# **Food Monitoring**

Building an IoT flood monitoring and early warning system requires careful planning and implementation. Here's a step-by-step guide to get you started on this project:

\*\*Step 1: Define Project Requirements\*\*

1. Identify flood-prone areas where you want to deploy IoT sensors.

2. Determine the number and types of sensors required (e.g., water level sensors).

3. Specify the communication protocols for data transmission (e.g., MQTT, HTTP, LoRa).

4. Define the early warning platform and its requirements.

5. Establish the alerting mechanisms (e.g., email, SMS, or a mobile app) for residents and authorities.

\*\*Step 2: Choose Hardware and Sensors\*\*

1. Select appropriate IoT hardware platforms (e.g., Raspberry Pi, Arduino, ESP8266/ESP32) for your sensors.

2. Choose water level sensors that are suitable for flood monitoring, considering factors like range, accuracy, and power requirements.

3. Acquire additional components such as power sources, enclosures, and connectivity modules.

\*\*Step 3: Set Up IoT Sensors\*\*

1. Install and securely mount water level sensors in the identified flood-prone areas.

2. Ensure proper power supply to the sensors.

3. Calibrate the sensors to accurately measure water levels.

\*\*Step 4: Develop Python Script for Sensors\*\*

1. Write Python scripts to run on the IoT sensors. You can use platforms like Raspberry Pi or Arduino IDE.

2. Implement code to read data from the water level sensors.

3. Set up data transmission using a chosen communication protocol.

4. Include error handling and logging in your script to ensure data integrity.

\*\*Step 5: Create the Early Warning Platform\*\*

1. Choose a platform for receiving and processing sensor data, such as a cloud server or a local server.

2. Set up a database to store incoming data.

3. Develop a backend system that processes the data, detects anomalies, and triggers alerts.

4. Implement security measures to protect data transmission and storage.

\*\*Step 6: Implement Alerting Mechanisms\*\*

1. Create a user interface or dashboard for monitoring the data in real-time.

2. Set up alerting mechanisms, including email notifications, SMS alerts, or a mobile app for residents and authorities.

\*\*Step 7: Testing and Calibration\*\*

1. Test the entire system in various scenarios to ensure its accuracy and reliability.

2. Calibrate the sensors regularly to maintain data accuracy.

3. Perform stress testing to evaluate the system's performance under extreme conditions.

\*\*Step 8: Maintenance and Data Storage\*\*

1. Establish a maintenance schedule for the sensors and the overall system.

2. Implement data storage solutions to keep historical data for analysis and research.

\*\*Step 9: Documentation and Training\*\*

1. Document all aspects of the project, including hardware configurations, software code, and maintenance procedures.

2. Provide training to the relevant personnel who will monitor and maintain the system.

\*\*Step 10: Scale and Expand (Optional)\*\*

1. Depending on the success of the initial deployment, consider expanding the network of sensors to cover more flood-prone areas.

Building an IoT flood monitoring and early warning system is a complex project that requires careful planning, hardware selection, software development, and ongoing maintenance. Collaboration with experts in sensor technology, IoT development, and data analysis is crucial for the success of the project.

Flood monitoring sensors are devices designed to measure and monitor various parameters related to floods, such as water level, rainfall, water flow, and other environmental conditions. These sensors play a critical role in flood prediction, detection, and early warning systems, helping to mitigate the impact of floods. Here are some common types of flood monitoring sensors:

1. \*\*Water Level Sensors\*\*:

- Water level sensors are used to measure the depth of water at a specific location, such as a river, lake, or flood-prone area. They come in various types, including ultrasonic, pressure, and float switches.

2. \*\*Rainfall Sensors\*\*:

- Rainfall sensors monitor the amount and intensity of rainfall in an area. This data is essential for assessing the potential for flooding, especially in regions prone to heavy rainfall.

3. \*\*River and Stream Flow Sensors\*\*:

- These sensors measure the flow rate and velocity of water in rivers and streams. Monitoring the flow of water helps in flood prediction and management.

4. \*\*Water Quality Sensors\*\*:

- Water quality sensors assess parameters like water temperature, pH, turbidity, and chemical composition. Changes in water quality can indicate contamination or environmental issues that may contribute to flooding.

5. \*\*Weather Sensors\*\*:

- Weather sensors measure meteorological conditions such as temperature, humidity, wind speed, and atmospheric pressure. This data is crucial for understanding weather patterns that can lead to floods.

6. \*\*Soil Moisture Sensors\*\*:

- Soil moisture sensors measure the moisture content in the ground. High soil moisture levels can indicate a heightened risk of flooding, especially in areas with poor drainage.

7. \*\*Infrared and Radar Sensors\*\*:

- Infrared and radar sensors can provide remote sensing capabilities to monitor large areas. They are often used in conjunction with satellite data to track changes in water levels and flooding.

8. \*\*Acoustic Sensors\*\*:

- Acoustic sensors use sound waves to monitor water levels and can be particularly useful in harsh environmental conditions or in situations where physical contact with the water is challenging.

9. \*\*Communication and Telemetry Systems\*\*:

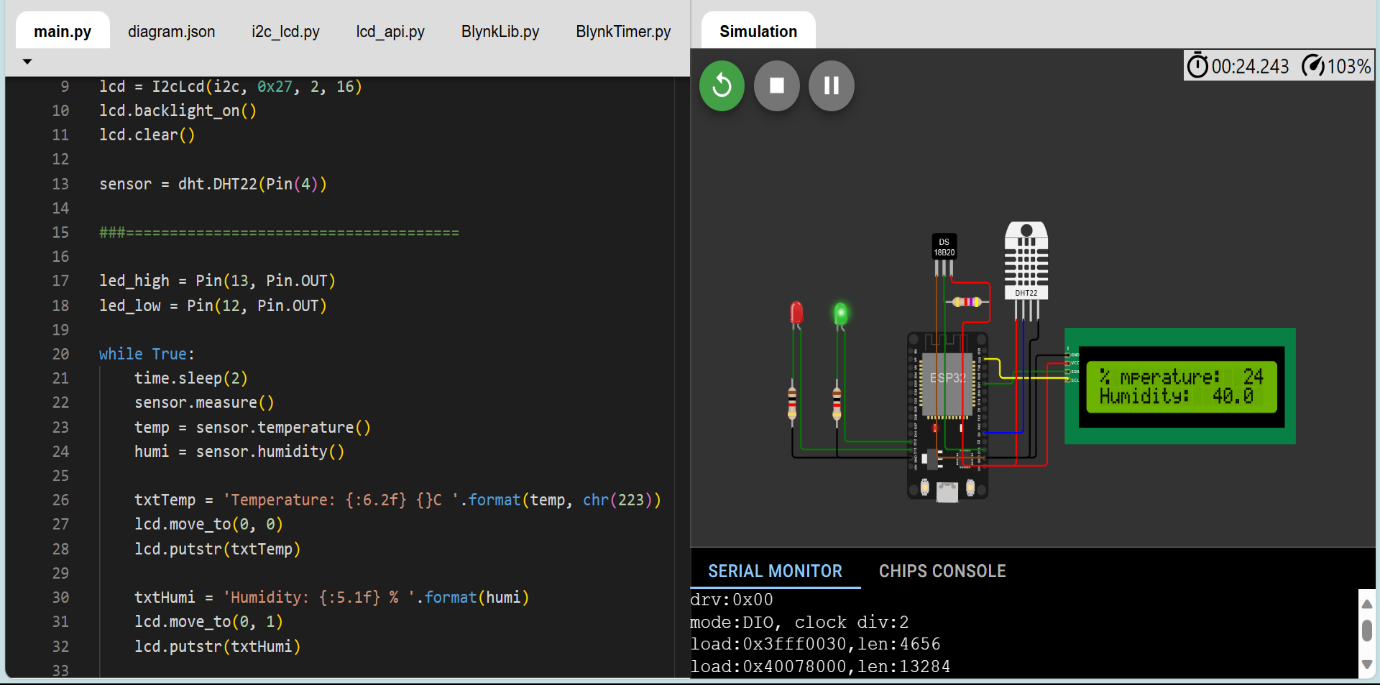
- Communication systems, such as satellite, cellular, or radio communication, are essential for transmitting sensor data to central monitoring and early warning systems.

Flood monitoring sensors are typically integrated into an early warning system, where the collected data is analyzed and used to issue alerts to relevant authorities and residents in flood-prone areas. The combination of various sensor types and advanced data analysis helps in predicting and responding to flood events effectively, potentially saving lives and minimizing property damage.

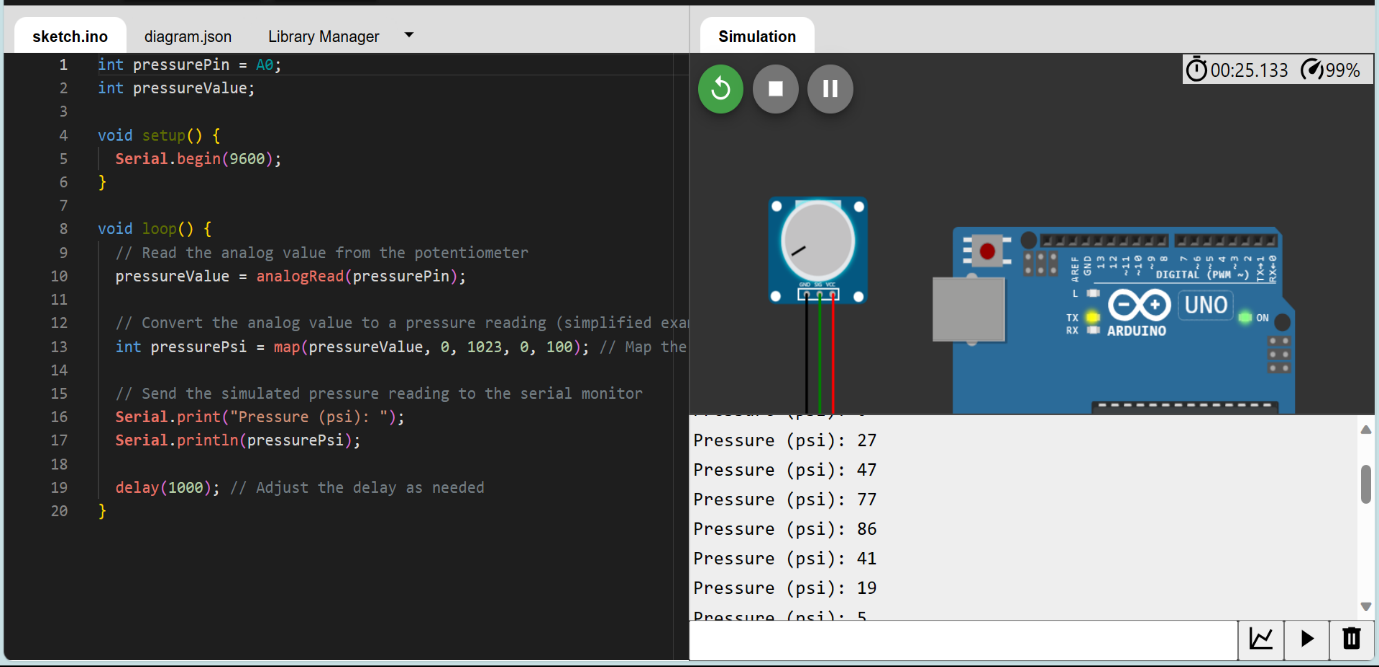
# Configured Sensors

Screenshots:

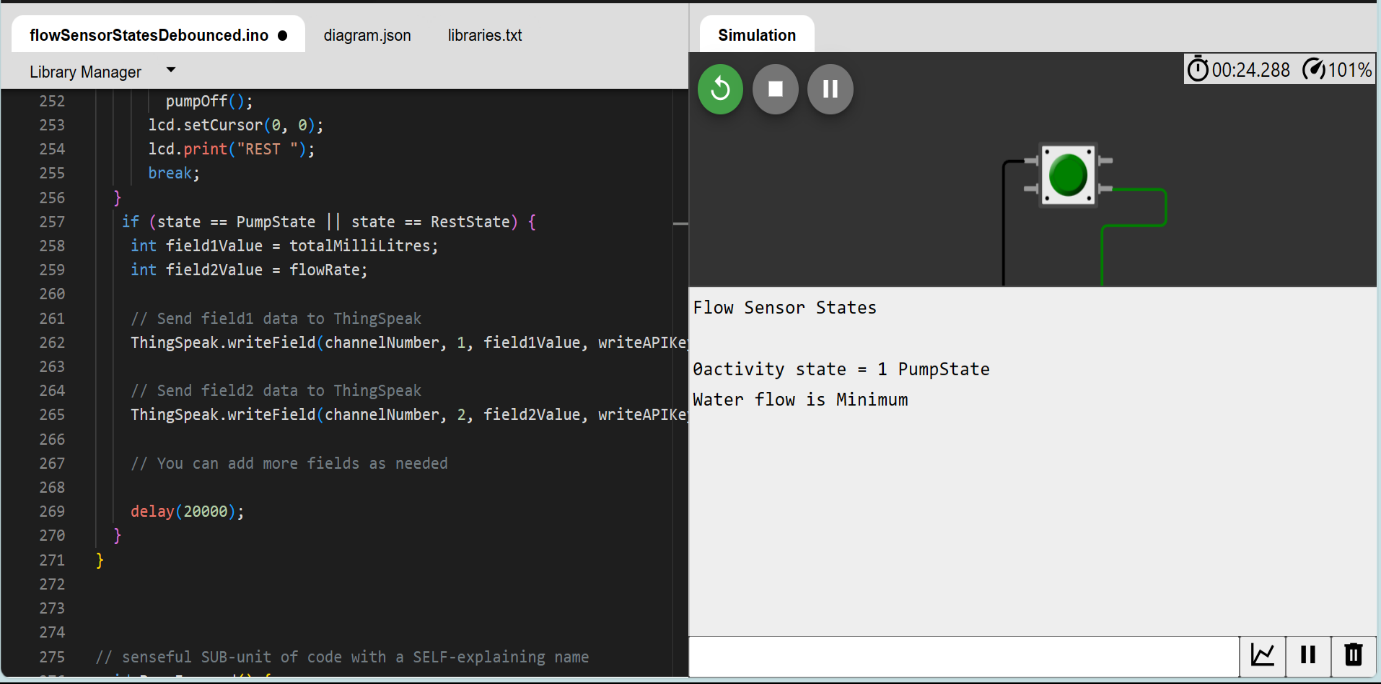
Temperature and Humidity



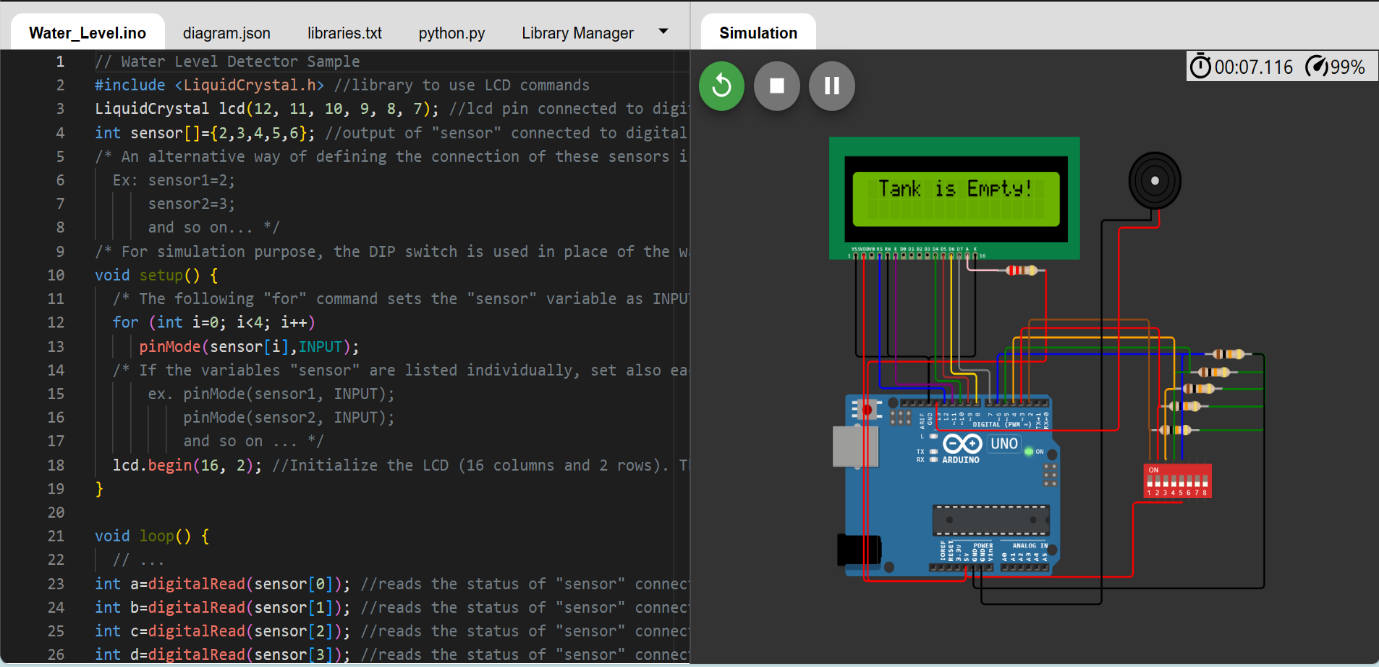
Pressure



Water Flow



Water Level



Codes used available here:

https://drive.google.com/drive/folders/1O8PMTsy5sHP-eAH0xma4c6pbRRohEGnz?usp=share\_link

# Script for data sharing

Python script for sharing data from Temperature and Humidity Sensor

import requests

import time

import json

thingspeak\_url = "https://api.thingspeak.com/update"

api\_key = "6EV4VJEM23TR6EO2"

ssid = "Wokwi-GUEST"

password = ""

DHT\_PIN = 15

TRIG\_PIN = 13

ECHO\_PIN = 12

def get\_distance():

  from machine import Pin

  import dht

  dht\_sensor = dht.DHT22(Pin(DHT\_PIN))

  while True:

    try:

        dht\_sensor.measure()

        temperature = dht\_sensor.temperature()

        humidity = dht\_sensor.humidity()

        distance = get\_distance()

        print("Temperature: {:.2f}°C, Humidity: {:.2f}%, Distance: {:.2f} cm".format(temperature, humidity, distance))

        data = {

            "api\_key": api\_key,

            "field1": temperature,

            "field2": humidity,

            "field3": distance

        }

        response = requests.post(thingspeak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Python script for sharing data from WaterFlow Sensor

import requests

import time

import random

channel\_id = "2306722"

write\_api\_key = "J7UB4P9UTY5Z206M"

thing\_speak\_url = f"https://api.thingspeak.com/update?api\_key={write\_api\_key}"

def simulate\_water\_flow\_data():

    return random.uniform(0, 10)

while True:

    try:

        water\_flow\_rate = simulate\_water\_flow\_data()

        data = {

            "field1": water\_flow\_rate

        }

        response = requests.post(thing\_speak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Python script for sharing data from Water Meter

import requests

import time

import random

channel\_id = "2306899"

write\_api\_key = "VMMZ4TEZVTV6UX8N"

thing\_speak\_url = f"https://api.thingspeak.com/update?api\_key={write\_api\_key}"

def simulate\_water\_meter\_data():

    return random.uniform(0, 1000)

while True:

    try:

        water\_consumption = simulate\_water\_meter\_data()

        data = {

            "field2": water\_consumption

        }

        response = requests.post(thing\_speak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Python script for sharing data from Water Level Sensor

import requests

import time

import machine

from machine import Pin

thingspeak\_url = "https://api.thingspeak.com/update"

channel\_id = "2306899"

api\_key = "VMMZ4TEZVTV6UX8N"

sensor\_pins = [2, 3, 4, 5, 6]

lcd = machine.LCD()

lcd.init()

def read\_sensor\_state():

    sensor\_states = [digitalRead(pin) for pin in sensor\_pins]

    return sensor\_states

def get\_water\_level(sensor\_states):

    if all(sensor\_states):

        return "Overflowing"

    elif sensor\_states[0]:

        return "Tank is Full"

    elif sensor\_states[1]:

        return "Tank is 75% Full"

    elif sensor\_states[2]:

        return "Tank is 50% Full"

    elif sensor\_states[3]:

        return "Tank is 25% Full"

    else:

        return "Tank is Empty"

while True:

    try:

        sensor\_states = read\_sensor\_state()

        water\_level = get\_water\_level(sensor\_states)

        lcd.text(water\_level)

        data = {

            "api\_key": api\_key,

            "field1": water\_level

        }

        response = requests.post(thingspeak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Python script for sharing data from Leak Detection Sensor

import requests

import time

import random

channel\_id = "2309000"

write\_api\_key = "28WR12W5X9OSKXY2"

thing\_speak\_url = "https://api.thingspeak.com/update?api\_key={write\_api\_key}"

def simulate\_leak\_data():

    return random.choice([0, 1])

while True:

    try:

        leak\_status = simulate\_leak\_data()

        data = {

            "field1": leak\_status

        }

        response = requests.post(thing\_speak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Python script for sharing data from Pressure Sensors

import requests

import time

import random

channel\_id = "2310757"

write\_api\_key = "OL8FZEGPB5KK8DCW"

thing\_speak\_url = f"https://api.thingspeak.com/update?api\_key={write\_api\_key}"

def simulate\_pressure\_data():

    return random.uniform(0, 100)

while True:

    try:

        pressure\_value = simulate\_pressure\_data()

        data = {

            "field1": pressure\_value

        }

        response = requests.post(thing\_speak\_url, data=data)

        print("Data sent to ThingSpeak. Status code:", response.status\_code)

    except Exception as e:

        print("Error:", str(e))

    time.sleep(15)

Flood Monitoring and Early Warning

# Project Objectives

The project’s objectives for creating a platform that displays real-time water level data and issues flood warnings to provide a platform that displays real-time water level data from various locations and to issue flood warnings when water levels exceed predefined thresholds, ensuring public safety.

Frontend Web Interface

Use HTML, CSS, and JavaScript to create the user interface.

Display a map or list with markers or entries for different monitoring locations.

Implement interactive features for users to view water levels and receive warnings.

Index.html

<!DOCTYPE html>

<html>

<head>

<title>Water Level Monitor</title>

<link rel=”stylesheet” type=”text/css” href=”style.css”>

</head>

<body>

<div id=”map”></div>

<div id=”alerts”></div>

<script src=”script.js”></script>

</body>

</html>

Style.css

#map {

width: 100%;

height: 400px;

}

#alerts {

margin: 10px;

padding: 10px;

background-color: #f7f7f7;

}

Script.js

Const sensorData = [

{ location: ‘Sensor A’, waterLevel: 2.5 },

{ location: ‘Sensor B’, waterLevel: 3.7 },

// Add more sensor data here

];

// Function to update the map with sensor data

Function updateMap() {

Const mapElement = document.getElementById(‘map’);

}

// Function to check for flood warnings and display alerts

Function checkFloodWarnings() {

Const alertsElement = document.getElementById(‘alerts’);

alertsElement.innerHTML = ‘’; // Clear previous alerts

for (const data of sensorData) {

if (data.waterLevel > 3.0) {

// Display a flood warning

Const alertMessage = `Flood warning at ${data.location}! Water level: ${data.waterLevel} m`;

Const alertDiv = document.createElement(‘div’);

alertDiv.textContent = alertMessage;

alertsElement.appendChild(alertDiv);

}

}

}

// Initial update

updateMap();

checkFloodWarnings();

# Backend Server

Developing a backend server using a technology like Node.js.

Setting up REST APIs for receiving and serving data.

Handle data from IoT sensors, such as water level measurements.

Server.js

Const express = require(‘express’);

Const app = express();

Const port = process.env.PORT || 3000;

App.listen(port, () => {

Console.log(`Server is running on port ${port}`);

});

# Database

Use a database to store sensor data.(used MongoDB)

Create tables for location information, historical data, and alerts.

Use MongoDB’s querying capabilities to retrieve historical data for visualization.

Retrieve the most recent data for real-time updates on the frontend

// Import required libraries

const express = require('express');

const mongoose = require('mongoose');

const bodyParser = require('body-parser');

const http = require('http');

const socketIO = require('socket.io');

// Initialize Express app

const app = express();

// Set up MongoDB connection

mongoose.connect('mongodb://localhost/your-database-name', {

useNewUrlParser: true,

useUnifiedTopology: true,

});

// Define MongoDB schema and model

const SensorDataSchema = new mongoose.Schema({

location: String,

timestamp: Date,

waterLevel: Number,

});

const SensorData = mongoose.model('SensorData', SensorDataSchema);

// Set up Express middleware

app.use(bodyParser.urlencoded({ extended: true }));

app.use(bodyParser.json());

// Define REST API routes

// Route to save sensor data

app.post('/api/sensor-data', (req, res) => {

const { location, timestamp, waterLevel } = req.body;

const sensorData = new SensorData({ location, timestamp, waterLevel });

sensorData.save((err, data) => {

if (err) {

res.status(500).send(err);

} else {

// Notify connected clients about new data

io.sockets.emit('new-data', data);

res.status(201).send('Data saved');

}

});

});

// Route to get historical sensor data

app.get('/api/sensor-data', (req, res) => {

SensorData.find({}, (err, data) => {

if (err) {

res.status(500).send(err);

} else {

res.json(data);

}

});

});

// Set up HTTP server

const server = http.createServer(app);

// Set up WebSocket for real-time updates

const io = socketIO(server);

io.on('connection', (socket) => {

console.log('Client connected');

});

// Start the server

const port = process.env.PORT || 3000;

server.listen(port, () => {

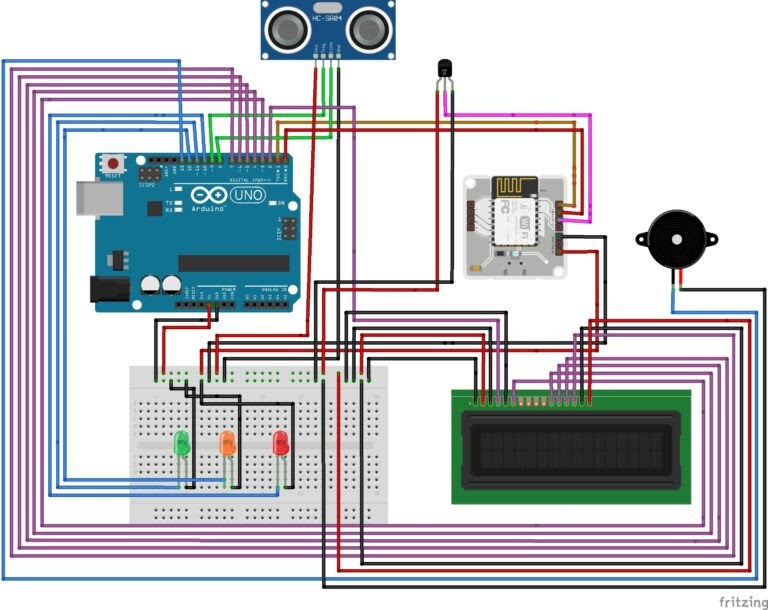
console.log(`Server is running on port ${port}`);

});

IoT Sensors

Set up IoT sensors to measure water levels.

Transmit data to the server through protocols like MQTT or HTTP.



Integrate.js

const channelID = '';

const apiKey = '';

const fieldNumber = 1; // Adjust to the field number where water level data is stored in your ThingSpeak channel

// Function to fetch data from ThingSpeak

async function fetchThingSpeakData() {

try {

const response = await fetch(`https://api.thingspeak.com/channels/${channelID}/fields/${fieldNumber}.json?results=1&api\_key=${apiKey}`);

const data = await response.json();

if (data && data.feeds && data.feeds.length > 0) {

const latestData = data.feeds[0];

return parseFloat(latestData[`field${fieldNumber}`]);

}

} catch (error) {

console.error('Error fetching ThingSpeak data:', error);

}

return null;

}

// Function to update the map and check flood warnings using ThingSpeak data

async function updateMapAndCheckWarnings() {

const waterLevel = await fetchThingSpeakData();

if (waterLevel !== null) {

updateMap();

if (waterLevel > 3.0) {

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

alertsElement.innerHTML = `Flood warning! Water level: ${waterLevel} m`;

}

}

}

// Initial update

updateMapAndCheckWarnings();

Real-Time Data Updates

Implement WebSocket or Server-Sent Events (SSE) for real-time updates.

Push sensor data to the frontend in real-time as measurements change.

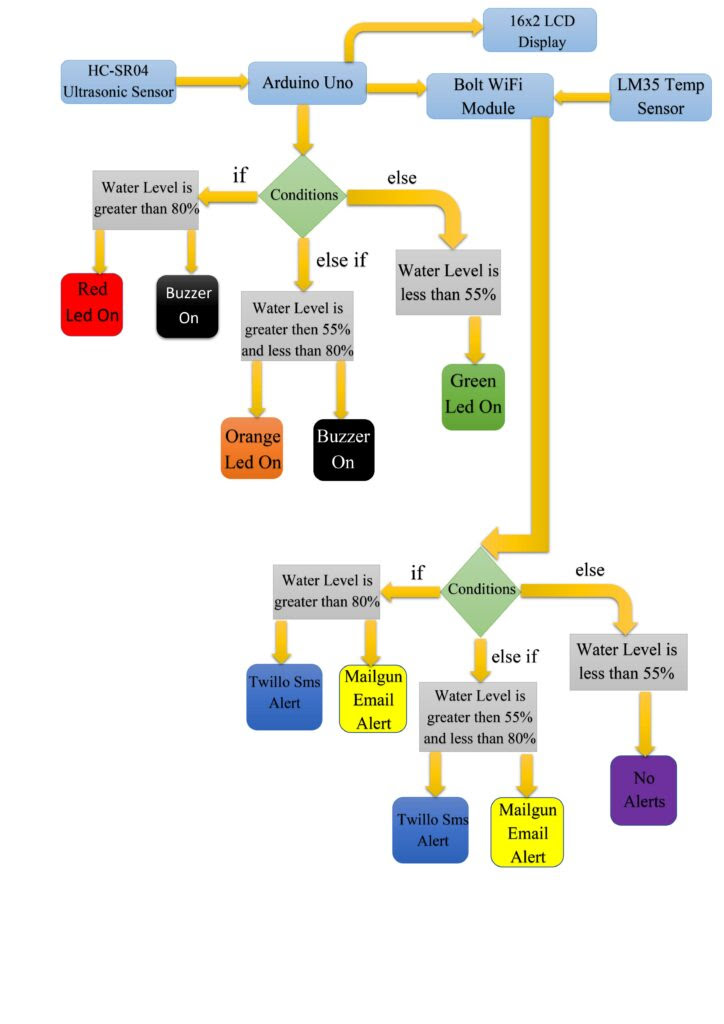
Flood Warning System

Define threshold levels for flooding at each location.

Continuously monitor water levels and compare them to thresholds.

Issue flood warnings when levels exceed thresholds.

Notify users through the frontend



# Conclusion

By following the outlined steps and utilizing the mentioned technologies, you can design a system that receives and displays water level data from IoT sensors and issues flood warnings when necessary. Remember to focus on user-friendly interfaces, data accuracy, security, and scalability to provide an effective solution for monitoring and addressing potential flood situations. This project can greatly contribute to public safety and environmental monitoring.