**PHASE 4**

**Development part 2**

**Submitted by :**

**Name :Gajjala Harsha Kesava Reddy**

**NMid:au723921244020**

**Gmai:** [**harshagajjala122@gmail.com**](mailto:harshagajjala122@gmail.com)

**Building an early warning platform for real-time water level data and flood warnings using web development technologies like HTML, CSS, and JavaScript is a multi-step process. I'll outline the key components and steps you can follow to create this platform:**

**1. Define Requirements**

- Begin by defining the specific requirements of your project, including the IoT sensor data format, the criteria for issuing flood warnings, and the user interface design.

**2. Data Collection:**

- Set up IoT sensors to collect real-time water level data. These sensors should transmit data to a central server or database.

**3. Server-Side Development:**

- Create a server or backend using technologies like Node.js, Python, or any other language of your choice to handle data from the sensors. This server should process the data, apply the flood warning criteria, and store it in a database.

**4. Database:**

- Choose a database system (e.g., MySQL, MongoDB) to store the water level data. Design a database schema to efficiently store and retrieve sensor data.

**5. API Development:**

- Develop APIs on the server to serve real-time data to the web platform. These APIs will be responsible for fetching and updating the water level data.

**6. Web Platform Development:**

- Create the front-end web platform using HTML, CSS, and JavaScript. Here are some key components to consider:

- Dashboard: Design an interactive dashboard to display the real-time water level data.

- Map Interface: You can use a map library like Leaflet or Google Maps to show sensor locations and water levels.

- Alert System: Implement an alert system that monitors the data and issues flood warnings based on predefined criteria. This could be a simple JavaScript function that triggers alerts when thresholds are crossed.

- Data Visualization: Use JavaScript libraries like Chart.js or D3.js to create charts or graphs for visualizing historical data.

- User Authentication: Implement user authentication and access control to ensure that only authorized users can view and interact with the data.

- Notifications: Send notifications to users via email or push notifications when flood warnings are issued.

**7. Real-Time Updates:**

- Use technologies like WebSockets or Server-Sent Events (SSE) to enable real-time updates on the platform without the need for manual refreshing.

**8. Mobile Responsiveness:**

- Ensure that your platform is responsive and works on various devices and screen sizes, including mobile phones and tablets.

**9. Testing:**

- Thoroughly test your platform to ensure that it works correctly, including the flood warning system and real-time data updates.

**10. Deployment:**

- Deploy the server and database to a hosting service (e.g., AWS, Heroku, or Azure) and the front-end platform to a web hosting provider. Secure your server and database with proper authentication and encryption.

**11. User Training and Documentation:**

- Create user documentation and provide training to users who will be using the platform, including how to interpret flood warnings and use the interface effectively.

**12. Ongoing Maintenance:**

- Regularly maintain and update the platform to ensure it continues to operate reliably. Monitor the performance of sensors, server, and database to address any issues promptly.

Remember to keep data security and privacy in mind, as real-time water level data may have sensitive implications. Implement appropriate security measures to protect the data and ensure that user access is controlled. Additionally, consider data retention policies and compliance with relevant regulations.

**Design the platform to receive and display water level data from IoT sensors and issue flood warnings when necessary.**

As we all know that Flood is one of the major well known Natural Disasters. When water level suddenly rises in dams, river beds etc. Alot of Destruction happens at surrounding places. It causes a huge amount of loss to our environment and living beings as well. So in these case, it is very important to get emergency alerts of the water level situation in different conditions in the river bed.  
  
The purpose of this project is to sense the water level in river beds and check if they are in normal condition. If they reach beyond the limit, then it alerts people through LED signals and buzzer sound. Also it alerts people through Sms and Emails alerts when the water level reaches beyond the limit.

**Things used in this project**

**Hardware components -**

1. Bolt-IoT wifi module
2. Arduino uno
3. Breadboard- 400 tie points
4. 5mm LED:(Green, Red, Orange) and Buzzer
5. 16×2 LCD Display
6. LM35 Temperature Sensor
7. HC-SR04 Ultrasonic Sensor
8. Some Jumper Wires
   1. Male to Female Jumper Wires- 15 pcs
   2. Male to Male Jumper Wires- 10 pcs
   3. Female to Female Jumper Wires- 5 pcs
9. 9v Battery and Snap Connector
10. USB Cable Type B

**Software components -**

1. [Arduino IDE](https://www.arduino.cc/en/software)
2. [Python 3.7 IDLE](https://www.python.org/downloads/)
3. [Bolt IoT Cloud](https://cloud.boltiot.com/)
4. [Bolt IoT Android App](https://play.google.com/store/apps/details?id=com.bolt.com.bolt)
5. [Twillo SMS Messaging API](https://www.twilio.com/)
6. [Mailgun EMAIL Messaging APISoftware components](https://www.mailgun.com/)

**Hand tools and fabrication machines**

1. Electrical Tape
2. Green Cello Tape

**Hardware Setup**

For Building this project we first configure the hardware connections. Then later on moving to the software part.

**Designing a platform to receive and display water level data from IoT sensors and issue flood warnings using a Python program involves several components. Below is a high-level design for such a platform using Python:**

**1. Sensor Data Ingestion:**

- IoT sensors continuously collect water level data and transmit it to a central server. Python can be used to set up a server that listens for incoming sensor data.

**2. Server-Side Components:**

- The server-side components in Python handle data ingestion, processing, and flood warning logic.

- Backend Server: Create a Python server using a framework like Flask, Django, or FastAPI. This server should handle incoming data from IoT sensors.

- Database:Store the incoming sensor data in a database. You can use SQLite, PostgreSQL, or any other database system.

- Data Processing:Implement data processing routines in Python to validate and normalize the incoming data. You can calculate statistics and detect anomalies.

- Flood Warning Engine: Develop a Python script that continuously monitors the water level data. When predefined criteria are met, the engine triggers flood warnings**.**

**3. Alerting System:**

- When the flood warning engine detects a potential flood event, it should initiate the alerting process in Python.

- Notifications: Use Python libraries for sending notifications, such as smtplib for email notifications, Twilio for SMS, or push notification services like Firebase Cloud Messaging (FCM) for push notifications.

- Logging: Implement a logging mechanism in Python to record flood warnings and alerts.

**4. Front-End Web Platform:**

- For displaying water level data and flood warnings, you can use a Python web framework like Flask or Django for the front-end. Design it with a user-friendly interface.

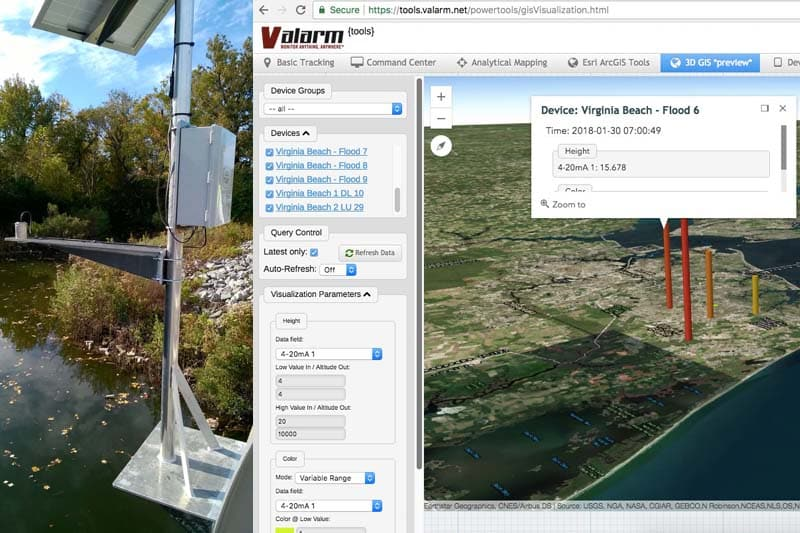
- Dashboard: Create a web page that displays water level data in real-time, shows sensor locations on a map, and presents historical data in charts.

- Alert Section: Dedicate a section to display flood warnings and alerts, which should be prominently visible to users.

- User Interaction: Allow users to interact with the data, zoom in on maps, view historical trends, and configure their alert preferences.

- User Authentication: Implement user authentication and role-based access control in Python.

-Mobile Responsiveness: Ensure that the web platform is responsive and accessible on various devices.

**5. Real-Time Updates:** - Implement real-time updates of water level data on the platform using WebSocket libraries like Flask-SocketIO or Django Channels. 

**6. Data Storage and Historical Analysis:**

- Store historical data in a database for long-term analysis and trend recognition. Python scripts can be used for data retrieval and analysis**.**

**7. User Notifications and Preferences:**

- Implement user notification preferences within the web platform using Python for user management and settings.

**8. Deployment and Security:**

- Deploy the Python web application on a secure hosting environment, ensuring data encryption, access control, and regular security updates.

**9. Testing and Monitoring:**

- Thoroughly test the Python components for data accuracy, performance, and reliability. Implement monitoring tools to detect issues and anomalies.

**10. Documentation and Training:**

**-** Provide documentation and training for users on how to interpret flood warnings and use the platform effectively.

By following this design and implementing the various components in Python, you can create a platform that effectively receives and displays water level data from IoT sensors and issues flood warnings when necessary.

//IOT Based Flood Monitoring And Alerting System.

#include<LiquidCrystal.h>

**Code**

LiquidCrystal lcd(2, 3, 4, 5, 6, 7);

const int in = 8;

const int out = 9;

const int green = 10;

const int orange = 11;

const int red = 12;

const int buzz = 13;

void setup() {

Serial.begin(9600);

lcd.begin(16, 2);

pinMode( in , INPUT);

pinMode(out, OUTPUT);

pinMode(green, OUTPUT);

pinMode(orange, OUTPUT);

pinMode(red, OUTPUT);

pinMode(buzz, OUTPUT);

digitalWrite(green, LOW);

digitalWrite(orange, LOW);

digitalWrite(red, LOW);

digitalWrite(buzz, LOW);

lcd.setCursor(0, 0);

lcd.print("Flood Monitoring");

lcd.setCursor(0, 1);

lcd.print("Alerting System");

delay(5000);

lcd.clear();

}

void loop() {

long dur;

long dist;

long per;

digitalWrite(out, LOW);

delayMicroseconds(2);

digitalWrite(out, HIGH);

delayMicroseconds(10);

digitalWrite(out, LOW);

dur = pulseIn( in , HIGH);

dist = (dur \* 0.034) / 2;

per = map(dist, 10.5, 2, 0, 100);

#map

function is used to convert the distance into percentage.

if(per < 0) {

per = 0;

}

if (per > 100) {

per = 100;

}

Serial.println(String(per));

lcd.setCursor(0, 0);

lcd.print("Water Level:");

lcd.print(String(per));

lcd.print("% ");

if (per >= 80) #MAX Level of Water--Red Alert!{

lcd.setCursor(0, 1);

lcd.print("Red Alert! ");

digitalWrite(red, HIGH);

digitalWrite(green, LOW);

digitalWrite(orange, LOW);

digitalWrite(buzz, HIGH);

delay(2000);

digitalWrite(buzz, LOW);

delay(2000);

digitalWrite(buzz, HIGH);

delay(2000);

digitalWrite(buzz, LOW);

delay(2000);

}

else if (per >= 55) #Intermedite Level of Water--Orange Alert!{

lcd.setCursor(0, 1);

lcd.print("Orange Alert! ");

digitalWrite(orange, HIGH);

digitalWrite(red, LOW);

digitalWrite(green, LOW);

digitalWrite(buzz, HIGH);

delay(3000);

digitalWrite(buzz, LOW);

delay(3000);

}

else #MIN / NORMAL level of Water--Green Alert!{

lcd.setCursor(0, 1);

lcd.print("Green Alert! ");

digitalWrite(green, HIGH);

digitalWrite(orange, LOW);

digitalWrite(red, LOW);

digitalWrite(buzz, LOW);

}

delay(15000);

}