SMART WATER FOUNTAIN

PHASE 4

To create a real-time water fountain status platform, you'll need to use a combination of HTML for structure, CSS for styling, and JavaScript for interactivity. Additionally, you'll likely need a backend server to handle real-time data updates. Here's a basic outline to get you started:

Step 1: Set Up the Project

1. Create a new directory for your project.

2. Inside this directory, create the following files:

- index.html: for the main structure of the platform.
- style.css: for styling the platform.
- script.js: for handling interactivity and real-time updates.
- server.js: to simulate a backend server (for testing purposes).

Step 2: Design the HTML Structure

In your `index.html` file, set up the basic structure:

HTML:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Water Fountain Status</title>
  <link rel="stylesheet" href="style.css">
</head>
<body>
  <div class="container">
    <h1>Water Fountain Status</h1>
    <div id="flowRate">Flow Rate: <span id="flowValue">Loading...</span></div>
```

```
    <div id="malfunction">Malfunction Alert: <span id="malfunctionValue">No</span></div>
  </div>
 <script src="script.js"></script>
</body>
</html>
```

Step 3: Style the Platform

In your `style.css` file, you can add some basic styles. Customize these as per your design preferences:

Css:

```
body {
  font-family: Arial, sans-serif;
  text-align: center;
}
.container {
  margin: 50px auto;
  max-width: 600px;
  padding: 20px;
  border: 1px solid #ccc;
  border-radius: 10px;
  background-color: #f9f9f9;
}

h1 {
  margin-bottom: 20px;
}

#flowRate, #malfunction {
  font-size: 1.2em;
  margin-bottom: 10px;
}
#malfunctionValue {
```

```
    color: #FF0000; /* Red for alert */
}
```

Step 4: Implement Real-Time Updates

In your `script.js` file, use JavaScript to simulate real-time updates. For the sake of this example, we'll use a timer to update the values:

JAVASCRIPT:

```
function updateFlowRate() {
    const flowValue = Math.random() * 10; // Simulated flow rate (replace with actual data)
    document.getElementById('flowValue').textContent = flowValue.toFixed(2);
}
function updateMalfunctionAlert() {
    const malfunctionValue = Math.random() > 0.8; // Simulated malfunction (replace with actual data)
    document.getElementById('malfunctionValue').textContent = malfunctionValue ? 'Yes' : 'No';
}
setInterval(() => {
    updateFlowRate();
    updateMalfunctionAlert();
}, 5000); // Update every 5 seconds (adjust as needed)
```

Step 5: Set Up a Backend Server (Optional)

If you have a real backend that provides data, you can replace the simulated data with actual API calls in `script.js`.

In `server.js`, you would set up routes to handle data requests from your front end.

You can now test your platform locally by opening `index.html` in a web browser. If everything works as expected, you can deploy it to a web server for public access.

this is a basic example and you may need to adapt it based on your specific requirements and the technologies you're using for real-time data retrieval.