**Phase 4**

**FLOOD MONITORING AND EARLY WARNING**

**Development Part-2**

Web-based Dashboard for Administrators:

Create a web-based dashboard for administrators to monitor and manage the parking system. This dashboard should provide real-time information about parking spot occupancy, reservations, and transaction history. Use web development technologies like HTML, CSS, and JavaScript, and consider using a web framework for efficiency.

HTML/CSS:

Design the dashboard's layout and style using HTML and CSS.

JavaScript:

Implement interactivity for real-time updates, charts, and user management.

Web Framework:

You can use popular frameworks like React, Angular, or Vue.js for a organized and responsive interface.

Mobile App:

Develop a mobile app to reserve parking spots, make payments, and receive notifications. Use cross-platform mobile app development frameworks like React Native or Flutter to streamline app development for both Android and iOS.

React Native or Flutter:

Build the app's frontend using these frameworks, which allow you to write code once and deploy it on multiple platforms.

API Integration:

Connect the app to the backend server for user authentication, reservation processing, and payment handling.

Online Reservation System:

Implement a web-based reservation system for students to check parking spot availability and make reservations. This system can be integrated with the mobile app and can be developed using standard web technologies.

HTML/CSS:

Design the reservation interface.

JavaScript:

Develop interactive features, such as selecting a parking spot and specifying the reservation duration.

Backend:

Implement reservation logic on the server side, making use of frameworks like Express.js (Node.js) or Django (Python).

Payment Gateway Integration:

If you include a payment system, you'll need to integrate a payment gateway into your web app for processing payments. Popular payment gateways often provide APIs for this purpose. Here's a simplified example using Python and Flask:

Flask:

Create an API endpoint to handle payment requests.

Payment Gateway API:

Utilize the API provided by the payment gateway provider (e.g., Stripe, PayPal) for processing payments.

Frontend Integration:

Integrate the payment process into your mobile app or web app, allowing users to enter payment details securely.

Real-time Updates:

Use web development technologies to ensure real-time updates on parking spot availability, reservation confirmation, and payment status. You can achieve this with technologies like WebSocket for real-time communication between the server and clients.

WebSocket:

Implement WebSocket communication to push real-time updates to the web and mobile clients when a parking spot's status changes.

User Authentication and Management:

For user authentication and management, you can create user registration and login systems within the mobile app and web interface. Use web development technologies for user interfaces and backend logic:

HTML/CSS:

Design registration and login forms.

JavaScript:

Implement form validation and submission handling.

Backend:

Create user accounts, manage authentication, and store user data securely in a database.

Data Analytics and Reporting:

Utilize web technologies to create data analytics and reporting features for administrators. You can use JavaScript libraries for data visualization and reporting tools.

Here to learn how to build an IoT Based Flood Monitoring and Email, SMS alert System using Ultrasonic, NodeMCU ESP8266 12E, myDevices Cayenne. In more detail, this IoT tutorial discovers how to use an ESP8266 to send data to Cayenne using the MQTT protocol.

Abstract: Flood is one of the natural disasters that cannot be avoided. It happens too fast and affected so many lives and properties. Before this, most of the existing system that has been developed are only focus on certain areas. Other than that, majority of the public cannot monitor and have no idea when the flood going to be happened since they do not have any information and data about the weather condition. This system is suitable for cities and village areas. Furthermore, if the public has an internet access, they can monitor what is happening and predict if there is any upcoming flood at the web server. This project will update the water level at the web server and the system will issue an alert signal to the citizens for evacuation so that fast necessary actions can be taken.

This is a complete step-by-step tutorial on building an IoT system using Cayenne and ESP.

On the other hand, Cayenne is an IoT cloud platform that provides several cloud services, such as:

Data visualization IoT cloud

Alerts

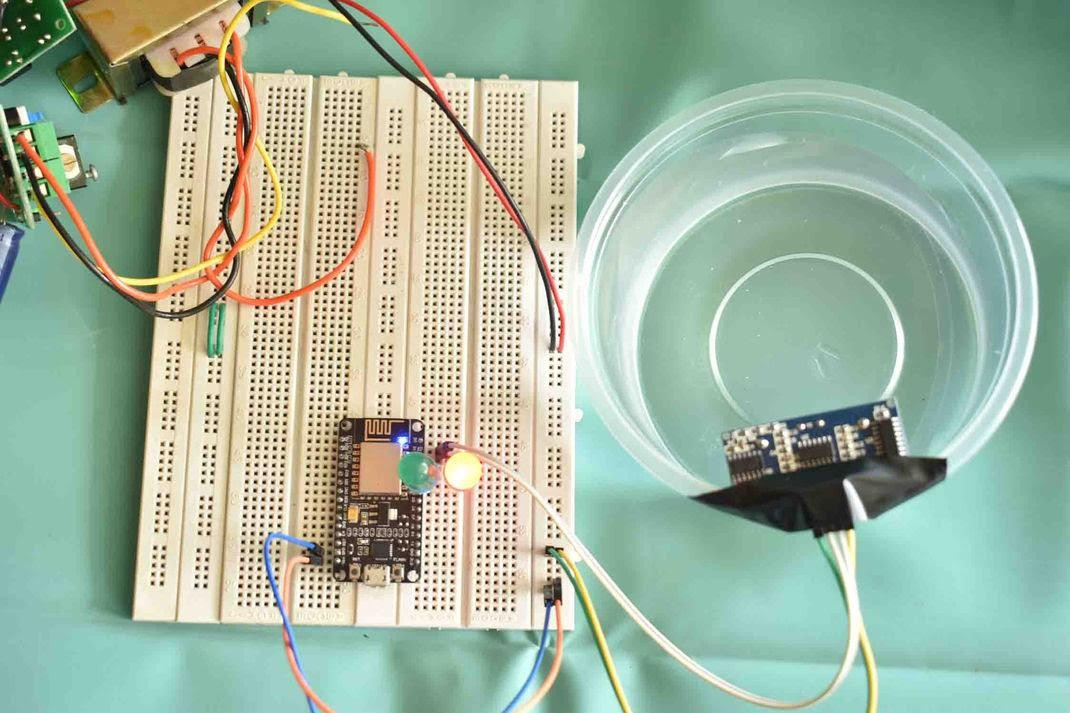
Scheduling Events

We will focus our attention on data visualization and on the IoT cloud services.

Cayenne IoT Platform accelerates the development of IoT-based solutions, including quick design, prototyping and other commercialized projects. It is a drag-and-drop IoT project builder that can help developers build complete, ready-to-use IoT solutions with little to no coding.

Cayenne IoT Platform contains a vast catalogue of certified IoT-ready devices and connectivity options. This allows users to easily add any device to the library utilizing MQTT API. All devices in Cayenne are interoperable and benefit from features such as rules engine, asset tracking, remote monitoring and control, and tools to visualize real-time and historical data.

MQTT is a lightweight messaging protocol for sensors and mobile devices.



**Components Required** ESP8266 NodeMCU

Ultrasonic Sensor LEDs (Red & Green) Breadboard Jumpers

**NodeMCU**

NodeMCU ESP8266-12E MCU is a development board with one analogue and many general-purpose input output (GPIO) pins. It has 4MB flash memory, and can operate at a default clock frequency of 80MHz. In this project, digital pin D1 of NodeMCU is used to read Data of Dht11 temperature sensor.

Ultrasonic:

The HC-SR04 ultrasonic module is a module that can provide non-contact measurement within the range of 2cm to 400cm with ranging accuracy that can reach 3mm. It works on the principle of echolocation.

The ultrasonic sensor as a trigger and an echo pin. The arduino provides a high signal of 10microseconds to this pin. After the HC-SR04 is triggered, it sends out eight 40Khz sound waves to the surface of the water. On getting to the surface of the water, the wave is echoed back to the sensor and the ESP8266 reads the echo pin to determine time spent between triggering and receiving of the echo. Since we know that the speed of sound is around 340m/s then we can calculate the distance using;

Distance = (time/2)\*speed of sound

Ultrasonic HC-SR04 wiring to ESP8266 Ultrasonic HC-SR04 ESP8266

Vcc Pin Vin Pin

Trig Pin D1 (GPIO 5) Echo Pin D2 (GPIO 4) GND Pin GND

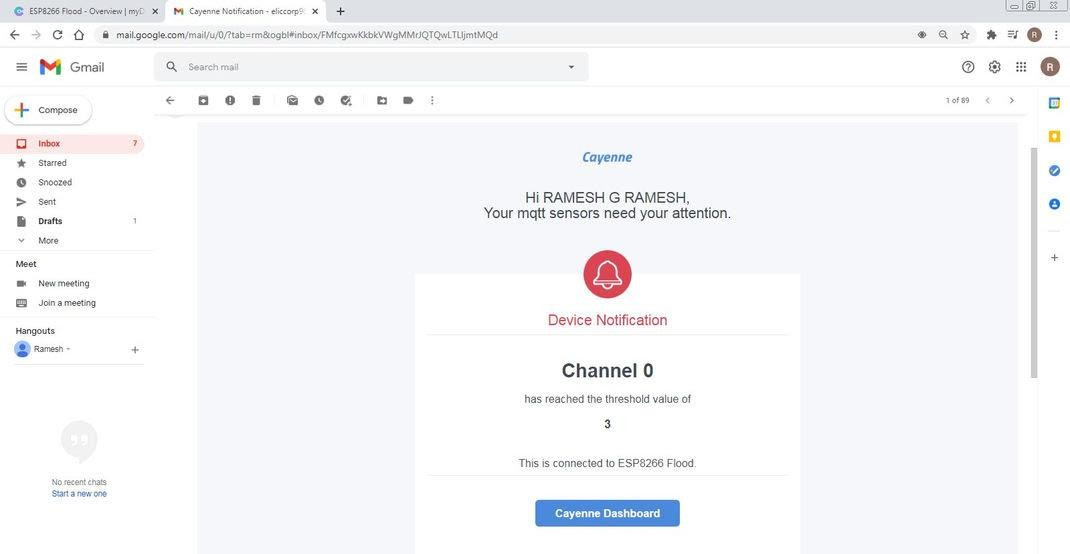
Installing the ESP8266\_Arduino\_Library

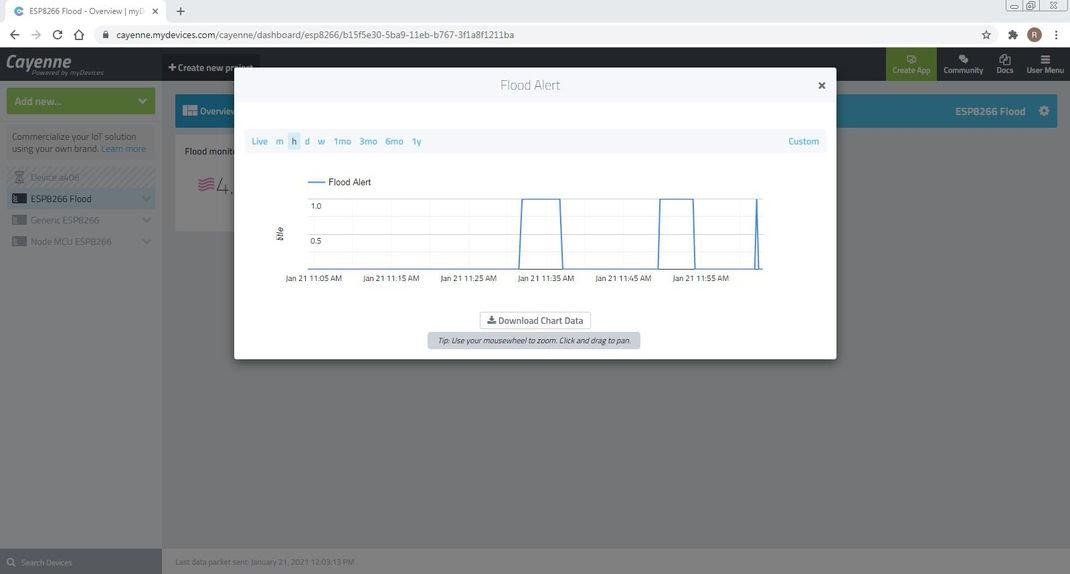
To get readings from the Ultrasonic HC-SR04 sensor module you need to have the next library installed:

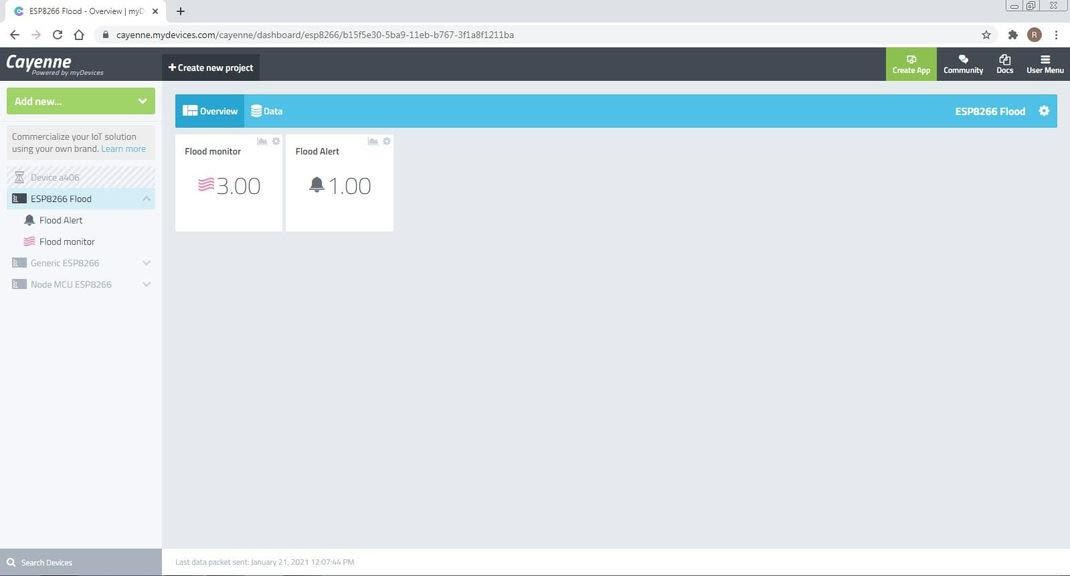
Download Ultrasonic Library

Download the Cayenne-MQTT-ESP-master library from this link.

Click on Add ZIP Library and add Cayenne-MQTT-ESP-master zip file, or directly copy the folder (Cayenne-MQTT-ESP-master) and paste it in Libraries folder of Arduino IDE.





●

When a connection is made, sensor data gets uploaded to Cayenne. Distance and digital output LED state data on Cayenne.

You can get a graphical representation of flood monitor data by clicking on the Graph icon.

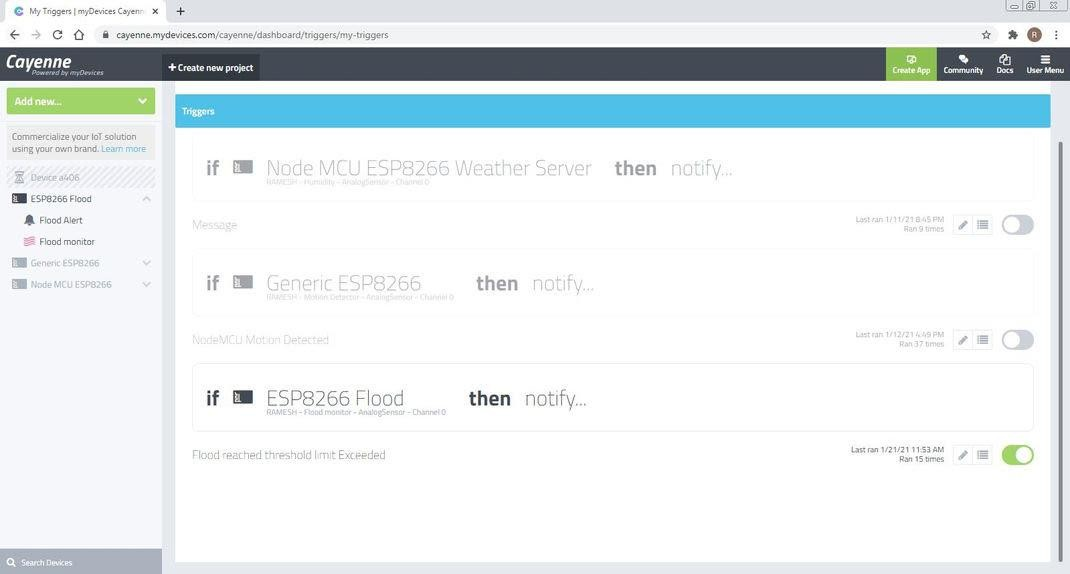
You can set Trigger alert using if and then condition, and add Mobile number and email for receiving alert message.

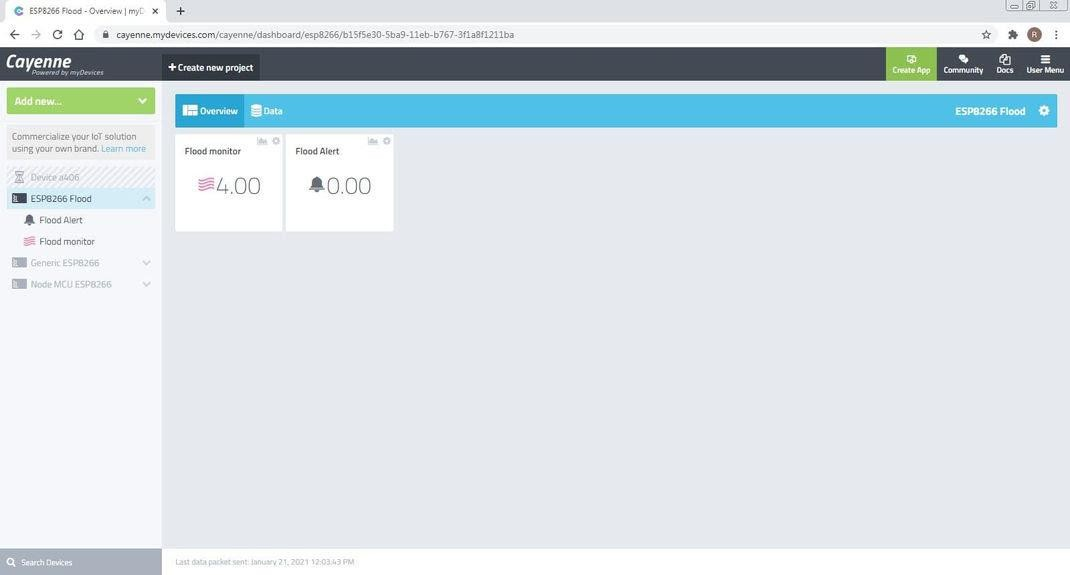
You can set Trigger alert ON/OFF

If Once sensor data gets Reached to Threshold then digital output LED state data to Cayenne. then Flood alert notify value changed to 1.00.

You can get a graphical representation of flood Alert data by clicking on the Graph icon.

And also Trigger send alert message to Email.





## Building an early warning platform involves several steps and considerations. Here's a high-level outline of how you can continue developing your project:

## 1.Project Scope and Goals:

## Clearly define the scope and objectives of your early warning platform. What specific events or risks do you want to monitor and provide warnings for?

## 2.Data Collection:

## Identify sources of relevant data. This could include weather data, social media feeds, sensors, news sources, or any other data streams that are relevant to the events you want to monitor.

## 3.Data Processing:

## Develop data processing pipelines to clean, format, and prepare the data for analysis. This may involve data normalization, filtering, and the use of machine learning or statistical techniques to identify patterns.

## 4.Risk Assessment:

## Implement a risk assessment framework to assign severity levels to potential events. This could involve defining risk criteria and thresholds for triggering warnings.

## 5.Model Development:

## Build predictive models that use historical and real-time data to forecast potential events. Machine learning techniques like regression, time series analysis, and natural language processing can be valuable tools.

## 6.Early Warning System:

## Develop an early warning system that integrates the models and risk assessment framework. When the system detects a potential event that meets or exceeds predefined thresholds, it should trigger alerts.

## 7.Alerting Mechanisms:

## Implement alerting mechanisms such as emails, SMS, mobile app notifications, or even automated voice calls to inform relevant stakeholders about potential risks.

## 8.User Interface:

## Create a user-friendly interface for stakeholders to access the early warning system. This could include dashboards and reports with real-time updates on risk levels and recommendations.

## 9.Testing and Validation:

## Rigorously test the system by using historical data to validate its effectiveness. Continuously refine the models and risk assessment criteria to improve accuracy.

## 10.Integration with Other Systems:

## Consider how the early warning platform can integrate with existing systems and processes. This may involve APIs, webhooks, or custom connectors.

## 11.Security and Privacy:

## Ensure that the platform adheres to data security and privacy regulations. Implement encryption, access controls, and data anonymization as needed.

## 12.Scalability and Redundancy:

## Plan for scalability to handle increased data volume and redundancy to ensure the system's availability even during technical failures.

## 13.Maintenance and Updates:

## Develop a maintenance plan to keep the platform up-to-date, including regular model retraining, software updates, and security patches.

## 14.User Training:

## Train relevant stakeholders on how to interpret and respond to alerts generated by the early warning platform.

## 15.Feedback Loop:

## Establish a feedback mechanism to gather input from users and use it to improve the system continuously.

## 16.Regulatory Compliance:

## Ensure that the platform complies with any industry-specific or regional regulations related to early warning systems.

## 17.Documentation:

## Create comprehensive documentation for the system's architecture, data sources, models, and processes.

## 18.Emergency Response Planning:

## Work with relevant authorities and organizations to ensure a coordinated response plan in the event of a warning being issued.

## Creating a platform to display real-time water level data and flood warnings involves web development technologies such as HTML, CSS, JavaScript, and potentially integrating with data sources and APIs that provide the necessary information. Here's a step-by-step guide to help you get started:

## ****1.Project Setup:****

## Set up a development environment with a text editor, web server, and a database (if needed).

## 2.HTML Structure:

## Create the basic HTML structure for your platform. Start with an HTML document that includes the necessary elements for your user interface. For example:

## <!DOCTYPE html>

## <html>

## <head>

## <title>Real-Time Water Level and Flood Warnings</title>

## <!-- Include CSS and JavaScript files here -->

## </head>

## <body>

## <header>

## <h1>Water Level and Flood Warnings</h1>

## </header>

## <main>

## <!-- Display real-time data here -->

## </main>

## <footer>

## <p>&copy; Your Company 2023</p>

## </footer>

## </body>

## </html>

### ****CSS Styling****:

### Use CSS to style your platform, making it visually appealing and user-friendly. You may want to create a responsive design for different screen sizes. Include your CSS file in the <head> section of your HTML document.

### ****JavaScript for Real-Time Data****:

### Use JavaScript to fetch and display real-time water level data and flood warnings. To do this, you'll need to interact with a data source or API that provides this information. You can use JavaScript's fetch method to make API requests and update your web page dynamically.

**fetch**('https://api.example.com/water-level')

**.then**(response => response.json())

**.then**(data => {

})

**.catch**(error => {

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

});

**5. Integration with Data Sources:**

Depending on the data sources you plan to use, you may need to work with APIs provided by meteorological agencies, environmental organizations, or government bodies. Ensure you have the necessary credentials and permissions to access and use this data.

**6.Displaying Data:**

Create sections on your web page to display relevant information. You might have a map showing areas with high water levels, numerical values, and textual warnings. Use HTML elements like `<div>`, `<p>`, and `<canvas>` as needed.

**7.Real Time Updates:**

Implement periodic data fetching to keep the displayed information up-to-date. You can use JavaScript's `setInterval` function to periodically fetch and update the data.

**8.Flood Warnings and Alerts:**

Create a system to generate flood warnings and alerts based on the received data. You can use JavaScript to compare the water level data with predefined thresholds and trigger alerts accordingly.

**9. User Interaction:**

Consider how users can interact with your platform, such as toggling between different data views or accessing additional information. Implement user-friendly controls and navigation.

**10. Testing:**

Test your platform with both real and simulated data to ensure it functions correctly and provides accurate warnings.

**11. Deployment:**

Choose a web hosting service to deploy your platform and make it accessible to the public. Ensure your chosen hosting service supports the technologies and APIs you are using.

**12. Documentation and Help:**

Provide documentation or tooltips to help users understand the data, warnings, and how to interpret them. Include a "Contact Us" section for support and feedback.

**13. Scalability and Performance:**

Optimize your platform for scalability and performance, especially if you expect a high volume of users or data.

**14. Accessibility and Compliance:**

Ensure that your platform complies with web accessibility standards and relevant regulations.

**15. Continuous Maintenance:**

Regularly maintain and update your platform, keeping it in sync with changing data sources and user needs.

**Designing a platform to receive and display water level data from IoT sensors and issue flood warnings involves a combination of hardware, software, and data processing components. Below is a high-level architecture for such a system:**

**1. IoT Sensors:**

- Install IoT water level sensors at strategic locations near water bodies, such as rivers or lakes. These sensors will measure water levels at regular intervals and transmit the data to a central server using wireless communication protocols like LoRa, Wi-Fi, or cellular networks.

**2. Data Transmission:**

- IoT sensors transmit data to a central gateway device, which aggregates and forwards the data to a cloud server. The gateway ensures data integrity and reliability.

**3. Cloud Server:**

- Use a cloud server or cloud computing platform to collect, store, and process the incoming sensor data. Popular options include AWS, Azure, or Google Cloud.

**4. Data Processing and Analysis:**

- Implement data processing and analysis services on the cloud server to interpret the incoming water level data. Analyze the data to detect anomalies, such as rapid increases in water levels.

**5. Flood Warning System:**

- When an anomaly or potential flood risk is detected, trigger the flood warning system. The warning system can be designed to issue different levels of alerts based on the severity of the situation.

**6. User Interface:**

- Create a web-based user interface for visualizing real-time water level data and flood warnings. Use HTML, CSS, and JavaScript to design a user-friendly dashboard. Include the following elements:

**-Map Display:** Show the location of IoT sensors on a map, along with water level data and color-coded risk zones.

**- Data Visualization:** Use charts or graphs to display historical and real-time water level data for each sensor.

**- Alerts and Warnings:** Display flood warnings, alerts, and notifications clearly on the interface.

**- User Controls:** Provide user-friendly controls for zooming, panning, and filtering data. Include the ability to set alert thresholds or customize notifications.

**7. Real-time Updates:**

- Ensure that the user interface is updated in real-time as new data arrives from the sensors. Use technologies like WebSockets to enable real-time data updates.

**8. Mobile App (Optional):**

- Develop a mobile application for users to receive flood warnings on their smartphones. This app can use push notifications to alert users in real-time.

**9. SMS/Email Alerts (Optional):**

- Implement an automated system to send SMS or email alerts to registered users when critical flood thresholds are met.

**10. Backup Power and Redundancy:**

- Ensure that the IoT sensors, gateways, and cloud servers have backup power sources and redundancy to maintain operation during power outages or technical failures.

**11. Data Security and Privacy:**

- Implement robust security measures to protect data privacy and integrity. Use encryption for data in transit and at rest.

**12. Continuous Monitoring and Maintenance:**

- Regularly monitor the system's performance, sensor health, and data accuracy. Implement automatic alerts for sensor malfunctions.

**13. Collaboration with Authorities:**

- Collaborate with local authorities and emergency services to ensure that the flood warnings issued by the system are in line with official guidelines.

**14. Compliance and Regulations:**

- Ensure that the platform complies with relevant data privacy regulations and standards for early warning systems.

**15. Documentation and User Training:**

- Provide clear documentation for users and system administrators. Offer training to relevant personnel on system operation and response procedures.

**Conclusion:**

**1. Data Sources:**

IoT sensors play a vital role in collecting real-time water level data. Proper placement and maintenance of these sensors are critical for accurate measurements.

**2. Cloud Computing:**

Leveraging cloud servers for data collection, storage, and analysis is a scalable and cost-effective solution. Major cloud providers offer tools and services that facilitate data processing.

**3. Data Processing and Analysis:**

Implement data processing and analysis to detect anomalies in water level data. Algorithms and rules can trigger flood warnings when necessary.

**4. User Interface:**

A user-friendly web-based interface, coupled with real-time data visualization, is essential for communicating flood risk to users effectively.

**5. Alerting Mechanisms:**

Develop alerting systems that notify users through web interfaces, mobile apps, SMS, and email when flood warnings are issued.