Phase 5

FLOOD MONITORING AND EARLY WARNING

Final Documentation

**Problem Description:**

Flooding is a devastating natural disaster that can result from various factors such as heavy rainfall, river overflow, storm surges, or rapid snowmelt. It poses a significant threat to public safety, property, and the environment. To address this challenge, the development of a flood monitoring and early warning system is crucial.

**Solution Overview:**

The solution is decided and executed in an effective way so that each and every user will use it without any limit. Our project idea is based on the IoT devices and to send the data through the website created by using html, css and js script. This website is exposed by public by means of QR code and a social media platform.

**IoT Devices:**

The following IoT components are used for our project are described below:

* ESP32 Microcontroller.
* Ultrasonic Sensor(HC-SR04).
* Potentiometer (as Rain Sensor Alternative).
* Buzzer.
* Push Button.
* LCD Display 20x4 (I2C).

**Device Setup:**

The above components are connected to the corresponding pins as follows:

* **Ultrasonic Sensor (HC-SR04):**
* VCC to 5V on ESP32
* GND to GND on ESP32
* Trig to a GPIO pin (e.g., GPIO2) on ESP32
* Echo to a GPIO pin (e.g., GPIO4) on ESP32
* **Potentiometer (as Rain Sensor Alternative):**
* Connect one leg of the potentiometer to 3.3V on ESP32.
* Connect the other leg of the potentiometer to GND on ESP32.
* Connect the wiper (middle pin) of the potentiometer to an analog pin (e.g., ADC1\_0) on ESP32.
* **Buzzer:**
* Connect one leg of the buzzer to a GPIO pin (e.g., GPIO18) on the ESP32.
* Connect the other leg of the buzzer to GND on the ESP32.
* **Push Button:**
* Connect one terminal of the push button to a GPIO pin (e.g., GPIO19) on the ESP32.
* Connect the other terminal of the push button to GND on the ESP32.
* **LCD Display 20x4 with I2C Backpack:**
* Connect the SDA (Serial Data) pin of the I2C backpack to the SDA pin on the ESP32.
* Connect the SCL (Serial Clock) pin of the I2C backpack to the SCL pin on the ESP32.
* Connect the VCC and GND of the I2C backpack to the respective 3.3V and GND pins on the ESP32.

**Circuit Diagram:**

**A circuit board with wires and a screen

Description automatically generated**

**Diagram representation Code:**

{

  "version": 1,

  "author": "Team ",

  "editor": "wokwi",

  "parts": [

    { "type": "board-esp32-devkit-c-v4", "id": "esp", "top": 38.4, "left": -4.76, "attrs": {} },

    { "type": "wokwi-hc-sr04", "id": "ultrasonic1", "top": -84.9, "left": -32.9, "attrs": {} },

    { "type": "wokwi-potentiometer", "id": "pot1", "top": -87.7, "left": 153.4, "attrs": {} },

    {

      "type": "wokwi-buzzer",

      "id": "bz1",

      "top": -26.4,

      "left": 232.2,

      "attrs": { "volume": "0.1" }

    },

    {

      "type": "wokwi-pushbutton",

      "id": "btn1",

      "top": 60,

      "left": 173,

      "rotate": 180,

      "attrs": { "color": "green" }

    },

    {

      "type": "wokwi-lcd2004",

      "id": "lcd1",

      "top": -272,

      "left": -90.4,

      "attrs": { "pins": "i2c" }

    }

  ],

  "connections": [

    [ "esp:TX", "$serialMonitor:RX", "", [] ],

    [ "esp:RX", "$serialMonitor:TX", "", [] ],

    [ "ultrasonic1:VCC", "esp:5V", "red", [ "h-105.6", "v230.4" ] ],

    [ "ultrasonic1:GND", "esp:GND.1", "black", [ "v19.2", "h-116.4", "v163.2", "h9.6" ] ],

    [ "ultrasonic1:TRIG", "esp:2", "green", [ "v9.6", "h86", "v182.4", "h9.6" ] ],

    [ "ultrasonic1:ECHO", "esp:4", "green", [ "v19.2", "h66.4", "v153.6" ] ],

    [ "pot1:VCC", "esp:3V3", "red", [ "v57.6", "h-221.6", "v28.8" ] ],

    [ "pot1:GND", "esp:GND.1", "black", [ "v48", "h-220.8", "v9.6" ] ],

    [ "bz1:2", "esp:18", "green", [ "v86.4", "h-173.2" ] ],

    [ "bz1:1", "esp:GND.3", "green", [ "v67.2", "h-28.8" ] ],

    [ "btn1:2.r", "esp:19", "green", [ "h-19.4", "v28.6" ] ],

    [ "btn1:1.r", "btn1:2.r", "green", [ "v0" ] ],

    [ "btn1:2.l", "esp:GND.3", "green", [ "h0", "v-19.4", "h-124.8", "v57.6" ] ],

    [ "btn1:1.l", "btn1:2.l", "green", [ "h0" ] ],

    [ "pot1:SIG", "esp:VP", "green", [ "v48", "h-0.4", "v19.2", "h-220.8", "v38.4" ] ],

    [ "lcd1:VCC", "esp:3V3", "red", [ "h-67.2", "v288.1" ] ],

    [ "lcd1:GND", "esp:GND.1", "black", [ "h-48", "v432", "h38.4" ] ],

    [ "lcd1:SDA", "esp:21", "green", [ "h-19.2", "v-76.6", "h422.4", "v412.8" ] ],

    [ "lcd1:SCL", "esp:22", "green", [ "h-38.4", "v470.7", "h230.4", "v-172.8" ] ]

  ],

  "dependencies": {}

}

**Simulation code:**

Here is the simulation code for testing the circuit.

#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

**Developing the simulation for data transmission.**

The monitored data is transmitted to a endpoint on the domain called “beeceptor”. The name and the url of the endpoint is given below:

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

The procedures are described as follows:

* Create an account in beeceptor domain to create a endpoint for creatin mock API.
* Define the mocking rules for HTTP request such as POST and GET rules for storing data and used for website.

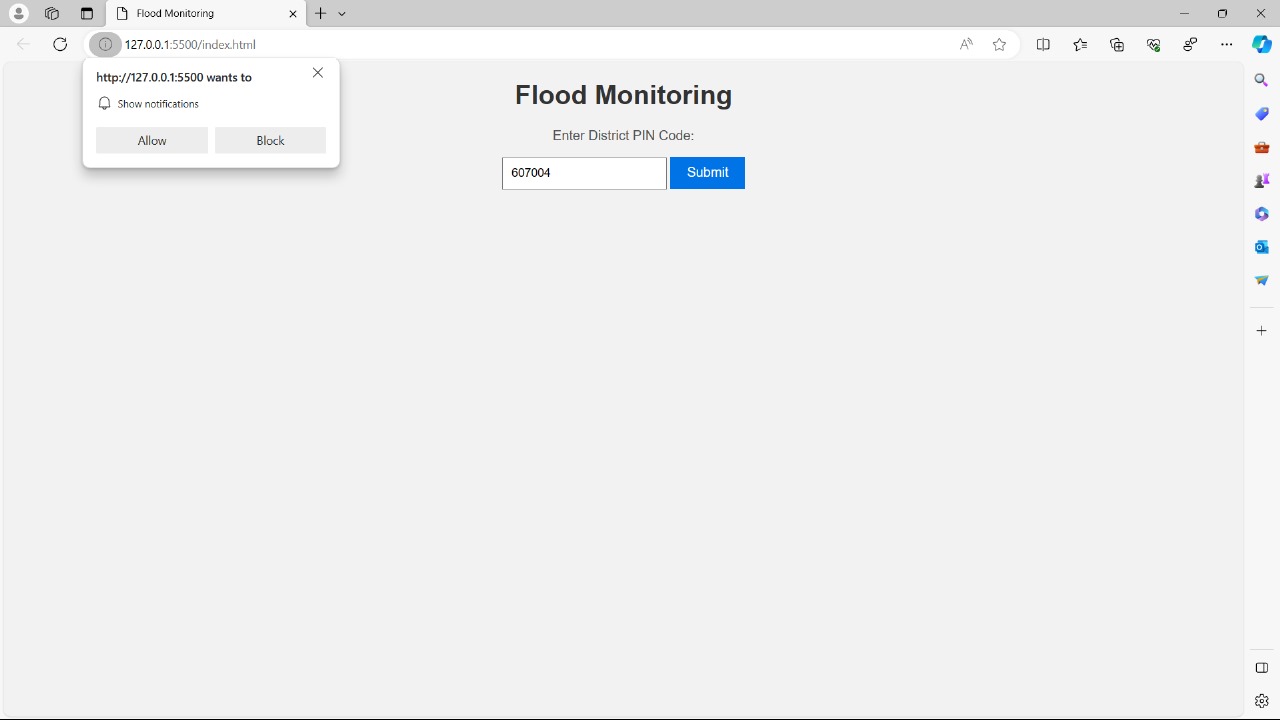
**Data Exposure to public:**

The data is shared to the public using a website. We chose to use website instead of applications because:

* Every people owns a 4G mobile but there is a question raised whether their mobile is supported for our application.
* And also many people did not like to install and use the application rather than some people.

So we ended up choosing the website that has been published via social media and QR code that can be available on parks.

The user interface of the website is given below:



The program for creating the website is as follows:

**Index.html:**

<!DOCTYPE html>

<html>

<head>

  <title>Flood Monitoring</title>

  <link rel="stylesheet" type="text/css" href="styles.css">

</head>

<body>

  <h1>Flood Monitoring</h1>

  <p>Enter District PIN Code:</p>

  <input type="text" id="pinInput" placeholder="Enter PIN Code">

  <button id="submitButton">Submit</button>

  <div id="alerts"></div>

  <script src="main.js"></script>

</body>

</html>

**Styles.css:**

body {

  font-family: Arial, sans-serif;

  text-align: center;

  background-color: #f2f2f2;

  margin: 0;

  padding: 0;

}

h1 {

  color: #333;

}

p {

  font-size: 16px;

  color: #555;

}

input[type="text"] {

  padding: 10px;

  font-size: 14px;

}

button {

  padding: 10px 20px;

  font-size: 16px;

  background-color: #0073e6;

  color: #fff;

  border: none;

  cursor: pointer;

}

#alerts {

  margin-top: 20px;

  font-size: 16px;

  color: #333;

}

**Script.js:**

document.addEventListener('DOMContentLoaded', function () {

  const pinInput = document.getElementById('pinInput');

  const submitButton = document.getElementById('submitButton');

  const alertsDiv = document.getElementById('alerts');

  submitButton.addEventListener('click', function () {

    const pinCode = pinInput.value;

    // Request notification permission when the user clicks the "Submit" button.

    requestNotificationPermission();

    // Make an HTTP request to your Beeceptor endpoint.

    fetch('https://floodmonitoring.free.beeceptor.com.')

      .then(response => response.json())

      .then(data => {

        // Replace this logic with how you want to process the received data.

        // For this example, we simulate a high alert if the received data is high.

        if (pinCode === '607004') {

          if (data && data.value > 500) {

            sendHighAlertNotification();

          }

          showAlert('No flood alerts for PIN code 12345.');

        } else {

          showAlert('Flood alert for PIN code ' + pinCode + '!');

        }

      })

      .catch(error => {

        console.error('Error:', error);

      });

  });

  function showAlert(message) {

    const alertElement = document.createElement('p');

    alertElement.textContent = message;

    alertsDiv.appendChild(alertElement);

  }

  function requestNotificationPermission() {

    if ('Notification' in window) {

      Notification.requestPermission().then(function (permission) {

        if (permission === 'granted') {

          console.log('Notification permission granted.');

        } else {

          console.log('Notification permission denied.');

        }

      });

    }

  }

  function sendHighAlertNotification() {

    if ('Notification' in window) {

      if (Notification.permission === 'granted') {

        const notification = new Notification('High Flood Alert', {

          body: 'High water level detected. Take necessary precautions.',

          icon: 'notification-icon.png' // You can provide an icon image.

        });

      }

    }

  }

});

**Procedure:**

Procedure for creating the website is described as follow:

* Choose a platform for creating website. We chose Visual studio code.
* Create a index, styles and script file for the website on the basis of html, css and javascript(js).
* Upload the necessary images for the backgound images.

**Hosting the website:**

The created website is hosted by using from various free platform like vercel, google clouds etc., We chose the Netlify platform for hosting our website because it is free of cost and its simplicity is the real deal.

To host the website, follow the instructions as follows:

* Create a account on netlify platform.
* Upload the required files for hosting the website and their required images for background.

Once created and uploaded, the website is hosted successfully. The url of our hosted website is followed by

<https://floodmonitoringibm.netlify.app/>

**Conclusion:**

The real-time Flood Monitoring and Early Warning System serves as a critical asset in enhancing public safety and emergency response coordination. Its capacity to deliver timely alerts and early warnings empowers individuals and communities to take proactive measures, reducing the risk of casualties, property damage, and economic losses. The system's prompt notification of authorities optimizes resource allocation, enhancing the efficiency of emergency services. Furthermore, it fosters a culture of preparedness, building community resilience and reducing panic during actual flooding incidents. Notably, this system not only minimizes human suffering and property damage but also generates valuable data for long-term flood mitigation and urban planning, ultimately contributing to a more resilient and prepared society in flood-prone areas.