FLOOD MONITORING AND EARLY WARNING:

Phase 4: Development Part 2

Front-End Web Development:

Create the user interface with HTML, CSS, and JavaScript, ensuring it's responsive and user-friendly.
Develop the main dashboard where real-time water level data and flood warnings will be displayed.
Implement interactive charts or maps to visualize water level data, using libraries like Chart.js, D3.js, Leaflet, or Mapbox.

Real-Time Data Integration:

Set up WebSocket or server-sent events (SSE) to enable real-time data updates on the platform.
Establish a connection to the back-end server to receive live data from IoT sensors.

Back-End Development:

Enhance the back-end server developed in the previous phase.
Create RESTful APIs or GraphQL endpoints to provide real-time data to the front-end.
Implement data processing and validation to ensure data accuracy and integrity.

Flood Warning System:

Extend the flood warning system logic to detect threshold breaches or unusual patterns in the incoming data.
Develop the algorithm to trigger flood warnings when certain conditions are met.
Integrate notification mechanisms (e.g., email, SMS, push notifications) for alerting users in case of a flood warning.

User Authentication and Authorization:

Implement user registration, login, and user management features.
Set up user roles and permissions, allowing different levels of access based on user type (e.g., admin, regular user).
Ensure that flood warnings are only accessible to authorized users.

Database Management:

If not done in a previous phase, set up a database (e.g., MySQL, PostgreSQL) to store historical water level data.
Create tables to store sensor data, user information, and flood warning history.
Implement database optimization to handle a large volume of data.

User Notifications:

Design a user-friendly notification system to display flood warnings on the dashboard and send notifications to users.
Develop a notification queue to manage the timing and delivery of warnings.

Testing and Quality Assurance:

Thoroughly test the entire platform, including real-time updates, flood warning triggers, and user interactions.
Perform load testing to ensure the system can handle a high volume of simultaneous users and data updates.

Deployment and Scalability:

Prepare the platform for deployment on a web server or cloud infrastructure (e.g., AWS, Azure, or Google Cloud).
Set up automated deployment pipelines for easy updates and maintenance.
Implement horizontal scaling to ensure the platform can handle increasing data loads.

Documentation and Training:

Create user and administrator documentation to guide users on how to use the platform.
Provide training sessions for administrators and users on platform usage and maintenance.

Security and Compliance:

Implement security measures to protect user data and ensure secure communication between the platform and IoT sensors.
Ensure compliance with relevant data privacy regulations (e.g., GDPR) if applicable.

Maintenance and Support:

Plan for ongoing maintenance and support to fix issues, release updates, and improve the platform over time.

index.html (Front-End):

```
<!DOCTYPE html>
<html>
<head>
  <title>Flood Warning System</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <header>
    <h1>Flood Warning System</h1>
  </header>
  <main>
    <section id="data-visualization">
      <h2>Water Level Data</h2>
      <div id="chart-container">
```

```
            <!-- Real-time chart will be displayed here -->
          </div>
        </section>
        <section id="flood-warnings">
          <h2>Flood Warnings</h2>
          <ul id="warnings-list">
            <!-- Flood warnings will be displayed here -->
          </ul>
        </section>
    </main>
    <script src="script.js"></script>
</body>
</html>
```

styles.css (CSS for Styling):

```
body {
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

header {
    background-color: #0077FF;
    color: #FFF;
    text-align: center;
    padding: 10px;
}

main {
    max-width: 800px;
    margin: 0 auto;
    padding: 20px;
}

section {
    margin-bottom: 20px;
}

h2 {
    border-bottom: 1px solid #333;
    padding-bottom: 10px;
    margin-bottom: 10px;
}

ul {
    list-style: none;
    padding: 0;
}

li {
    padding: 10px;
    background-color: #F2F2F2;
    border: 1px solid #DDD;
```

```
    margin-bottom: 5px;
}
```

script.js (JavaScript for Real-time Data Updates and Flood Warnings):

```
// Simulated real-time data updates and flood warnings
function updateWaterLevel() {
    // Replace this with actual data from IoT sensors
    const waterLevel = Math.random() * 100; // Simulated water level data

    // Update the chart with new data
    const chartContainer = document.getElementById("chart-container");
    chartContainer.textContent = `Water Level: ${waterLevel.toFixed(2)} meters`;
}

function issueFloodWarning() {
    // Simulated flood warning
    const warningList = document.getElementById("warnings-list");
    const warningItem = document.createElement("li");
    warningItem.textContent = "Flood Warning: High water level detected!";
    warningList.appendChild(warningItem);
}

// Simulate real-time updates
setInterval(updateWaterLevel, 5000); // Update every 5 seconds

// Simulate flood warning trigger
setTimeout(issueFloodWarning, 15000); // Issue a warning after 15 seconds
```