SMART WATER FOUNDATION [PHASE 4]

Creating a real-time platform for water fountain data requires a more comprehensive approach, involving backend development, database integration, and potentially a cloud service for handling real-time data. Here's a high-level structure that outlines the essential components for such a platform:

1.Backend Server:

Utilize a backend server (Node.js, Django, Flask, etc.) to handle data management, storage, and communication with the frontend. Implement endpoints for data retrieval and storage.

2. Database Integration:

Set up a database (such as MongoDB, MySQL, or PostgreSQL) to store real-time and historical data, including water flow rates, alerts, and other relevant information.

3. Real-Time Communication:

Use technologies such as WebSockets or Socket.IO to establish a real-time communication channel between the server and the frontend, enabling the display of live data updates.

4. Frontend Development:

Create a user-friendly interface for visualizing the water fountain data. Use HTML, CSS, and JavaScript, and potentially a frontend framework like React, Angular, or Vue.js to build an interactive dashboard.

5. Data Visualization:

Incorporate data visualization libraries such as Chart.js, D3.js, or Plotly.js to represent the water flow rate data in a meaningful and intuitive way.

6. Malfunction Alerts:

Implement a notification system that triggers alerts when certain predefined conditions (e.g., abnormal flow rates, system malfunctions) are met. This can be achieved using email notifications, in-app alerts, or SMS notifications.

7. Security Measures:

Implement appropriate security measures to protect the data and ensure secure communication between the frontend and backend components. This may include data encryption, user authentication, and authorization.

8. Deployment and Scaling:

Deploy the platform on a reliable cloud service such as AWS, Google Cloud, or Microsoft Azure, considering the scalability and performance requirements of handling real-time data.

This outline provides a comprehensive approach to building a robust platform for real-time water fountain data management and visualization. The specifics of the implementation would depend on the particular requirements and constraints of the project.

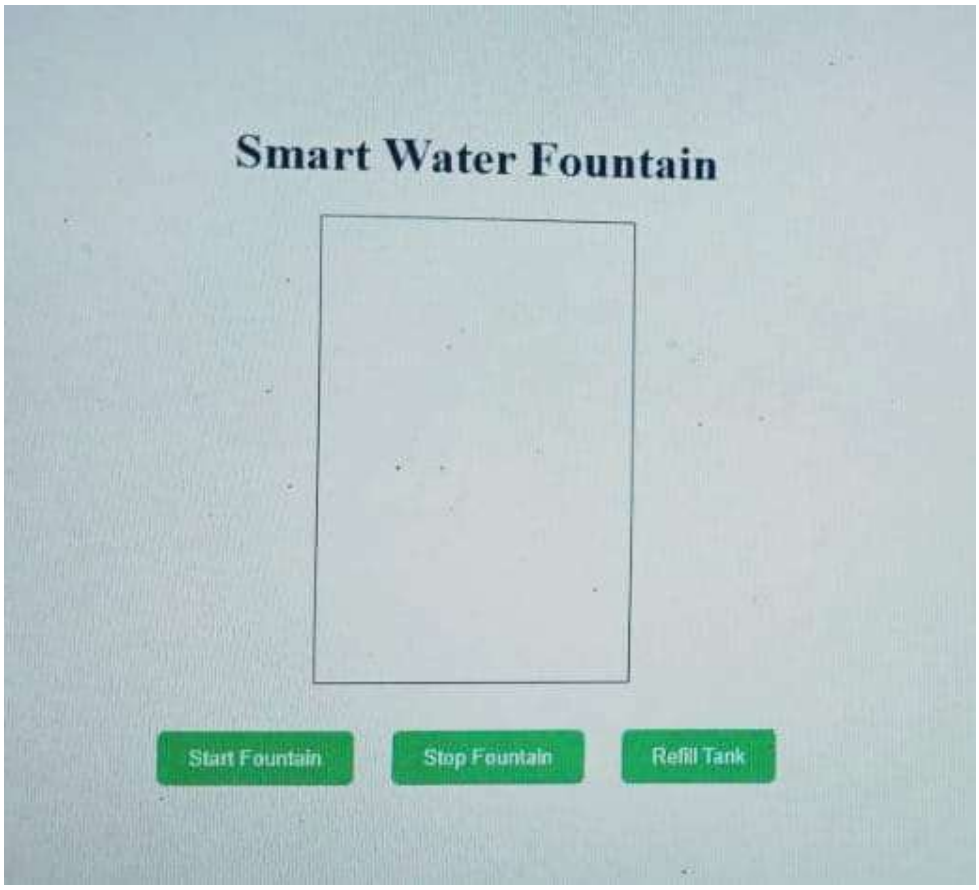
```
<!DOCTYPE html>
<html>

<head>
  <title>Smart Water Fountain</title>
  <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
  <style>
    body {
      font-family: Arial, sans-serif;
      display: flex;
      justify-content: center;
```

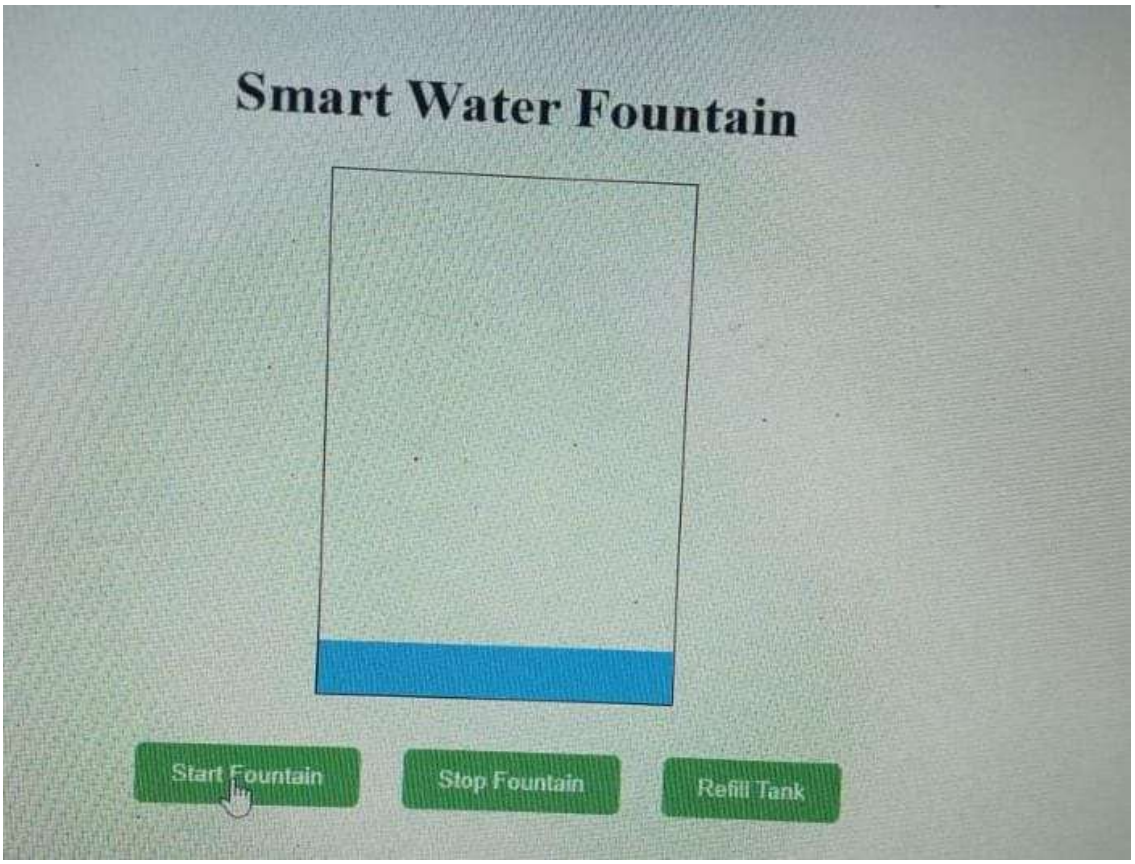
```
      align-items: center;
      height: 100vh;
      margin: 0;
      background-color: #f2f2f2;
    }

    .container {
      text-align: center;
    }

    .header {
      font-size: 36px;
      margin-bottom: 20px;
    }

    .button {
      display: inline-block;
      padding: 10px 20px;
      margin: 10px;
      font-size: 18px;
      border: none;
      border-radius: 5px;
      cursor: pointer;
cursor: pointer;
      color: #fff;
    }

    #waterLevelChart {
      margin-top: 20px;
```

```
    }
  </style>
</head>

<body>
  <div class="container">
    <h1 class="header">Smart Water Fountain</h1>
    <div id="tank" style="text-align: center;">
      <!-- Include your tank visualization here -->
    </div>
    <div style="text-align: center;">
      <button id="startButton" class="button">Start Fountain</button>
      <button id="stopButton" class="button">Stop Fountain</button>
    </div>
    <canvas id="waterLevelChart" width="400" height="200"></canvas>
  </div>

  <script>
    const waterLevelData = []; // Simulated water level data
    const ctx = document.getElementById('waterLevelChart').getContext('2d');
    const myChart = new Chart(ctx, {
      type: 'line',
      data: {
        labels: ['Time-1', 'Time-2', 'Time-3', 'Time-4', 'Time-5'],
        datasets: [{
          label: 'Water Level',
          data: waterLevelData,
          backgroundColor: 'rgba(75, 192, 192, 0.2)',
          borderColor: 'rgba(75, 192, 192, 1)',
```

```
borderWidth: 1
        }]
      },
      options: {
        scales: {
          y: {
            beginAtZero: true
          }
        }
      }
    });

    // Simulate start and stop fountain actions
    function startFountain() {
      waterLevelData.push(0.7); // Simulated data for demonstration
      myChart.update();
    }

    function stopFountain() {
      if (waterLevelData[waterLevelData.length - 1] > 0.9) {
        alert("Water is Overflowing!");
      } else if (waterLevelData[waterLevelData.length - 1] < 0.1) {
        alert("Water is Empty! Refilling...");
        waterLevelData.push(0.5); // Simulated refill data for demonstration
      } else {
        waterLevelData.push(0.3); // Simulated data for demonstration
      }
      myChart.update();
}
```

```
    function stopFountain() {
      if (waterLevelData[waterLevelData.length - 1] > 0.9) {
        alert("Water is Overflowing!");
      } else if (waterLevelData[waterLevelData.length - 1] < 0.1) {
        alert("Water is Empty! Refilling...");
        waterLevelData.push(0.5); // Simulated refill data for demonstration
      } else {
        waterLevelData.push(0.3); // Simulated data for demonstration
      }
      myChart.update();
    }
  </script>
</body> </html/>
```





This outline provides a comprehensive approach to building a robust platform for real-time water fountain data management and visualization. The specifics of the implementation would depend on the particular requirements and constraints of the project.