IOT-PHASE 3

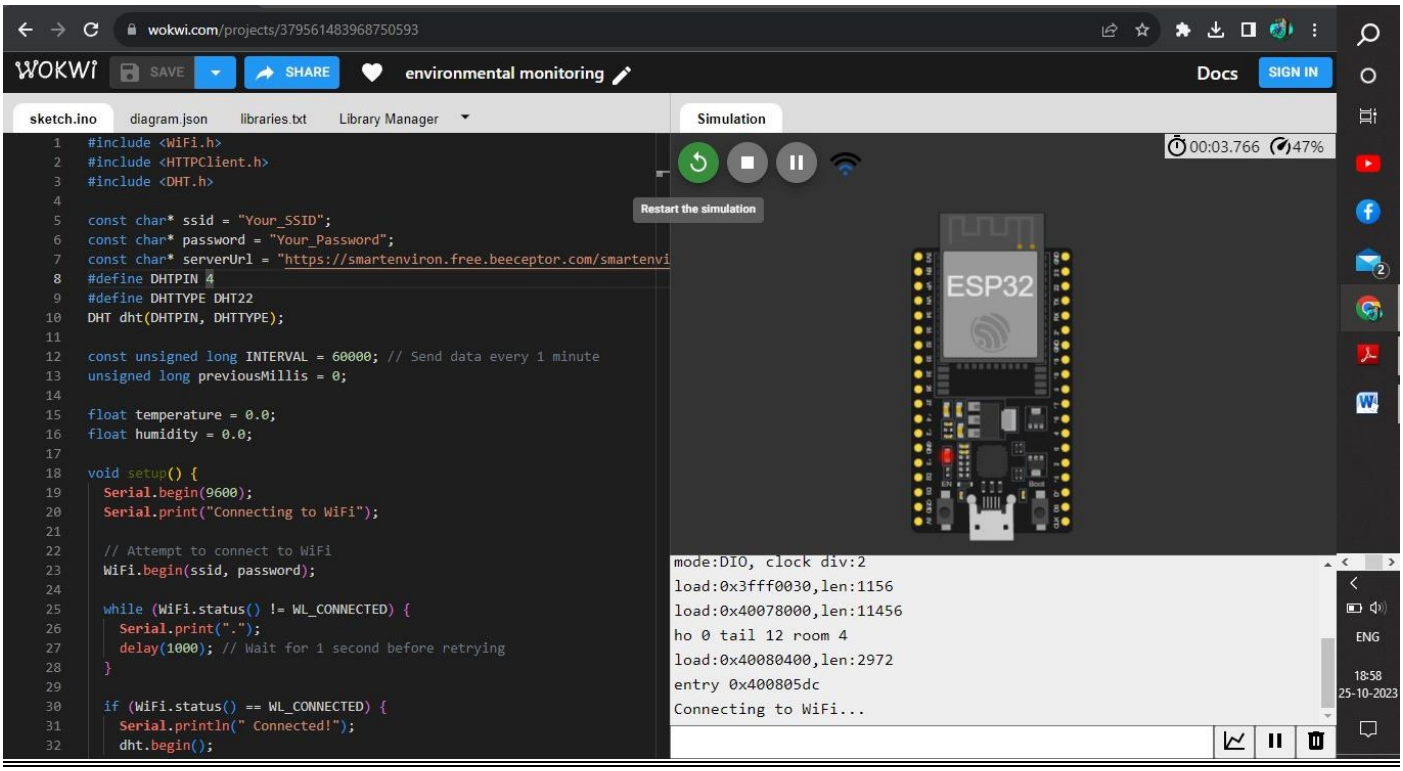
Development Part 1

CODE:

```
#include <WiFi.h>
#include <HTTPClient.h>
#include <DHT.h>
const char* ssid = "Your_SSID";
const char* password = "Your_Password";
const char* serverUrl = "https://smartenviron.free.beeceptor.com/smartenviron/";
#define DHTPIN 4
#define DHTTYPE DHT22
DHT dht(DHTPIN, DHTTYPE);
const unsigned long INTERVAL = 60000; // Send data every 1 minute
unsigned long previousMillis = 0;
float temperature = 0.0;
float humidity = 0.0;
void setup() {
  Serial.begin(9600);
  Serial.print("Connecting to WiFi");
 // Attempt to connect to WiFi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000); // Wait for 1 second before retrying
  }
if (WiFi.status() == WL_CONNECTED) {
    Serial.println(" Connected!");
    dht.begin();
  } else {
    Serial.println(" WiFi not connected.") }
```

```
}void loop() {
  if (WiFi.status() != WL_CONNECTED) {
    // WiFi is not connected, you can handle this scenario as needed
    Serial.println("WiFi not connected. Reconnecting...");
    connectToWiFi();
  } else {
    unsigned long currentMillis = millis();
    if (currentMillis - previousMillis >= INTERVAL) {
      previousMillis = currentMillis;
    float newTemperature = dht.readTemperature();
      float newHumidity = dht.readHumidity();
     if (!isnan(newTemperature) && !isnan(newHumidity)) {
        // Only send data if there's a significant change
        if (abs(newTemperature - temperature) >= 0.5 || abs(newHumidity - humidity) >= 1.0) {
          temperature = newTemperature;
          humidity = newHumidity;
          sendSensorData();
        }
      } else {
        Serial.println("Failed to read from DHT sensor!");
      } }}}
void connectToWiFi() {
  WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(1000); // Wait for 1 second before retrying
  }
if (WiFi.status() == WL_CONNECTED) {
    Serial.println("WiFi reconnected!");
  }}
void sendSensorData() {
  // The rest of your sendSensorData function remains the same}
```

OUTPUT:



CODE DISCRIPTION:

This Arduino sketch is designed to create a robust environment monitoring system. It utilizes a DHT22 sensor to capture temperature and humidity data and wirelessly transmits this data to a designated server via Wi-Fi. The core functionalities encompass Wi-Fi network connectivity, sensor data acquisition, and periodic data transmission.

Initialization and Wi-Fi Connection:

The sketch starts by initializing the Serial communication for debugging purposes with a baud rate of 9600.It then attempts to connect to a Wi-Fi network using the provided SSID (Service Set Identifier) and password.During the connection process, the sketch displays a series of dots to indicate the connection status. If a connection is established, it informs the user with a "Connected!" message. If not, it signals that Wi-Fi is not connected.

Main Loop:

The primary loop continually checks the Wi-Fi connection status. If the connection is lost, it enters a reconnection phase, calling the connectToWiFi() function.When the Wi-Fi connection is stable, the sketch monitors the time elapsed since the last data transmission. It waits for a specified interval (in this case, one minute) before checking the sensor for new temperature and humidity readings.If fresh data is available and is significantly different from the previous readings (with predefined thresholds of 0.5°C for temperature and 1.0% for humidity), it proceeds to send the updated data to the server.

Wi-Fi Reconnection Function (connectToWiFi):

This function is responsible for reconnecting to the Wi-Fi network in case of a connection loss.

It employs a loop to repeatedly attempt connection and displays progress dots.

When a successful reconnection occurs, it prints "WiFi reconnected!".

Data Transmission Function (sendSensorData):

This function constructs a URL to the designated server, incorporating the temperature and humidity data as query parameters.

It initiates an HTTP GET request to the server using the HTTPClient library.The function then evaluates the HTTP response code to confirm the success of the transmission.In the event of success, it prints the server's response.If the HTTP request fails, it reports an error code.

ADDITIONAL FEATURES:

Error Handling: Improve error handling to handle network or server issues gracefully.

Modularize Code: Break the code into smaller functions for better organization and readability.

Minimize String Usage: Avoid using the String class, as it can cause memory fragmentation on microcontrollers. Use character arrays (char[]) for data instead.

Reduce HTTP Requests: Instead of sending data in every loop iteration, consider sending data at longer intervals or only when there's a significant change in sensor values.