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, () => {

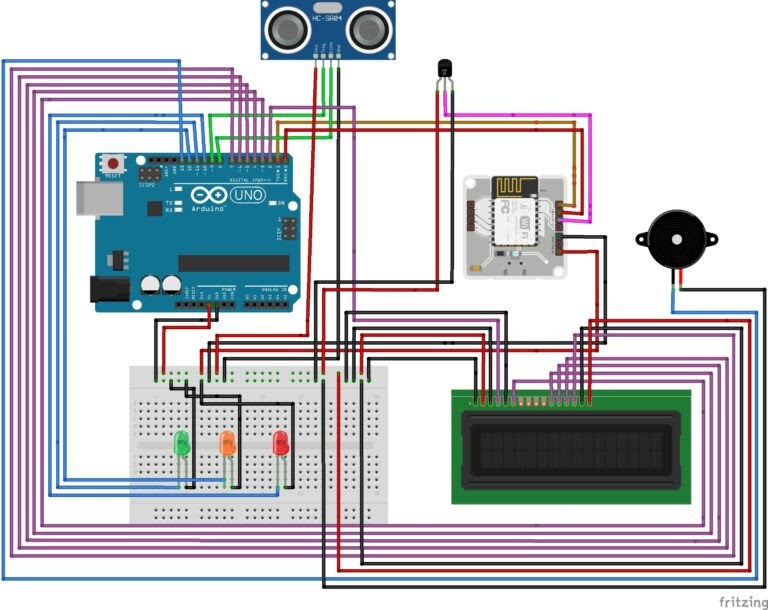
console.log(`Server is running on port ${port}`);

});

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.

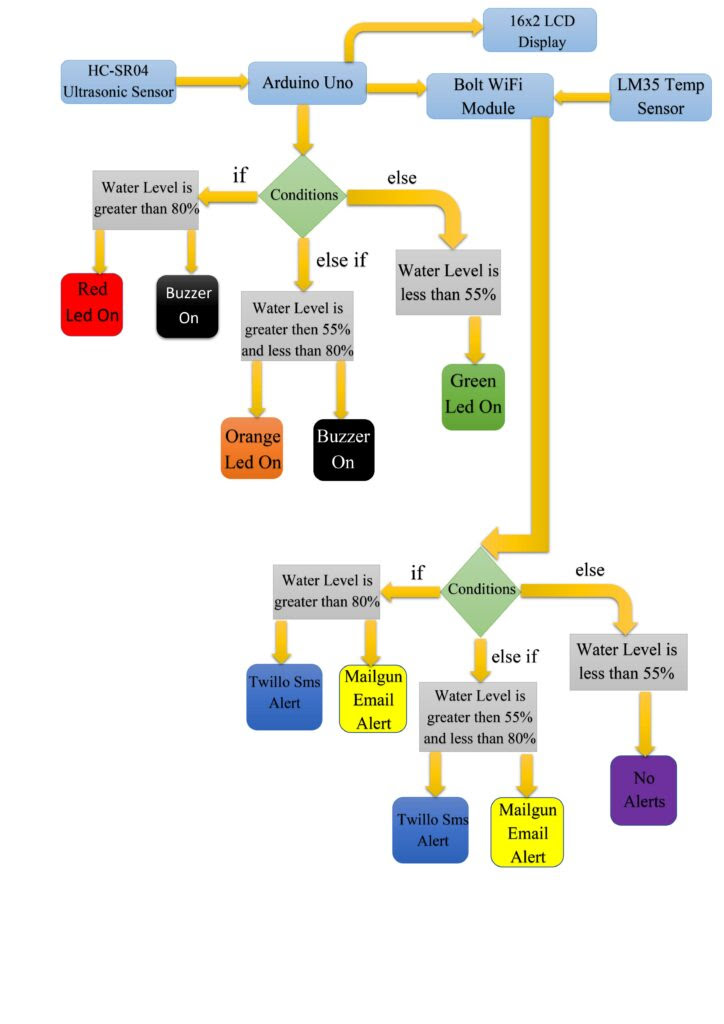
Flood Warning System

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.