**FLOOD MONITORING AND EARLY WARNING SYSTEM .**

**PHASE 1:**

**PROBLEM:**

**The project focuses on implementing iot sensors to develop a system for flood monitoring and early warning.This system should be able to acurately detect** **and monitor flood conditions in real-time,and provide timely warning to affected areas to minimize the potential damage and loss of life.**

A flood monitoring and early warning IoT (Internet of Things) project can be a valuable solution to mitigate the risks associated with flooding. Such a project involves the use of various sensors, data collection and analysis, communication technologies, and a user interface to provide timely warnings and information to residents and authorities. Here is a high-level overview of the key components and steps involved in creating such a project:

**Sensor Deployment:**

* Deploy various types of sensors in flood-prone areas. These sensors can include:
* Water level sensors to measure water height in rivers, streams, or floodplains.
* Rainfall sensors to measure precipitation.
* Weather sensors to monitor atmospheric conditions (temperature, humidity, pressure, etc.).
* Soil moisture sensors to gauge ground saturation.
* Cameras or image sensors for visual monitoring.
* These sensors should be strategically placed in areas susceptible to flooding.

**Data Collection**:

* Collect data from the deployed sensors in real-time.
* Store the data in a centralized database or cloud storage for analysis.

**Data Analysis**:

* Implement algorithms to analyze the collected data.
* Detect patterns and trends in water levels, rainfall, and other relevant data.
* Set threshold values to trigger alerts and warnings.

**Alert Generation**:

* When the analysis indicates potential flooding, generate automated alerts and warnings.
* These alerts can be sent via various communication channels, such as SMS, email, mobile apps, and sirens.
* Different warning levels can be issued based on the severity of the flood risk.

**Interface**: **User**

* Develop a user-friendly interface for residents and authorities to access real-time data and warnings.
* Include maps, charts, and visualizations to help users understand the flood situation.
* Allow users to sign up for alerts and notifications.

**Communication Infrastructure**:

* Ensure reliable communication between sensors, data analysis systems, and the user interface.
* Use a combination of wired and wireless technologies, such as Wi-Fi, cellular, and satellite communication.

**Power Management**:

* Implement efficient power management solutions for sensors, such as solar panels and battery backup systems.

**Testing and Calibration**:

* Regularly test and calibrate sensors to ensure accurate data collection and analysis.
* Conduct simulation tests to evaluate the system's response to various flood scenarios.

**Integration with Emergency Services**:

* Establish protocols for sharing flood information with local emergency services and authorities.
* Ensure that the system can automatically notify relevant agencies when severe flooding is detected.

**Education and Public Awareness**:

* Educate the local community about the flood monitoring system and how to respond to warnings.
* Conduct outreach programs to raise awareness about flood safety.

**Maintenance and Updates**:

* Perform routine maintenance on sensors and communication equipment.
* Keep software and algorithms up-to-date to improve accuracy and effectiveness.

**Data Storage and Analysis for Long-term Planning**:

* Retain historical data for long-term flood analysis and planning.
* Use historical data to identify flood-prone trends and improve flood management strategies.

A flood monitoring and early warning IoT project can significantly reduce the impact of floods on communities by providing timely information and enabling proactive responses. It's crucial to collaborate with local authorities, experts, and the community to ensure the success of such a project. Additionally, considering the scalability and sustainability of the system is essential for long-term .

**PHASE 2 : INNOVATION**

After through research and analysis,we arrived at an innovative solution to solve the above problem as detailed in phase 1 of our project.

* We will be using the ESP8266 and GSM 900A module as both these suit the best of our project .

**SENSOR:**

Here we using ultrasonic sensor and float sensor.

**Ultrasonic sensor:**

These sensors can provide continuous monitoring of water levels,which is crucial for flood detection and early warning.they can detect rising water levels and trigger alerts when the water level exceeds a certain threshold

**Float sensors:**

Float sensors can provide early warning by triggering alerts when the water level exceeds a predetermined threshold.these alerts can be used to initiate evacuation procedures and take preventive measures.

**GSM900A module:**

The GSM 900A module allows for wireless communication over the GSM network.this enables the flood monitoring system to send data,alerts,and status updates to a central monitoring station or designated authorities in real-time.

**STORAGE:**

For storage ,we using google cloud as it ensures scalability,data storage and analytics

**NETWORK:**

GSM 900A module can be usedas a redundancy mechanism.if one communication method fails,the GSM module can still transmit data via the mobile network,ensuring continuous operation of the flood monitoring system.

**PHASE 3 :**

**Problem**

To develop the python script on IoT devices as per the project requirement.

**Solution :**

In flood monitoring and early warning we made a easy approach in simulating by instead of using all the components ,here we only using

Three main components like ESP32 ,ultrasonic sensor and servo motor .

**Source Code**

**Include libraries:**

Wi-Fi.h -this library is to connect the ESP32 to Wi-Fi network

Wi-FiClient.h0 - this library allows the ESP32 to create client that can connect to IP address and port

BlynksimpleESP32.h-this library provides a blynk-related functions for ESP32Servo.h-this library enables you to control servo motors

**Initialize objects and variables:**

Defines the pins for the ultrasonic sensor and servo motor

**Setup fuction:**

Initialize the ultrasonic sensor with its D2 and D4 as output pin.it sends the command using blynk.

**Loops function:**

Read the distance from the ultrasonic sensor using the sonar.ping-cm()function.

Check if the flood level exceeds the defined threshold by calculating distance from ultrasonic sensor.if the conditions are match ,send alerts function by blynk.

#define BLYNK\_TEMPLATE\_ID "TMPL3tobBFjj-"

#define BLYNK\_TEMPLATE\_NAME "IOT FLOOD MONITORING"

#define BLYNK\_AUTH\_TOKEN "gy2bzR-i-RbPW3oWOpAiDgr6sSVzIHVZ"

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "Wokwi-GUEST";

char pass[] = "";

#define BLYNK\_PRINT **Serial**

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <ESP32Servo.h>

Servo gate;

const int trigPin=2;//d2

const int echoPin=4;//d4

const int servoPin = 18;//d18

long duration;

int distance;

void setup() {

**Serial**.begin(9600);

Blynk.begin(auth, ssid, pass);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

gate.attach(servoPin, 500, 2400);

}

void loop()

{

digitalWrite(trigPin, LOW);

delay(2);

digitalWrite(trigPin,HIGH);

delay(10);

digitalWrite(trigPin, LOW);

duration=pulseIn(echoPin,HIGH);

distance=duration\*0.034/2;

**Serial**.println(distance);

Blynk.virtualWrite(V0,distance);

if(distance<50)

{

gate.write(90);

Blynk.virtualWrite(V1,"FLOOD DETECTED GATES OPENED");

}

else

{

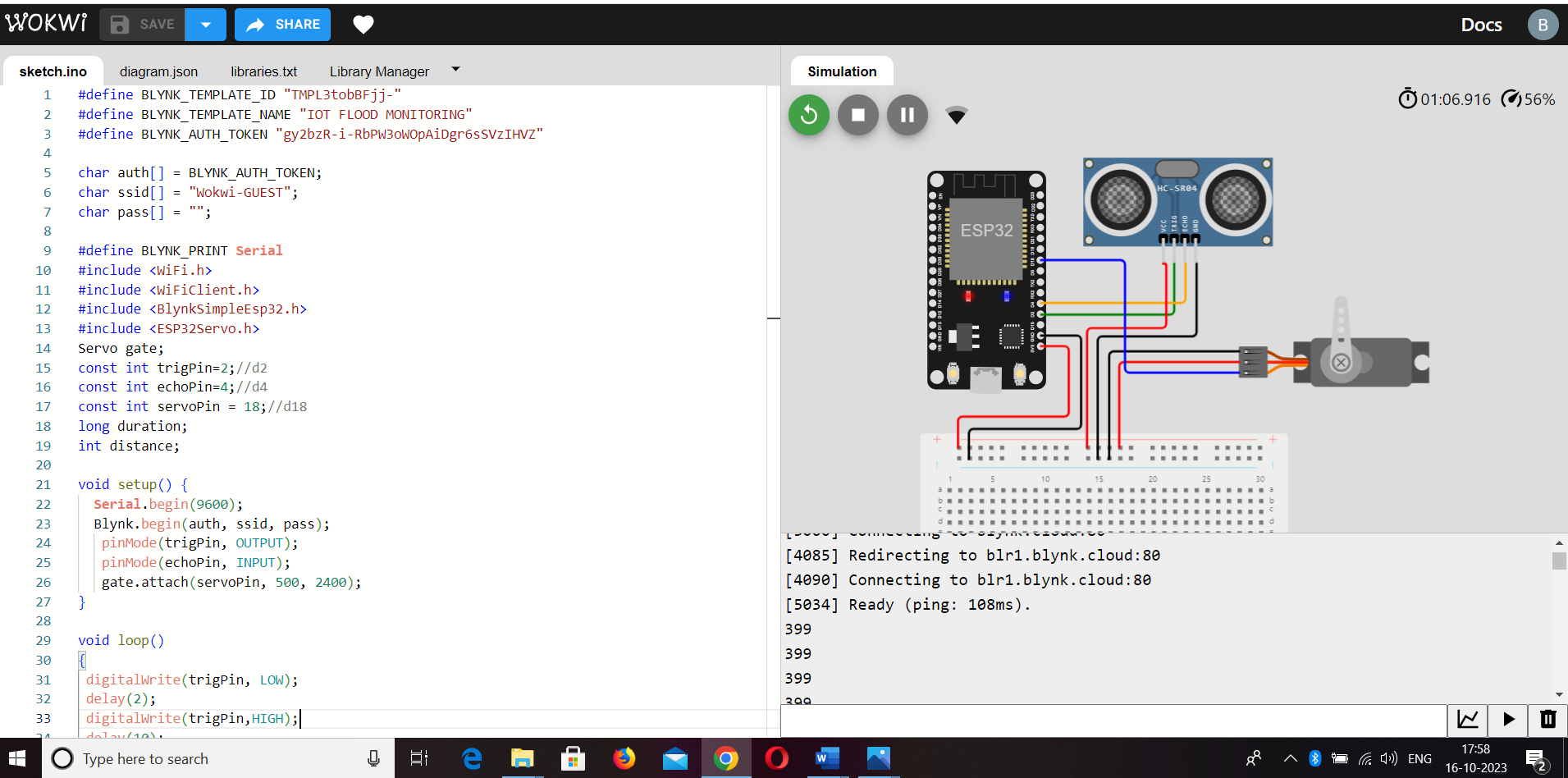
gate.write(0);

Blynk.virtualWrite(V1,"SAFE CONDITIONS GATES CLOSED");

}

}

**Simulation output**



**Simulation link:**

[**https://wokwi.com/projects/376186118233828353**](https://wokwi.com/projects/376186118233828353)

**PHASE 4:**

PROBLEM:

As we submitted arduino code in previous phase ,so we continue the same code instead of python and in this phase we just adding blynk and beeceptor in the code and simulating it.

Code :

#include <HTTPClient.h>

#include <WiFi.h>

#include <WiFiClient.h>

#include <BlynkSimpleEsp32.h>

#include <ESP32Servo.h>

#define BLYNK\_TEMPLATE\_ID "TMPL3i2OK5rVO"

#define BLYNK\_TEMPLATE\_NAME "FLOOD MONITORING AND EARLY WARNING"

#define BLYNK\_AUTH\_TOKEN "AO2XlsXPyLLBLpgRzGFwM8dyJWRt0AVq"

char auth[] = BLYNK\_AUTH\_TOKEN;

char ssid[] = "YourWiFiSSID";

char pass[] = "YourWiFiPassword";

#define BLYNK\_PRINT **Serial**

Servo gate;

const int trigPin = 2; // d2

const int echoPin = 4; // d4

const int servoPin = 18; // d18

long duration;

int distance;

void setup() {

**Serial**.begin(9600);

Blynk.begin(auth, ssid, pass);

pinMode(trigPin, OUTPUT);

pinMode(echoPin, INPUT);

gate.attach(servoPin, 500, 2400);

}

void loop() {

digitalWrite(trigPin, LOW);

delay(2);

digitalWrite(trigPin, HIGH);

delay(10);

digitalWrite(trigPin, LOW);

duration = pulseIn(echoPin, HIGH);

distance = duration \* 0.034 / 2;

**Serial**.println(distance);

Blynk.virtualWrite(V0, distance);

if (distance < 50) {

gate.write(90);

Blynk.virtualWrite(V1, "FLOOD DETECTED GATES OPENED");

// Send flood alert to Beeceptor

sendFloodAlert("FLOOD DETECTED GATES OPENED");

}

else {

gate.write(0);

Blynk.virtualWrite(V1, "SAFE CONDITIONS GATES CLOSED");

// Send safe condition to Beeceptor

sendFloodAlert("SAFE CONDITIONS GATES CLOSED");

}

}

void sendFloodAlert(const char\* alertMessage) {

HTTPClient http;

String url = "<https://phase.free.beeceptor.com>"; // Replace with your Beeceptor URL

http.begin(url);

http.addHeader("Content-Type", "application/json");

String jsonPayload = "{\"message\": \"" + String(alertMessage) + "\"}";

int httpCode = http.POST(jsonPayload);

if (httpCode > 0) {

**Serial**.printf("HTTP POST response code: %d\n", httpCode);

} else {

**Serial**.println("HTTP POST request failed");

}

http.end();

}

Here's a step-by-step explanation of the code:

**Include necessary libraries**:

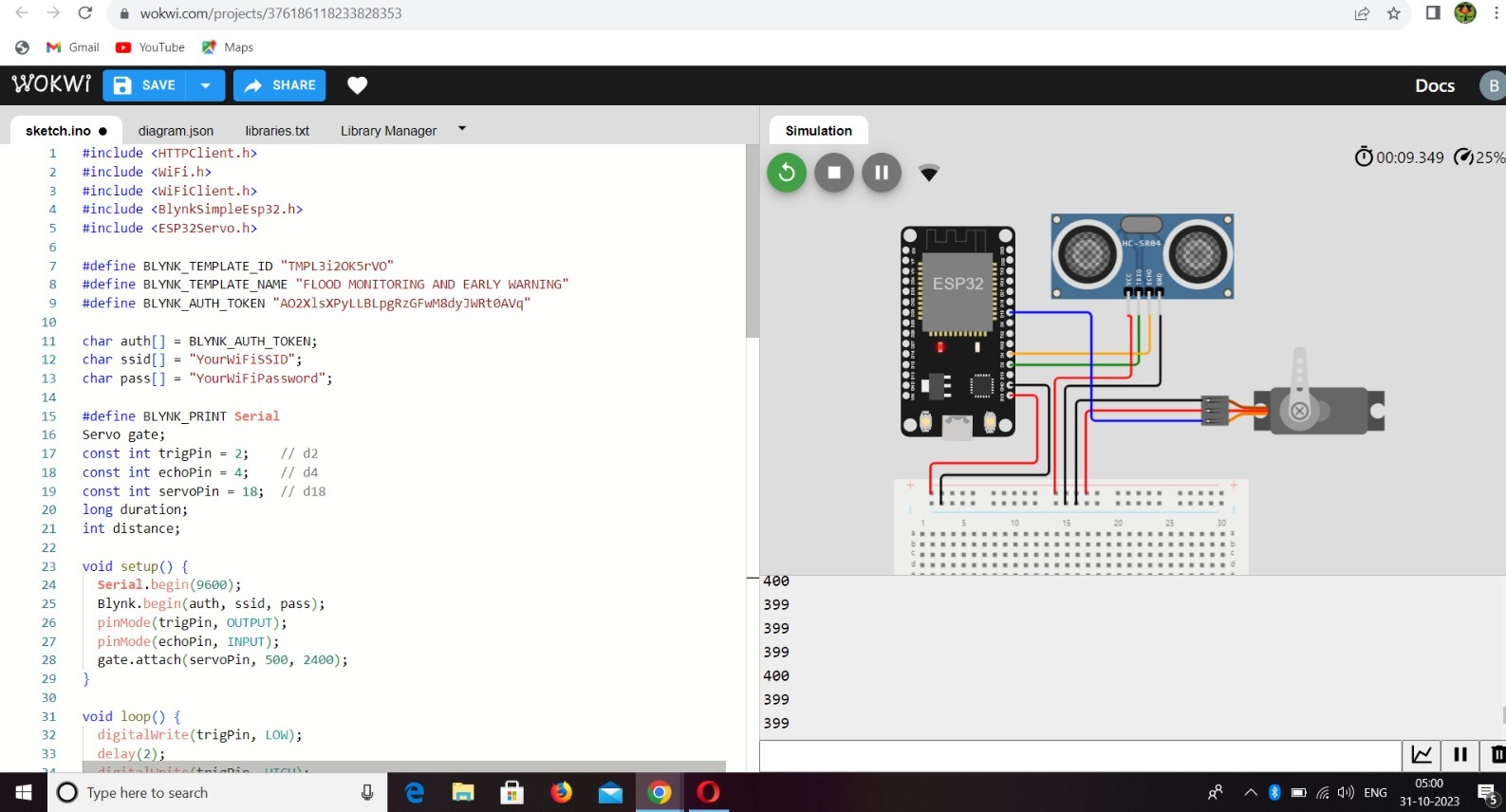
* **HTTPClient.h**: This library is used for making HTTP requests.
* **WiFi.h** and **WiFiClient.h**: These libraries are used for connecting to a Wi-Fi network.
* **BlynkSimpleEsp32.h**: This library provides support for the Blynk IoT platform, which allows you to create a mobile app to control and monitor your ESP32 device remotely.
* **ESP32Servo.h**: This library is used for controlling a servo motor on the ESP32.
* Define some constants and variables:
* **BLYNK\_TEMPLATE\_ID**, **BLYNK\_TEMPLATE\_NAME**, and **BLYNK\_AUTH\_TOKEN** are related to Blynk. They define the template ID, template name, and authentication token for connecting to the Blynk service.
* **auth**, **ssid**, and **pass** are variables for the Blynk authentication token, Wi-Fi SSID, and Wi-Fi password.
* **Servo gate** is an instance of the Servo class for controlling a servo motor.
* **trigPin**, **echoPin**, and **servoPin** define the GPIO pins used for the ultrasonic sensor's trigger, echo, and the servo motor, respectively.
* Setup function (**void setup()**):
* Initializes serial communication for debugging.
* Calls **Blynk.begin()** to connect to the Blynk service using the provided authentication token and Wi-Fi credentials.
* Configures the trigger and echo pins for the ultrasonic sensor as output and input, respectively.
* Attaches the servo motor to the designated pin (servoPin).
* Loop function (**void loop()**):
* Starts by sending a trigger signal to the ultrasonic sensor and measuring the time it takes for the echo to bounce back. This is used to calculate the distance to an object in front of the sensor.
* The distance is then printed to the serial monitor and sent to Blynk for display on a mobile app (Virtual Pin V0).
* If the distance is less than 50 centimeters, it means a flood condition is detected. In response, the code:
* Rotates the servo motor to an angle of 90 degrees, which presumably opens a gate (Virtual Pin V1 is updated with a message).
* Calls the **sendFloodAlert** function to send an alert to the Beeceptor service with the message "FLOOD DETECTED GATES OPENED."
* If the distance is greater than or equal to 50 centimeters, indicating safe conditions, the code:
* Rotates the servo motor to an angle of 0 degrees (presumably closing the gate).
* Updates Virtual Pin V1 with the message "SAFE CONDITIONS GATES CLOSED."
* Calls **sendFloodAlert** to send a "SAFE CONDITIONS GATES CLOSED" alert to Beeceptor.
* **sendFloodAlert** function:
* This function is responsible for sending HTTP POST requests to a Beeceptor URL with a JSON payload containing the alert message.
* It uses the **HTTPClient** library to make the POST request and includes the alert message in the payload.
* The HTTP response code is printed to the serial monitor for debugging

Overall, this code is a simple example of an ESP32-based flood monitoring and control system that uses an ultrasonic sensor to detect water levels and a servo motor to control a gate. It also integrates with the **Blynk** platform for remote monitoring and control and sends alerts to **Beeceptor** when flood conditions are detected. Note that you need to replace the placeholders with your actual Blynk and Beeceptor information for this code to work properly.

**STEPS:**

* Create a Beeceptor Endpoint
* Configure the Endpoint
* Update th code
* HTTP data transmission
* Data analysis and testing

**Simulation output after adding beeceptor and blynk:**



**BY,**

|  |  |
| --- | --- |
| **NAME** | **REGISTER NUMBER** |
| BOOMIKA A | 2021504311 |
| DIVYA PRIYA E | 2021504314 |
| VANATHI K | 2021504308 |
| KAVIN S | 2021504316 |
| VIKRAM A | 2021504317 |