**PHASE 5 : FLOOD MONITORING AND EARLY WARNING**

**DOCUMENTATION:**

**OBJECTIVE :**

The **objective of flood monitoring and early warning systems is to detect, assess, and provide timely notifications and alerts about potential or ongoing flooding events. These systems aim to mitigate the impact of floods by offering the following key benefits:**

* Public Safety**: The primary goal of flood monitoring and early warning systems is to protect human lives and reduce the risk to public safety. By providing advance notice of impending floods, people can take precautionary measures and evacuate if necessary.**
* **Property Protection: Flood warnings also help in safeguarding property and assets. People can move their belongings to higher ground, install flood defenses, or take other protective actions.**
* **Infrastructure Resilience: Early warnings allow for the protection and preservation of critical infrastructure, such as roads, bridges, dams, and utility facilities. Preventing or minimizing damage to these structures helps maintain essential services.**
* **Agricultural and Environmental Management: In rural areas, flood warnings can help farmers and environmental agencies manage water resources more effectively. They can reduce crop damage and limit harm to ecosystems.**
* **Economic Impact Reduction: Flooding can have severe economic consequences. Timely warnings enable businesses to prepare and minimize financial losses due to flood-related disruptions.**
* **Community Resilience: Flood warning systems contribute to building community resilience by raising awareness about flood risks and encouraging preparedness and response measures.**
* **Data Collection and Research: Flood monitoring systems generate valuable data for scientific research and flood modeling, leading to better flood risk assessments and improved preparedness and response strategies.**
* **Improved Emergency Response: Early warnings are crucial for emergency responders, allowing them to prepare for and respond to flooding events more effectively.**
* **Public Awareness: These systems help raise awareness about the risks of flooding, thereby encouraging individuals and communities to be better prepared for natural disasters.**
* **Government and Policy Support: Flood monitoring and early warning systems are essential for governments and policymakers to make informed decisions about land use, zoning regulations, and infrastructure development in flood-prone areas.**
* **International Cooperation: In regions susceptible to transboundary flooding, such as river basins that cross multiple countries, early warning systems foster cooperation and information sharing among nations to mitigate cross-border flood risks.**
* **Climate Change Adaptation: As climate change leads to more extreme weather events, including heavy rainfall and increased flood risk, these systems become even more critical for adapting to a changing climate.**

Iot device setup:

**Setup Steps:**

**Device Configuration:**

* Set up your IoT device with the necessary sensors and connectivity options.
* Write firmware or code to collect sensor data and transmit it to the chosen cloud platform.

**Cloud Platform Configuration:**

* Create an account on your chosen cloud platform.
* Register your IoT device and configure it to accept data from your device.
* Set up data storage and establish rules for data processing.

**Alerting and Notifications:**

* Implement alerting rules that trigger notifications when flood conditions are detected. This can be done through the cloud platform or via third-party services like Twilio for SMS alerts.

**User Interface:**

* Build a user interface to visualize data and control the system. This can be done using the platform's built-in tools or by developing a custom application.

**Testing and Calibration:**

* Test your system extensively to ensure that it accurately detects and reports flood conditions.
* Calibrate your sensors if needed to provide accurate measurements.

**Deployment:**

* Install the IoT device in the flood-prone area, ensuring that it is securely placed and has access to power and an internet connection.

**Maintenance and Monitoring:**

* Regularly monitor the system for performance and data quality.
* Address any issues promptly and conduct maintenance as required.

**Public Awareness:**

* Educate the local community about the system, its purpose, and what actions they should take when they receive flood alerts.

**COMPONENTS USED :**

**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.

PLATFORM DEVELOPMENT:

**1. Define Requirements:**

* Identify the specific needs and objectives of your flood monitoring and early warning platform. Consider factors like the geographical area, flood types, expected user base, and available resources.

**2. Hardware and Sensor Selection:**

* Choose appropriate sensors to collect data, such as water level sensors, rain gauges, weather sensors, and communication modules.
* Select microcontrollers or IoT devices capable of connecting to the internet.

**3. Data Collection:**

* Develop firmware or software for the IoT devices to collect data from sensors.
* Implement a data transmission mechanism to send data to a central server or cloud platform.

**4. Cloud Platform Development:**

* Set up a cloud-based platform to manage and process the incoming data. Options include AWS IoT, Azure IoT, Google Cloud IoT, or other IoT platforms.
* Create data storage solutions (databases) to store historical data.
* Develop APIs to receive and process data from IoT devices.
* Implement data analytics to monitor trends, anomalies, and trigger alerts.

**5. Alerting and Notification System:**

* Develop alerting rules and mechanisms based on data analysis.
* Implement notification methods such as SMS, email, mobile app notifications, or automated phone calls.
* Ensure that alerts are timely and reliable.

**6. User Interface Development:**

* Create web-based dashboards or mobile apps for users to monitor flood data and receive alerts.
* Include interactive features for users to configure alert preferences and view historical data.

**7. Geographic Information System (GIS) Integration:**

* Incorporate GIS data for mapping and visualizing flood-prone areas, historical flood data, and evacuation routes.
* Utilize GIS for geospatial analysis to enhance flood predictions and alerts.

**8. Machine Learning and Predictive Analytics:**

* Implement machine learning models to improve the accuracy of flood predictions and early warnings.
* Train models on historical data and continuously update them with real-time data.

**9. Regulatory Compliance and Data Security:**

* Ensure compliance with data privacy and security regulations.
* Protect sensitive data and user information.

**10. Mobile Apps and Web Portals:**

* Develop user-friendly mobile applications and web portals.
* Include features like real-time monitoring, historical data access, alert settings, and user management.

**11. Integration with External Services:**

* Integrate with weather services, government agencies, and other external data sources to enhance the accuracy of flood predictions.

**12. Testing and Validation:**

* Thoroughly test the platform to ensure that it accurately collects and analyzes data, triggers alerts, and provides a reliable user experience.

**13. User Training and Awareness:**

* Train local authorities and communities on how to use the platform effectively.
* Promote public awareness about the platform's capabilities and the importance of flood preparedness.

**14. Deployment:**

* Deploy IoT devices and set up monitoring stations in flood-prone areas.
* Ensure reliable power sources and connectivity for the devices.

**15. Maintenance and Updates:**

* Regularly maintain and update the platform, including software, sensors, and hardware, to ensure its continued reliability and effectiveness.

**16. Community Engagement:**

* Engage with the local community to gather feedback, address concerns, and improve the platform based on user input.

**17. Scaling and Expansion:**

* Plan for the potential expansion of the platform to cover larger areas or serve more users.

**18. Data Backup and Redundancy:**

* Implement data backup and redundancy mechanisms to ensure data integrity and system reliability.

**19. Continuous Improvement:**

* Continuously monitor and assess the platform's performance and adapt it based on lessons learned and changing environmental conditions.

**CODE IMPLEMENTATION:**

#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](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**:

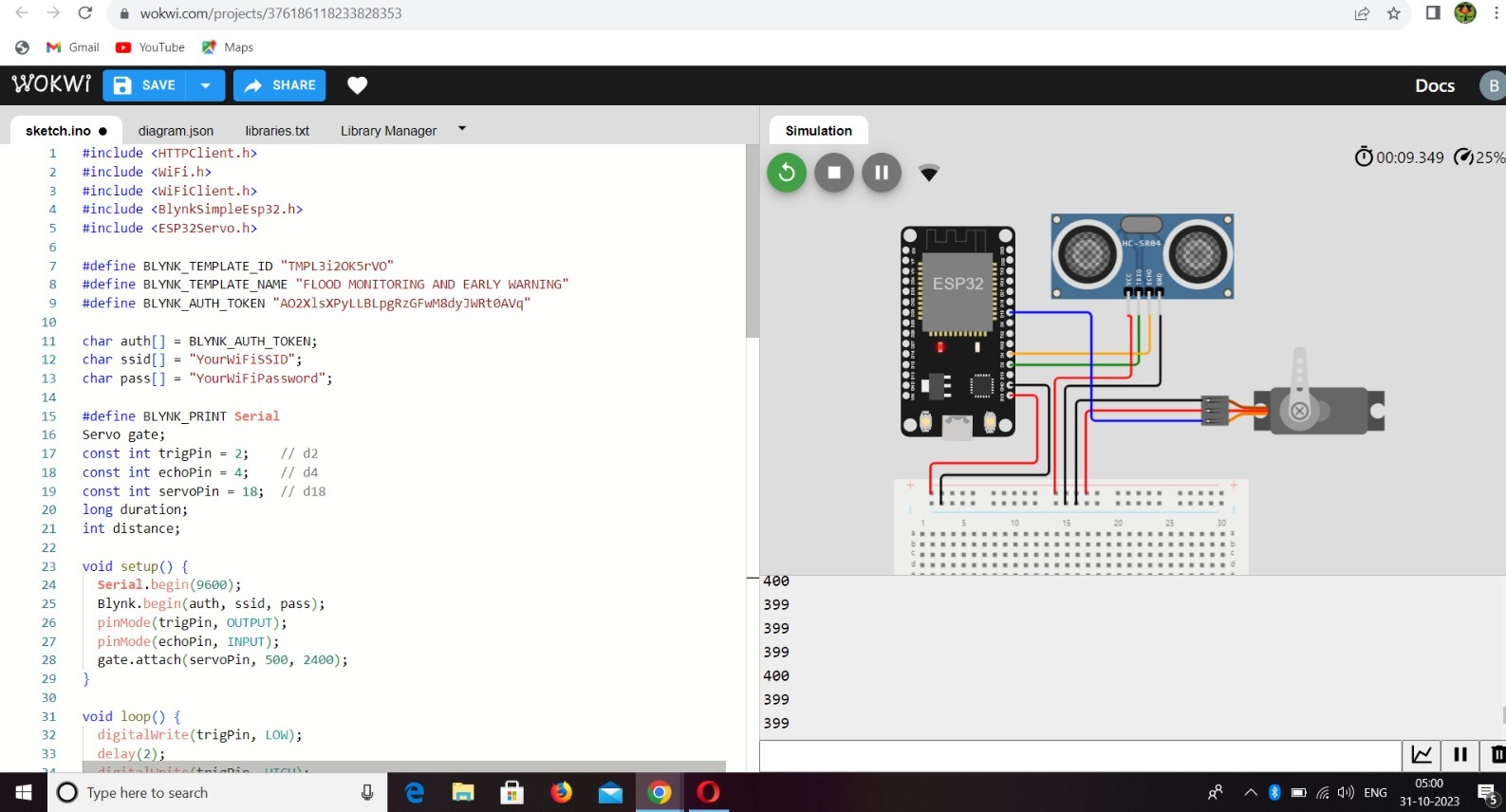
* **HTTPClient.h**: This library is used for making HTTP requests.
* **WiFi.h** and **WiFiClient.h**: These libraries are used for connecting to a Wi-Fi network.
* **BlynkSimpleEsp32.h**: This library provides support for the Blynk IoT platform, which allows you to create a mobile app to control and 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:**



**BEECEPTOR LINK:**

[**"https://phase.free.beeceptor.com"**](https://phase.free.beeceptor.com)

**BY,**

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