IOT_PHASE-4

Continuing from where we left off, in Part 1, we created the basic structure of a real-time water fountain status platform using HTML, CSS, and JavaScript. Now, let's further enhance the platform by adding interactivity and dynamic updates to improve the user experience.

Add User Interactivity:

To allow users to interact with the platform, you can add features like a manual refresh button and the ability to acknowledge and clear malfunction alerts.

html

```
<!-- Add refresh button -->

<button id="refreshButton">Refresh</button>



<!-- Modify malfunction alerts section to include a "Clear" button -->

<div class="status">

    <h2>Malfunction Alerts:</h2>

    <ul id="alerts">

        <li>Loading...</li>

    </ul>

    <button id="clearAlerts">Clear Alerts</button>

</div>
```

Now, update the JavaScript code (script.js) to handle these interactions:

javascript

```
const refreshButton = document.getElementById("refreshButton");

const clearAlertsButton = document.getElementById("clearAlerts");



// Function to update water fountain status

function updateWaterFountainStatus() {

    // Simulated data (replace with actual data source)

    const flowRate = (Math.random() * 10).toFixed(2); // Random flow rate

    const alerts = ["Low water level", "High pressure"];
    // Update the flow rate display

    document.getElementById("flowRate").textContent = flowRate + " liters/minute";



    // Update malfunction alerts

    const alertsList = document.getElementById("alerts");

    alertsList.innerHTML = "";

    alerts.forEach((alert) => {

        const listItem = document.createElement("li");

        listItem.textContent = alert;

        alertsList.appendChild(listItem);

    });

}

a

// Function to clear malfunction alerts

function clearAlerts() {

    const alertsList = document.getElementById("alerts");

    alertsList.innerHTML = "";

}



// Event listeners for buttons

refreshButton.addEventListener("click", updateWaterFountainStatus);
```

```
clearAlertsButton.addEventListener("click", clearAlerts);



// Update data every 5 seconds (adjust as needed)

setInterval(updateWaterFountainStatus, 5000);



// Initial update

updateWaterFountainStatus();
```

Now, users can manually refresh the data with the "Refresh" button, and they can clear malfunction alerts using the "Clear Alerts" button.

Real Data Integration:

To make this platform functional with real data, you will need to replace the simulated data with actual data from your water fountain sensors or an API. This typically involves making AJAX requests to your data source and updating the display with the retrieved data.

Here's an example of how you can use the `fetch` API to retrieve data from a JSON endpoint (replace the URL with your actual data source):

javascript

```javascript
function updateWaterFountainStatus() {

    fetch("https://your-data-source-url.com")

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

        .then((data) => {

            const flowRate = data.flowRate; // Replace with your data fields

            const alerts = data.alerts; // Replace with your data fields
            // Update the display with real data
            document.getElementById("flowRate").textContent = flowRate + " liters/minute";


            const alertsList = document.getElementById("alerts");
            alertsList.innerHTML = "";
            alerts.forEach((alert) => {
                const listItem = document.createElement("li");
                listItem.textContent = alert;
                alertsList.appendChild(listItem);
            });
        })
        .catch((error) => {
            console.error("Error fetching data:", error);
        });
}


// Continue with the same setup for button interactions, etc.
```

Creating a real-time water fountain status platform using web development technologies like HTML, CSS, and JavaScript involves a few key steps. For this example, I'll assume that you have a data source (sensors or an API) that provides real-time water fountain data. Here's a simple implementation using HTML, CSS, and JavaScript:

HTML Structure:

Create the basic HTML structure for the platform. We'll have a title, a section to display water flow rate, and another section for malfunction alerts.

html

```
<!DOCTYPE html>
<html>
<head>
    <title>Water Fountain Status</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <h1>Real-Time Water Fountain Status</h1>
    <div class="status">
        <h2>Water Flow Rate:</h2>
        <p id="flowRate">Loading...</p>
    </div>
    <div class="status">
        <h2>Malfunction Alerts:</h2>
        <ul id="alerts">
            <li>Loading...</li>
        </ul>
    </div>
    <script src="script.js"></script>
</body>
</html>
```

CSS Styling (styles.css):

Style the HTML elements for a clean, user-friendly interface.

css

```
body {
    font-family: Arial, sans-serif;
    text-align: center;
}
```

```css
h1 {
    color: #333;
}

.status {
    border: 1px solid #ccc;
    padding: 15px;
    margin: 10px;
    border-radius: 5px;
    display: inline-block;
}

#alerts {
    list-style: none;
    padding: 0;
}
```

JavaScript (script.js):

Use JavaScript to fetch and display real-time data. In this example, I'll simulate data updates using a timer. Replace this with actual data from your sensors or API.

javascript

```javascript
function updateWaterFountainStatus() {
    // Simulated data (replace with actual data source)
    const flowRate = Math.random() * 10; // Random flow rate
    const alerts = ["Low water level", "High pressure"];

    // Update the flow rate display
    document.getElementById("flowRate").textContent = flowRate.toFixed(2) + "
liters/minute";

    // Update malfunction alerts
    const alertsList = document.getElementById("alerts");
    alertsList.innerHTML = "";
    alerts.forEach((alert) => {
        const listItem = document.createElement("li");
        listItem.textContent = alert;
        alertsList.appendChild(listItem);
    });
}

// Update data every 5 seconds (adjust as needed)
setInterval(updateWaterFountainStatus, 5000);
updateWaterFountainStatus(); // Initial update
```