**Phase 4 Development part 2**

FLOOD MONITORING AND EARLY WARNING

* In the phase 3, we initiated to develop our idea of the problem statement and it ended up with the technologies to capture the real time environmental data from the water bodies using IOT devices.
* In the phase 3 mention that arduino board ,but there some problem in using arduino board . Instead of that ESP32 is used in the phase 4.
* In this phase, we are going to develop the requirements for data transmission to the cloud services and the exposure of the data to the public.
* For data transmission, we are going to use an endpoint and the data transmission is done by Wi-Fi on IoT and the transmission protocol used here is HTTP.

**Technologies Used:**

For our project, we used technologies like,

* **Data Transmission:** For data transmission, we ended up with Wi-Fi technology. We know that is a high power consuming implementation but for high coverage distances, Wi Fi is the best choice.
* **Communication Protocol:** We chose HTTP protocol for communication to the server because of its simplicity of coding and it is secure for data transmission.
* **Data Storage:**

We decided to use the endpoint for the data storage from the IoT devices that can be implemented in the parks. The name of the endpoint domain is Beeceptor. It provides the user to create mock API’s for testing. We can change the data storage idea for future implementation. Our beeceptor endpoint is <https://floodmonitoring.free.beeceptor.com>

* **Idea of Data Sharing:**

The data monitored and received from the IoT project kit(i.e Temperature and Humidity) can be exposed to the people by integrating the created API to a website that will be hosted on the network for information sharing.

**Implementation of ESP32 Controller:**

In wokwi platform, we have to develop the code for simulating our device that works on the wi-fi for the data transmission. We send the data to our beeceptor endpoint so that the code must be in the form of both using wi-fi and the http protocol. Here is the code:

#include <Wire.h>

#include <LiquidCrystal\_I2C.h>

#include <WiFi.h> // Include the WiFi library

#define TRIG\_PIN 2

#define ECHO\_PIN 4

#define RAIN\_SENSOR\_PIN 34 // ADC1\_0 on ESP32

#define BUZZER\_PIN 18

#define BUTTON\_PIN 19

const char\* ssid = "Wokwi-GUEST"; // Replace with your Wi-Fi network name

const char\* password = ""; // Replace with your Wi-Fi password

// Beeceptor Endpoint URL

const char\* beeceptorURL = "https://floodmonitoring.free.beeceptor.com"; // Replace with your Beeceptor URL

// Simulated GPS data

float latitude = 42.3601;

float longitude = -71.0589;

LiquidCrystal\_I2C lcd(0x27, 20, 4);

void setup() {

  lcd.init();

  lcd.backlight();

  lcd.clear();

  lcd.setCursor(0, 0);

  lcd.print("Flood Monitoring");

  lcd.setCursor(0, 1);

  lcd.print("and Early Warning");

**Serial**.begin(9600);

  pinMode(TRIG\_PIN, OUTPUT);

  pinMode(ECHO\_PIN, INPUT);

  pinMode(BUZZER\_PIN, OUTPUT);

  pinMode(BUTTON\_PIN, INPUT\_PULLUP);

  // Connect to Wi-Fi

  WiFi.begin(ssid, password);

  while (WiFi.status() != WL\_CONNECTED) {

    delay(1000);

**Serial**.println("Connecting to WiFi...");

  }

**Serial**.println("Connected to WiFi");

}

void loop() {

  // Simulated rain data (using potentiometer)

  int rainValue = analogRead(RAIN\_SENSOR\_PIN);

  // Read and process ultrasonic sensor data

  long duration, distance;

  digitalWrite(TRIG\_PIN, LOW);

  delayMicroseconds(2);

  digitalWrite(TRIG\_PIN, HIGH);

  delayMicroseconds(10);

  digitalWrite(TRIG\_PIN, LOW);

  duration = pulseIn(ECHO\_PIN, HIGH);

  distance = (duration / 2) / 29.1;  // Calculate distance in centimeters

  // Check if the button is pressed

  int buttonState = digitalRead(BUTTON\_PIN);

  // Check the water level based on the distance (you need to adjust these thresholds)

  int waterLevel = 0;

  if (distance < 10) {

    waterLevel = 3; // High water level

  } else if (distance < 20) {

    waterLevel = 2; // Medium water level

  } else if (distance < 30) {

    waterLevel = 1; // Low water level

  }

  // Update the LCD

  lcd.setCursor(0, 2);

  lcd.print("Water Level: ");

  lcd.print(waterLevel);

  lcd.setCursor(0, 3);

  lcd.print("Alert: ");

  if (waterLevel >= 2 || rainValue > 500) {

    lcd.print("HIGH");

    // Activate the buzzer

    digitalWrite(BUZZER\_PIN, HIGH);

    // Send the data alert to Beeceptor

    sendAlertToBeeceptor(waterLevel, rainValue);

  } else {

    lcd.print("LOW");

    // Deactivate the buzzer

    digitalWrite(BUZZER\_PIN, LOW);

  }

  // Simulated GPS data

**Serial**.print("Latitude: ");

**Serial**.print(latitude, 6);

**Serial**.print(" Longitude: ");

**Serial**.print(longitude, 6);

**Serial**.print(" Rain Value: ");

**Serial**.print(rainValue);

**Serial**.print(" Distance: ");

**Serial**.print(distance);

**Serial**.println(" cm");

  delay(6000);  // Simulate data update rate

}

void sendAlertToBeeceptor(int waterLevel, int rainValue) {

  // Create a JSON payload with the alert data

  String alertData = "{\"waterLevel\":" + String(waterLevel) + ",\"rainValue\":" + String(rainValue) + "}";

  // Create an HTTP client

  WiFiClient client;

  // Make an HTTP POST request to Beeceptor

  if (client.connect(beeceptorURL, 80)) { // Use port 80 for HTTP

    client.print("POST ");

    client.print(beeceptorURL);

    client.println(" HTTP/1.1");

    client.print("Host: ");

    client.println(beeceptorURL);

    client.println("Content-Type: application/json");

    client.print("Content-Length: ");

    client.println(alertData.length());

    client.println();

    client.println(alertData);

  }

  // Handle the response (you can add code to check the response if needed)

  while (client.available()) {

    char c = client.read();

**Serial**.print(c);

  }

  // Disconnect

  client.stop();

}

A screenshot of a computer

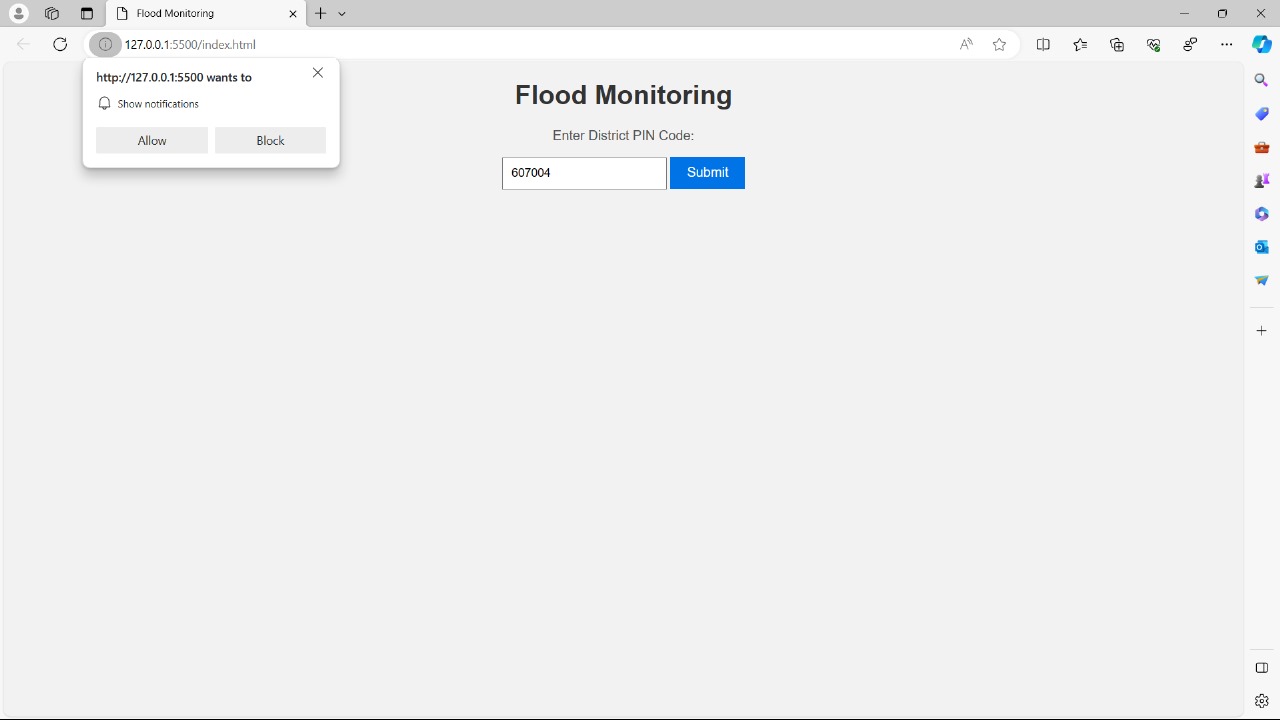
Description automatically generated

This is the simulated result image attached above. The data has to be sent to the endpoint and the from the endpoint the message “Your data is monitored” is received successfully.

**Website Implementation and creation:**

Our website is created by the programs based on coding languages like html for index, css for website design and java script for website information. The code has to be tested with programming tool. We used Visual Studio Code for Testing purposes. We have to configure the endpoint mock rules posting the data to the website we created.

The outlook of the website is attached below:



we ended up creating a website for sending notifications for the people by accessing the pincode . if the flood alert is high the notification is pushed to the user accordingly. In future we can develop for various places of alerting flood levels.

**Website Hosting:**

Our website is hosted on a free domain named Netlify since it is simple to host static websites like ours and its completely no cost for hosting.

Our website link hosted on Netlify platform is

<https://floodmonitoring.free.beeceptor.com>

All the information about the website and the mocking rules of the endpoint on beeceptor will be explained in the documentation of phase 5