**IOT\_PHASE 04**

**SMART WATER FOUNTAINS**

**Register no:**610821106009

**Name:** G BALAJI.

Certainly, developing a real-time water fountain status platform involves a series of steps. Here’s a high-level overview of what you need to do:

1. **Set Up the Environment:**

- Choose a code editor or IDE for web development.

- Ensure you have a server environment for running your web application (e.g., Apache, Nginx).

**2. Frontend Development:**

- Create an HTML file for the structure of your platform.

- Design the layout using CSS to make it visually appealing.

- Use JavaScript for real-time data updates.

**3. Real-Time Data Integration:**

- For real-time data, consider using technologies like WebSockets or Server-Sent Events (SSE).

- Set up a backend server to collect and push water fountain data to the platform.

**4. Displaying Water Fountain Data:**

- Create elements on your web page to display information such as water flow rate and malfunction alerts.

- Use JavaScript to update these elements with real-time data.

**5. Malfunction Alerts:**

- Implement an alert system that triggers when a malfunction is detected. You can use JavaScript for this and display a prominent message or notification.

**6. User Interface:**

- Ensure the platform has an intuitive user interface that is easy to understand and navigate.

- Consider using charts or graphs to visualize data trends over time.

**7. Testing**:

- Thoroughly test the platform to ensure it accurately displays real-time data and alerts.

**8. Security**:

- Implement security measures to protect data transmission and user access.

***9.* Documentation**:

- Document your code and system architecture for future reference and maintenance.

**10. Deployment:**

- Deploy your platform to a web server or cloud hosting service for public or private access.

Remember that this is a simplified overview, and the actual development process may require more specific details and considerations. Additionally, you may need to use libraries or frameworks, depending on your preferences and project requirements.

**Program:**

1. HTML (index.html):

Html

<!DOCTYPE html>

<html lang=”en”>

<head>

<meta charset=”UTF-8”>

<meta name=”viewport” content=”width=device-width, initial-scale=1.0”>

<link rel=”stylesheet” href=”styles.css”>

<title>Water Fountain Status</title>

</head>

<body>

<h1>Water Fountain Status</h1>

<div id=”flow-rate”>Flow Rate: Loading…</div>

<div id=”alerts”>Malfunction Alerts: None</div>

<script src=”script.js”></script>

</body>

</html>

2**.Css (styles.css):**

Ccs

Body {

Font-family: Arial, sans-serif;

Text-align: center;

Background-color: #f0f0f0;

}

H1 {

Color: #333;

}

Div {

Margin: 20px;

Padding: 10px;

Background-color: #fff;

Border: 1px solid #ccc;

}

3.**JavaScript (script.js):**

```javascript

// Simulated real-time data

Function generateRandomFlowRate() {

Return (Math.random() \* 10).toFixed(2); // Generates a random flow rate between 0 and 10 L/min

}

Function simulateMalfunction() {

Return Math.random() < 0.1; // Simulate a malfunction with a 10% chance

}

Function updateData() {

Const flowRateElement = document.getElementById(‘flow-rate’);

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

Const flowRate = generateRandomFlowRate();

Const hasMalfunction = simulateMalfunction();

flowRateElement.textContent = `Flow Rate: ${flowRate} L/min`;

if (hasMalfunction) {

alertsElement.textContent = ‘Malfunction Alerts: Yes’;

alertsElement.style.color = ‘red’;

} else {

alertsElement.textContent = ‘Malfunction Alerts: None’;

alertsElement.style.color = ‘green’;

}

}

// Update data every 5 seconds

setInterval(updateData, 5000);

// Initial data update

updateData();