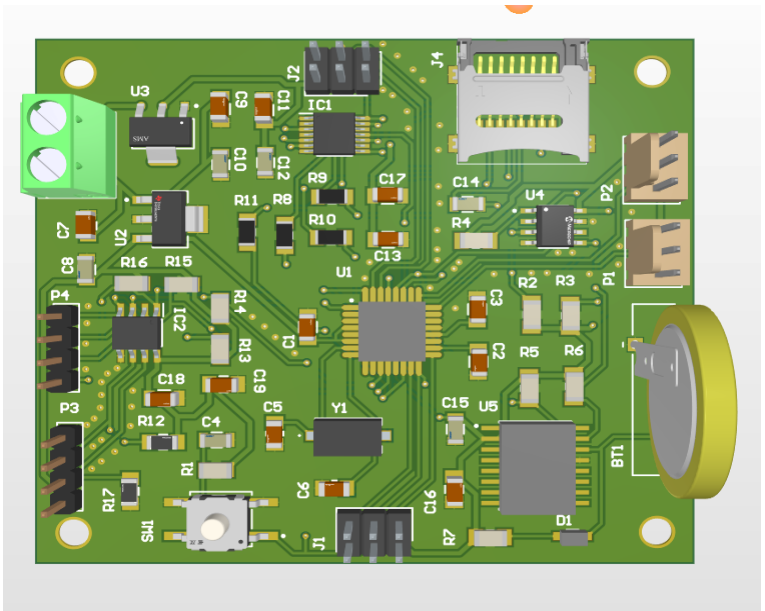


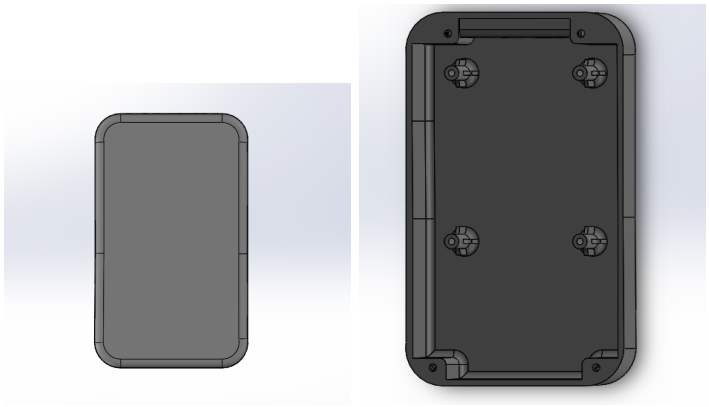
Figure 65: PCB - 3D view



08th April 2024 - 21st April 2024

In the revision 4.0(Final design), real time clock circuit was removed for cost effectivity as time stamps are not absolutely necessary and can also be taken from Wi-Fi. PCB was also improved with better techniques. It can be found under Schematic and PCB section in the main document.

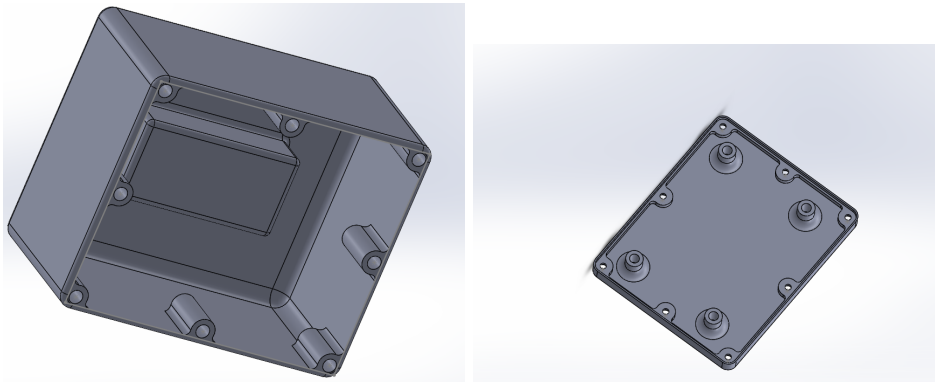
During this week, we also designed the enclosure for our product. However, the implemented design needs several modifications to align with the user-centered design model discussed in the Electronic Design Realization lecture by Prof. Jayasinghe. The main issue we encountered is that changing the dimensions of the enclosure alters the overall shape of the product. This affects both the aesthetics and functionality, necessitating careful adjustments to ensure that the final design meets user requirements without compromising on design integrity.



(a) (b)

Figure 66: enclosure designs

22st April 2024 - 28th April 2024
To enhance the aesthetics of the design, we have made significant upgrades, incorporating curved shapes in both the front and top planes. By employing hand sketches and surface modeling, we have achieved an improved design. Additionally, we changed our product shape following Prof. Jayasinghe's instructions, resulting in a new design that maintains its shape when dimensions are altered.
Moreover, draft angles have been set to ensure the designs are suitable for injection molding once the die and cavity are designed.
As a result of the lecture content, we have successfully applied the user-centered design model, particularly focusing on conceptual design. We have diligently followed the iterative design process, prioritizing users and their requirements through user need surveys.



(a) (b)

Figure 67: enclosure designs

29th April 2024 - 5th May 2024
Our PCB arrived on may 03rd. Upon arriving we did some basic board testing to see if the traces are properly made and so on. Since no board had any issues we selected one of the 5 PCBs we ordered. Components also arrived this week.

We also managed to finalise the authentication system of our app. Both on the backend and UI design.

6th April 2024 - 12th May 2024
This week we have completed the soldering of the PCB. We also completed the app development so that it can accurately exchange information with the firebase authentication and firestore real time data base.

11 Appendix

11.1 Initial Arduino Code

11.1.1 Code for ATmega328p

```
#include <DHT.h>
#include <SoftwareSerial.h>
#include <SD.h>
#include <SPI.h>
#include <ModbusMaster.h>
#include <avr/sleep.h>
#include <avr/wdt.h>

#define DHTPIN 13
#define CS_PIN 14
#define moisturesensor1 24
#define moisturesensor2 25
#define RX_PIN 30
#define TX_PIN 31
#define RE 12
#define DE 11
#define RO 32
#define DI 1

#define DHTTYPE DHT22
#define SLEEP_INTERVAL 1800000 // 30 minutes in milliseconds
#define SEND_INTERVAL 10800000 // 3 hours in milliseconds

volatile bool watchdogEvent = false;
volatile unsigned int sleepCounter = 0;

float moisture_val = NAN;
float humidity = NAN;
float air_temp = NAN;
float soil_temp = NAN;
float conductivity = NAN;
File file;
String last_line;
unsigned long lastWakeTime = 0;
unsigned long lastSendTime = 0;

DHT dht(DHTPIN, DHTTYPE);
SoftwareSerial espSerial(RX_PIN, TX_PIN);
SoftwareSerial Serial1(RO,DI);
ModbusMaster node;

void preTransmission(){
```

```
  digitalWrite(DE, HIGH);
  digitalWrite(RE, HIGH);
}
void postTransmission(){
  digitalWrite(DE, LOW);
  digitalWrite(RE, LOW);
}

void setup() {
  Serial.begin(115200);
  espSerial.begin(115200); // Initialize SoftwareSerial for communication with ESP-01
  Serial1.begin(4800); // Initialize Modbus communication baud rate

  pinMode(DE, OUTPUT);
  pinMode(RE, OUTPUT);

  // Initialize the SD card
  if (!SD.begin(CS_PIN)) {

    return;
  }
  dht.begin();

  // Initialize Modbus communication
  node.begin(1, Serial1);
  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);

  // Initialize watchdog timer
  wdt_disable();
  wdt_enable(WDTO_8S); // Set watchdog timer to 8 seconds
  WDTCSR |= (1 << WDIE); // Enable watchdog timer interrupt

}

void loop() {
  if (watchdogEvent) {
    watchdogEvent = false;
    sleepCounter++;

    // Check if it's time to take readings
    if (sleepCounter >= SLEEP_INTERVAL / 8000) {
      sleepCounter = 0; // Reset the counter

      moisture_val = readMoistureSensor();
      humidity = dht.readHumidity();
      air_temp = dht.readTemperature();
      soil_temp = readTemperature();
      conductivity = readConductivity();

      write_entry(moisture_val, soil_temp, conductivity, air_temp, humidity);
    }

    // Check if it's time to send data
    static unsigned int sendCounter = 0;
    sendCounter++;
    if (sendCounter >= SEND_INTERVAL / 8000) {
      sendCounter = 0; // Reset the counter

      readLastLine("data.txt");
      sendToESP(last_line);
    }
  }

  enterSleep();
}

void sendToESP(String data) {
```

```
  espSerial.println(data);
}

int reverseFind(File& f, char c) {
  for (unsigned long pos = f.position(); pos != 0 && f.seek(pos); --pos) {
    int rd = f.peek();
    if (rd < 0) {
      return 0;
    }

    if (rd == (c & 0xFF)) {
      return 1;
    }
  }
  return (f.peek() == (c & 0xFF));
}

int seekLastLine(File& f) {
  if (!f.seek(f.size() - 2)) {
    return 0;
  }

  if (reverseFind(f, '\n')) {
    return (f.read() != -1);
  }
  if (f.position() == 0) {
    return 1;
  }
  return 0;
}

void readLastLine(const char *filename) {
  char buf[128];

  file = SD.open("data.txt");
  if (!file) {
    return;
  }
  if (seekLastLine(file)) {
    unsigned long len = file.size() - file.position();
    if (len > 128) {
      return;
    } else if (!file.read(buf, len)) {
      return;
    } else {
      buf[len] = '\0';
      last_line = buf;
    }
  } else {
    return;
  }
  file.close();
}

void write_entry(float val1, float val2, float val3, float val4, float val5) {
  File file = SD.open("data.txt", FILE_WRITE);

  if (file) {
    file.print("Soil Moisture: ");
    file.print(String(val1));
    file.print("%, ");
    file.print("Soil Temperature: ");
    file.print(String(val2));
    file.print("'C, ");
    file.print("Soil conductivity: ");
    file.print(String(val3));
    file.print("%, ");
    file.print("Air temperature: ");
```

```
    file.print(String(val4));
    file.print("'C, ");
    file.print("Humidity: ");
    file.print(String(val5));
    file.println("%");
    file.close();
  } else {
    return
  }
}

float readMoistureSensor() {
  int val1 = analogRead(moisturesensor1);
  moisture_val1 = ((float)val1 / 1024) * 100; //this line should be properly
    calibrated with testing
  int val2 = analogRead(moisturesensor2);
  moisture_val12 = ((float)val2 / 1024) * 100;
  moisture_val = (moisture_val1+moisture_val2)/2 ;
  return moisture_val;
}

void temp_humidity() {
  delay(2000);
  humidity = dht.readHumidity();
  air_temp = dht.readTemperature();
}

float readTemperature() {
  uint8_t result;
  float soil_temp = NAN;

  const byte tempCommand[] = {0x01, 0x03, 0x00, 0x01, 0x00, 0x01, 0x84, 0x0A};
  result = node.readHoldingRegisters(0x0001, 1);
  if (result == node.ku8MBSuccess) {
    soil_temp = node.getResponseBuffer(0x00) / 10.0;
  } else {
    Serial.println("Failed to read soil temperature");
    return NAN;
  }
  return soil_temp;
}

float readConductivity() {
  uint8_t result;
  float conductivity = NAN;

  const byte conductivityCommand[] = {0x01, 0x03, 0x00, 0x03, 0x00, 0x01, 0x24, 0x0B
    };
  result = node.readHoldingRegisters(0x0003, 1);
  if (result == node.ku8MBSuccess) {
    conductivity = node.getResponseBuffer(0x00);
  } else {
    Serial.println("Failed to read soil conductivity");
    return NAN;
  }
  return conductivity;
}

void enterSleep() {
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  sleep_enable();
  sleep_mode();

  // The program continues from here after waking up
  sleep_disable(); // Disable sleep mode
}

ISR(WDT_vect) {
```

```
  // Watchdog timer interrupt service routine
  watchdogEvent = true;
}
```

11.1.2 Code for WiFi module

```
#include <ESP8266WiFi.h>
#include <FirebaseArduino.h>

// Define your Firebase and WiFi credentials
#define FIREBASE_AUTH "Fram_assistor_users"
#define FIREBASE_HOST "Farm_assistor"
#define WIFI_SSID "Farm_Assistor"
#define WIFI_PASSWORD "edr1234"

String sensor_data;

void setup() {
  // Initialize serial communication at 9600 baud rate to receive data from the
    ATmega328P
  Serial.begin(9600);

  // Connect to WiFi
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("WiFi connected");

  // Connect to Firebase
  Firebase.begin(FIREBASE_HOST, FIREBASE_AUTH);
}

void loop() {
  bool newData = false;

  // Read sensor data from the serial port
  while (Serial.available()) {
    sensor_data = Serial.readString();
    newData = true;
  }

  if (newData) {
    // Extract sensor values from the received data
    int firstCommaIndex = sensor_data.indexOf(',');
    int secondCommaIndex = sensor_data.indexOf(',', firstCommaIndex + 1);

    String moisture = sensor_data.substring(0, firstCommaIndex);
    String humidity = sensor_data.substring(firstCommaIndex + 1, secondCommaIndex);
    String temperature = sensor_data.substring(secondCommaIndex + 1);

    // Store sensor data in Firebase
    Firebase.setString("sensor/moisture", moisture);
    delay(10);
    Firebase.setString("sensor/humidity", humidity);
    delay(10);
    Firebase.setString("sensor/temperature", temperature);
    delay(10);

    // Store previous sensor data in Firebase
    Firebase.pushString("sensor/previous_moisture", moisture);
    delay(10);
```

```
    Firebase.pushString("sensor/previous_humidity", humidity);
    delay(10);
    Firebase.pushString("sensor/previous_temperature", temperature);

    // Check for Firebase errors
    if (Firebase.failed()) {
      Serial.print("Firebase set failed: ");
      Serial.println(Firebase.error());
      return;
    }

    // Print the sent data to the serial monitor for debugging
    Serial.println("Data sent to Firebase: " + sensor_data);

    newData = false; // Reset newData flag
  }

  delay(10000); // Wait for 10 seconds before the next loop
}
```

12 References

1. DHT Sensor AVR Library - GitHub
2. ATMEGA328P SD Card FAT32 SPI - GitHub
3. SDCore - GitHub
4. How to Connect ESP32 to WiFi Using ESP-IDF - Medium
5. ESP-IDF WiFi Station Example - GitHub

13 Signed Declaration by Other Group Members

This report was reviewed by,

(210216F :- H.M.G.N.I. Herath)

210669U. H.M.V. Vidumini

210037G
Amarathunga D.N.

210446J Pasira I.P.M