**DETAILED EXPLANATION**

To develop a flood monitoring and early warning system, several components and steps need to be considered:

1. **Data collection**: The system should collect data from various sources such as rainfall gauges, river level sensors, weather forecasts, and satellite imagery. This data will be used to monitor the current flood conditions and predict future flood events.

2. **Data processing and analysis**: The collected data needs to be processed and analyzed to identify patterns and trends that indicate potential flood risks. This can involve techniques such as data fusion, statistical analysis, and machine learning algorithms.

3***.* Real-time monitoring**: The system should continuously monitor the data and update the flood conditions in real-time. This can involve the use of IoT devices and sensors to collect data and transmit it to a central monitoring system.

4. **Early warning system**: Based on the analysed data and predefined thresholds, the system should issue timely warnings to the authorities and affected communities. These warnings can be sent through various channels such as SMS notifications, mobile apps, sirens, and public address systems

Implementation of IoT Sensors: IoT sensors are devices designed to measure and collect data related to flood monitoring system.

**PROBLEM DEFINITION:**

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

**FLOOD MONITORING AND EARLY WARNINGSYSTEMS**

5. **Visualization and reporting**: The system should provide a user-friendly interface that allows authorities and the general public to access and understand the flood information easily. This can include interactive maps, graphs, and reports that show the current flood conditions, predicted flood areas, and recommended evacuation routes.

6. **Integration with existing systems**: The flood monitoring and early warning system should be integrated with existing disaster management systems and emergency response mechanisms. This enables effective coordination and response during flood emergencies.

7. **Continuous improvement**: The system should be regularly updated and improved based on feedback and lessons learned from previous flood events. This can involve incorporating new data sources, refining algorithms, and enhancing the user interface.

Design thinking is a user-centric approach that can be applied to the development of a flood monitoring and early warning system. Here is a step-by-step process using design thinking principles:

1***.* Empathize**: Start by understanding the needs and challenges of the end-users, such as authorities responsible for flood management and the general public. Conduct interviews, surveys, and observations to gain insights into their experiences, expectations, and pain points related to flood monitoring and early warning.

2. **Define**: Based on the information gathered, define the specific problems and goals of the project. This could include identifying the key data sources, the accuracy and timeliness requirements of the system, and the desired user experience. Clearly define the scope and objectives of the project.

3. **Ideate**: Brainstorm and generate innovative ideas to address the defined problems and goals. Encourage creativity and collaboration among the team members. Explore different concepts, technologies, and approaches that can be incorporated into the flood monitoringand early warning system.

**DESIGN THINKING**

4. **Prototype**: Create a low-fidelity prototype of the system to visualize and test the proposed ideas. This can be in the form of wireframes, mockups, or even a simple interactive demo. The prototype should focus on the key features and functionality of the system, allowing stakeholders to provide feedback and make necessary improvements.

5. **Test**: Gather feedback from end-users and stakeholders by testing the prototype. Conduct usability tests, interviews, and surveys to understand how well the system meets their needs and expectations. Identify any usability issues, gaps, or areas for improvement.

6. **Iterate**: Based on the feedback and insights gathered during the testing phase, refine and iterate on the design of the system. Make adjustments to the features, user interface, and functionality to address the identified issues and enhance the user experience.

7***.* Implement**: Once the design has been finalized, proceed with the implementation of the flood monitoring and early warning system. Develop the necessary software, hardware, and infrastructure components, ensuring that they align with the user requirements and design specifications.

8.**Evaluate**: Continuously evaluate the performance and effectiveness of the implemented system. Monitor its functionality, accuracy, and reliability in real-world scenarios. Collect feedback from end-users and stakeholders to identify areas for further improvement.

**IMPORTANT COMPONENTS**

# Several important components are necessary for the development of a flood monitoring and early warning system:

* Data collection devices:
* Data transmission and communication:
* Central monitoring system:
* Early warning system:
* User interface and visualization:
* Integration with existing systems
* Continuous monitoring and evaluation:

.****

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

1. We will be using the ESP32 micro controller as well as Arduino UNO microcontroller as both these suit the best for our project

2. We chose this because we only need water level data and post it to a public platform.

3. Non local processing of data is required and hence we chose not to use Raspberry Pi Single board computer

**INNOVATION:**

**SENSORS:**

**Ultrasonic distance sensor HC-SR04**

An ultrasonic distance sensor is a device used in flood monitoring system to measure water level in rivers, lakes, and other bodies of water. It operates by emitting high –frequency sound waves that bounce off the water’s surface and return to the sensor. By measuring the time it takes for the sound waves to travel and return, the sensor calculates the distance to the Water’s surface. The real-time , non-contact measurement data s crucial for early flood detection and issuing timely warnings, helping to mitigate the impact of flooding on communities and infrastructure.



The ESP32 is a popular microcontroller chip known for its integrated Wi-Fi and Bluetooth capabilities. It is widely used in varies internet of things (IOT) and embedded system projects.ESP32 -based development boards provide an affordable and accessible solution for connecting devices to Wi-Fi networks, enabling communication, data transfer, and control over the internet. The ESP32’s Wi-Fi functionality makes it suitable for application such as home automation, sensor monitoring, remote control and more. We can program the ESP32 using the Arduino IDE .

**CONNECTIVITY:**

**Wi-Fi:**

**Sensors**

**CLOUD**

**Beeceptor:**

Beeceptor is a tool that allows you to easily create HTTP endpoints to collect and inspect HTTP requests. It's useful for testing and debugging.

**PROTOCOL**

**HTTP(Hypertext Transfer Protocol):**

HTTP is a foundational communication protocol used in flood monitoring and early warning systems. It facilitates the retrieval ,transmission and sharing of critical data related top water levels, weather conditions and flood risk. HTTP enables remote monitoring real time alerts data visualization and integrating with external services, enhancing the system’s ability to provide timely information and support decision-making for flood managements and early warnings.

One public platform commonly used in flood monitoring and early warning systems is the Global Flood Awareness System (GloFAS). Here are some of its key features:

1. Real-time Monitoring: GloFAS provides real-time monitoring of river flows and flood potential across the globe. It uses weather forecasts, satellite data, and hydrological models to generate flood forecasts and updates.

**PUBLIC TRANSFORM**

**Features:**

2.Early Warning Systems: The platform issues early warnings by predicting flood events up to 30 days in advance. It helps authorities and communities to prepare and respond effectively.

3. Global Coverage: GloFAS covers multiple countries and regions worldwide, allowing for a comprehensive assessment of flood risk on a global scale.

4.Interactive Maps: The platform offers interactive maps that display flood forecast information, river flows, and flood extent. Users can access these maps to visualize flood risk and plan accordingly

5.Data Sharing: GloFAS promotes data sharing and collaboration by providing access to its flood forecast data and information to governments, organizations, and researchers. This facilitates informed decision-making and supports the development of localized flood early warning systems.

6.Integration with Disaster Management Systems: The platform can be integrated with existing disaster management systems and emergency response frameworks, enabling seamless coordination and communication during flood events.

**PHASE 4:**

**Problem:**

As we submitted arduino code in previous phase, here we are submitting our python script along with our project functionalities.

**Python Script:**

import machine

import urequests as requests

import time

# Define your Wi-Fi credentials

ssid = "YourWiFiSSID"

password = "YourWiFiPassword"

# Define the URL of the cloud service where you want to send the data

cloud\_service\_url = "https://example.com/upload\_data" # Replace with the actual URL

# Define GPIO pins for the ultrasonic sensor and servo motor

trig\_pin = machine.Pin(2) # Pin 2 for trigger

echo\_pin = machine.Pin(4) # Pin 4 for echo

servo\_pin = machine.Pin(18) # Pin 18 for servo

# Create an instance of the Servo class (Wokwi-specific)

servo = machine.Servo(servo\_pin)

# Function to measure distance using the ultrasonic sensor

def get\_distance():

trig\_pin.value(1)

time.sleep\_us(10)

trig\_pin.value(0)

while echo\_pin.value() == 0:

pulse\_start = time.ticks\_us()

while echo\_pin.value() == 1:

pulse\_end = time.ticks\_us()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration / 58 # Convert to centimeters

return distance

# Connect to Wi-Fi

import network

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect(ssid, password)

# Wait for Wi-Fi connection

while not sta\_if.isconnected():

pass

print("Connected to Wi-Fi")

try:

while True:

distance = get\_distance()

if distance < 100:

servo.angle(90) # Rotate the servo to 90 degrees

flood\_status = "SAFE CONDITIONS: GATES CLOSED"

else:

servo.angle(0) # Rotate the servo to 0 degrees

flood\_status = "FLOOD DETECTED: GATES OPENED"

# Create a dictionary with the data to send

data = {"flood\_status": flood\_status, "distance": distance}

# Send the data to the cloud service using an HTTP POST request

response = requests.post(cloud\_service\_url, json

**Functionality for Python Script:**

1. ***Wi-Fi Connection:***

- The script connects the simulated ESP32 to a Wi-Fi network using the Wokwi platform's libraries.

- It specifies the SSID (Wi-Fi network name) and password for the network.

- The simulated ESP32 establishes a connection to the specified Wi-Fi network.

2. ***Ultrasonic Distance Measurement:***

- The script utilizes an ultrasonic sensor to measure the distance to an object.

- GPIO pins for the trigger and echo pins of the sensor are defined.

- It sends a brief pulse to the trigger pin and measures the time it takes for the echo pin to go HIGH. This duration is used to calculate the distance.

3. ***Servo Motor Control:***

- The script controls a servo motor, which represents a gate or barrier.

- It defines a GPIO pin for the servo motor.

- Depending on the measured distance from the ultrasonic sensor, the servo angle is set to either 0 degrees (representing an open gate) or 90 degrees (representing a closed gate).

4. ***HTTP Data Transmission:***

- The script prepares data in JSON format to send to a cloud service via HTTP POST requests.

- It sends HTTP POST requests to the specified cloud service URL with the prepared data.

- The data includes information about flood status and the measured distance.

- The cloud service receives the data and processes it for storage and analysis.

**Libraries for Python Script:**

* ***machine (Wokwi-specific):*** The `machine` module provides functions for controlling hardware components in the Wokwi simulation environment, including the servo and GPIO pins.
* ***urequests***: This library is used for making HTTP requests in the script. It is part of MicroPython and provides HTTP client functionality.
* ***time***: The `time` module is used for handling time-related operations, including creating delays in the script.
* ***network***: The `network` library is used for configuring and connecting to Wi-Fi networks. In the script, it's used to establish a connection to the Wi-Fi network.
* ***json***: The `json` module is used to encode data into JSON format. JSON is a common format for structuring data for HTTP requests and responses.

**USING BEECEPTOR CLOUD IN OUR PROJECT FOR DATA STORAGE:**

Beeceptor is a valuable tool for simulating cloud services in software development and prototyping projects.

* **Steps involved:**

1. Create a Beeceptor Endpoint

2. Configure the Endpoint

3. Update the Python Script

4. HTTP Data Transmission

5. Data Analysis and Testing

**Libraries for Python Script:**

To adapt our Arduino code to use Beeceptor cloud, we made some modifications to the code to send HTTP POST requests to our Beeceptor endpoint. Here's a modified Arduino code:

arduino

#include <WiFi.h>

#include <HTTPClient.h>

char ssid[] = "YourWiFiSSID";

char pass [] = "YourWiFiPassword";

const char\* endpoint = "https://your-subdomain.beeceptor.com/your-endpoint-name";

void setup() {

Serial.begin(115200);

delay(4000); // Allow some time to connect to the Serial Monitor

// Connect to Wi-Fi

WiFi.begin(ssid, pass);

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

delay(1000);

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

}

Serial.println("Connected to WiFi");

// Initialize the ultrasonic sensor and servo here

}

void loop() {

// Measure distance and control the servo based on our previous code

// Prepare the data in JSON format

String payload = "{\"flood\_status\": \"SAFE CONDITIONS GATES CLOSED\", \"distance\": " + String(distance) + "}";

// Create an HTTP client object

HTTPClient http;

http.begin(endpoint);

// Set the HTTP headers

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

// Send the POST request

int httpCode = http.POST(payload);

if (httpCode > 0) {

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

String response = http.getString();

Serial.println(response);

} else {

Serial.printf("HTTP request failed: %s\n", http.errorToString(httpCode).c\_str());

}

// End the HTTP session

http.end();

delay(1000); // Adjust the delay as needed

}

In this modified Arduino code:

1. We include the `HTTPClient` library, which allows us to make HTTP POST requests to our Beeceptor endpoint.

2. In the `setup` function, the code connects to Wi-Fi, just like in our previous code. Ensure you have the appropriate Wi-Fi credentials.

3. In the `loop` function, after measuring the distance and controlling the servo (as in our original code), we prepare the data in JSON format. This data represents the flood status and distance.

4. We create an `HTTPClient` object, set the necessary HTTP headers, and send an HTTP POST request to our Beeceptor endpoint. The response and HTTP status code are printed to the serial monitor for debugging.

5. After sending the request, we delay for a certain amount of time (adjust as needed) before sending the next request.

**Python Script:**

import machine

import network

import urequests as requests

import time

import json

# Wi-Fi credentials

ssid = "OurWiFiSSID"

password = "OurWiFiPassword"

# Beeceptor endpoint URL

beeceptor\_url = "https://our-subdomain.beeceptor.com/our-endpoint-name"

# GPIO pins

trig\_pin = machine.Pin(2)

echo\_pin = machine.Pin(4)

servo\_pin = machine.Pin(18)

# Create an instance of the Servo class

servo = machine.Servo(servo\_pin)

# Function to measure distance using the ultrasonic sensor

def get\_distance():

trig\_pin.value(1)

time.sleep\_us(10)

trig\_pin.value(0)

while echo\_pin.value() == 0:

pulse\_start = time.ticks\_us()

while echo\_pin.value() == 1:

pulse\_end = time.ticks\_us()

pulse\_duration = pulse\_end - pulse\_start

distance = pulse\_duration / 58 # Convert to centimeters

return distance

# Connect to Wi-Fi

sta\_if = network.WLAN(network.STA\_IF)

sta\_if.active(True)

sta\_if.connect(ssid, password)

while not sta\_if.isconnected():

pass

print("Connected to Wi-Fi")

try:

while True:

# Measure distance

distance = get\_distance()

# Control the servo based on distance

if distance < 100:

servo.angle(90) # Close the gate

flood\_status = "SAFE CONDITIONS: GATES CLOSED"

else:

servo.angle(0) # Open the gate

flood\_status = "FLOOD DETECTED: GATES OPENED"

# Prepare data in JSON format

data = {"flood\_status": flood\_status, "distance": distance}

# Send HTTP POST request to Beeceptor

headers = {"Content-Type": "application/json"}

response = requests.post(beeceptor\_url, json=data, headers=headers)

print("Data sent to Beeceptor. Response:", response.text)

time.sleep(1) # Adjust the delay as needed

except KeyboardInterrupt:

pass

**Functionalities for Python Script:**

1. ***Wi-Fi Connection:***

- The script connects the ESP32 to a Wi-Fi network using the provided credentials.

2. ***Ultrasonic Distance Measurement:***

- It measures the distance to an object using an ultrasonic sensor, similar to the Arduino code.

3. ***Servo Motor Control:***

- The script controls a servo motor to open or close a gate based on the measured distance.

4. ***HTTP Data Transmission (Beeceptor):***

- The script prepares data in JSON format containing flood status and distance measurements.

- It sends HTTP POST requests to the specified Beeceptor endpoint URL.

- The response from Beeceptor is printed for debugging.

**Libraries for Python Script:**

- ***machine (Wokwi-specific):*** The `machine` module provides functions for controlling hardware components in the Wokwi simulation environment, including the servo and GPIO pins.

- ***urequests***: This library is used for making HTTP requests. It is part of MicroPython and provides HTTP client functionality.

- ***time***: The `time` module is used for handling time-related operations, including creating delays in the script.

- ***network:*** The `network` library is used for configuring and connecting to Wi-Fi networks.

- ***json:*** The `json` module is used to encode data into JSON format, which is then sent in the HTTP POST request to Beeceptor.

This Python script replicates the Arduino's functionality, allowing to measure distance, control a servo motor, and send data via HTTP to a Beeceptor mock endpoint for testing and development.

**BY,**

**Fathima Nabisathul Sameera S(2021504006)**

**Megashree G (2021505022)**

**Srithika S (2021504046)**

**Keerthana K A (2021504017)**

**NaveenaaSuresh Babu (2021504027)**