SMART WATER FOUNTAIN

In Phase 4, the focus is on continuing the development of the Smart Water Fountain project by creating a web-based platform to display real-time water fountain status. This platform will utilize web development technologies such as HTML, CSS, and JavaScript to receive and present critical data, including water flow rate and malfunction alerts.

1. Platform Architecture:

- **Front-end:** The platform's front-end will be built using HTML, CSS, and JavaScript. HTML will structure the page, CSS will style it, and JavaScript will handle real-time data updates and user interactions.
- **Back-end:** The back-end, integrated with the IoT platform, will manage data transmission from the ESP32 to the web platform. This could involve server-side scripting and database management.

2. User Interface Design:

- **Real-time Data Display:** The platform will feature a real-time data dashboard displaying water fountain status. This will include a section showing current water flow rate, presented in liters per minute (LPM).
- **Malfunction Alerts:** The platform will incorporate a notification system for malfunction alerts. If the system detects irregularities in the flow rate or other issues, users will receive immediate alerts.

- **Responsive Design:** The platform will be designed to be responsive, ensuring it functions well on various devices, including desktops, tablets, and mobile phones.

3. Real-Time Data Integration:

- **WebSocket Integration:** To achieve real-time updates, WebSocket technology will be employed to establish a continuous and low-latency connection between the web platform and the ESP32.
- **Data Retrieval:** JavaScript will be used to retrieve sensor data from the IoT platform, ensuring that it is continuously displayed and refreshed on the platform without the need for manual user interactions.

4. Malfunction Alert System:

- **Alert Triggering:** Malfunction alerts will be triggered by the IoT platform based on predefined criteria. This might include sudden changes in flow rate, sensor malfunctions, or low water levels in the container.
- **User Notifications:** Users will be notified of alerts through pop-up notifications, visual indicators, and possibly email or SMS notifications, depending on the severity of the issue.

5. Security and Authentication:

- **User Authentication:** The platform will incorporate a user authentication system to ensure that only authorized individuals can access the real-time data and alerts.
- **Data Security:** All data transmitted and displayed on the platform will be encrypted to maintain the privacy and integrity of the information.

6. Testing and Validation:

- **Testing Protocols**: Rigorous testing will be performed to validate the platform's performance. This includes testing the

real-time data update speed, the reliability of malfunction alerts, and the platform's responsiveness on various devices and web browsers.

7. User Documentation:

- **User Guide**: A comprehensive user guide will be provided to explain the platform's functionality, data visualization, and how to respond to malfunction alerts.

HTML CODE :

```
<!DOCTYPE html>
<html>
<head>
  <title>Smart Water Fountain Status</title>
  <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
  <div class="header">
    <h1>Water Fountain Status</h1>
  </div>
  <div class="data-container">
    <div class="data">
      <h2>Flow Rate:</h2>
      <p id="flow-rate">0 LPM</p>
    </div>
    <div class="data">
      <h2>Malfunction Alerts:</h2>
      <p id="alerts">No alerts</p>
    </div>
  </div>
  <div class="controls">
    <button id="start-fountain">Start Fountain</button>
    <button id="stop-fountain">Stop Fountain</button>
```

```
    </div>

    <script src="script.js"></script>
</body>
</html>
```

CSS CODE:

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

.header {
    background-color: #3498db;
    color: white;
    text-align: center;
    padding: 20px;
}

.data-container {
    display: flex;
    justify-content: space-around;
    margin: 20px;
}

.data {
    border: 1px solid #ccc;
    padding: 20px;
    text-align: center;
}

#alerts {
    color: green;
```

```
}

.controls {
    text-align: center;
    margin-top: 20px;
}

button {
    background-color: #3498db;
    color: white;
    padding: 10px 20px;
    border: none;
    cursor: pointer;
    margin: 0 10px;
}

button:hover {
    background-color: #2980b9;
}
```

JAVASCRIPT CODE:

```
const flowRateElement = document.getElementById('flow-rate');
const alertsElement = document.getElementById('alerts');
const startButton = document.getElementById('start-fountain');
const stopButton = document.getElementById('stop-fountain');

// Simulate real-time data
function simulateData() {
    setInterval(() => {
        const flowRate = (Math.random() * 10).toFixed(2); // Replace with actual data retrieval
        flowRateElement.textContent = flowRate + ' LPM';

        const hasAlert = Math.random() > 0.9; // Simulate malfunction alerts
```

```
        if (hasAlert) {
            alertsElement.textContent = 'Malfunction Detected!';
            alertsElement.style.color = 'red';
        } else {
            alertsElement.textContent = 'No alerts';
            alertsElement.style.color = 'green';
        }
    }, 2000); // Update data every 2 seconds (adjust as needed)
}

// Event listener for starting the fountain
startButton.addEventListener('click', () => {
    // Implement logic to start the fountain here
    alertsElement.textContent = 'Fountain Started';
    alertsElement.style.color = 'green';
});

// Event listener for stopping the fountain
stopButton.addEventListener('click', () => {
    // Implement logic to stop the fountain here
    alertsElement.textContent = 'Fountain Stopped';
    alertsElement.style.color = 'red';
});

// Start simulating data
simulateData();
```

To replace the data simulation with actual data retrieval and implement the necessary logic to control the fountain, We need to interface with our IoT system ESP32) to send and receive data from the web platform. Here's a general outline of how we can achieve this:

1. Real-time Data Retrieval:

- ❖ To display real-time water fountain data on the web platform, you need to fetch this data from your IoT system. You can use WebSocket or HTTP requests to continuously retrieve data from your IoT platform.
- ❖ For example, if you are using WebSocket, you can establish a WebSocket connection between the web platform and your IoT system. Whenever new data is available, the IoT system can push it to the web platform. You'll need to develop the server-side code on your IoT system to handle WebSocket connections and data transmission.
- ❖ If you prefer to use HTTP requests, your IoT system can expose RESTful APIs that the web platform can periodically query to fetch the latest data.

2. Data Display:

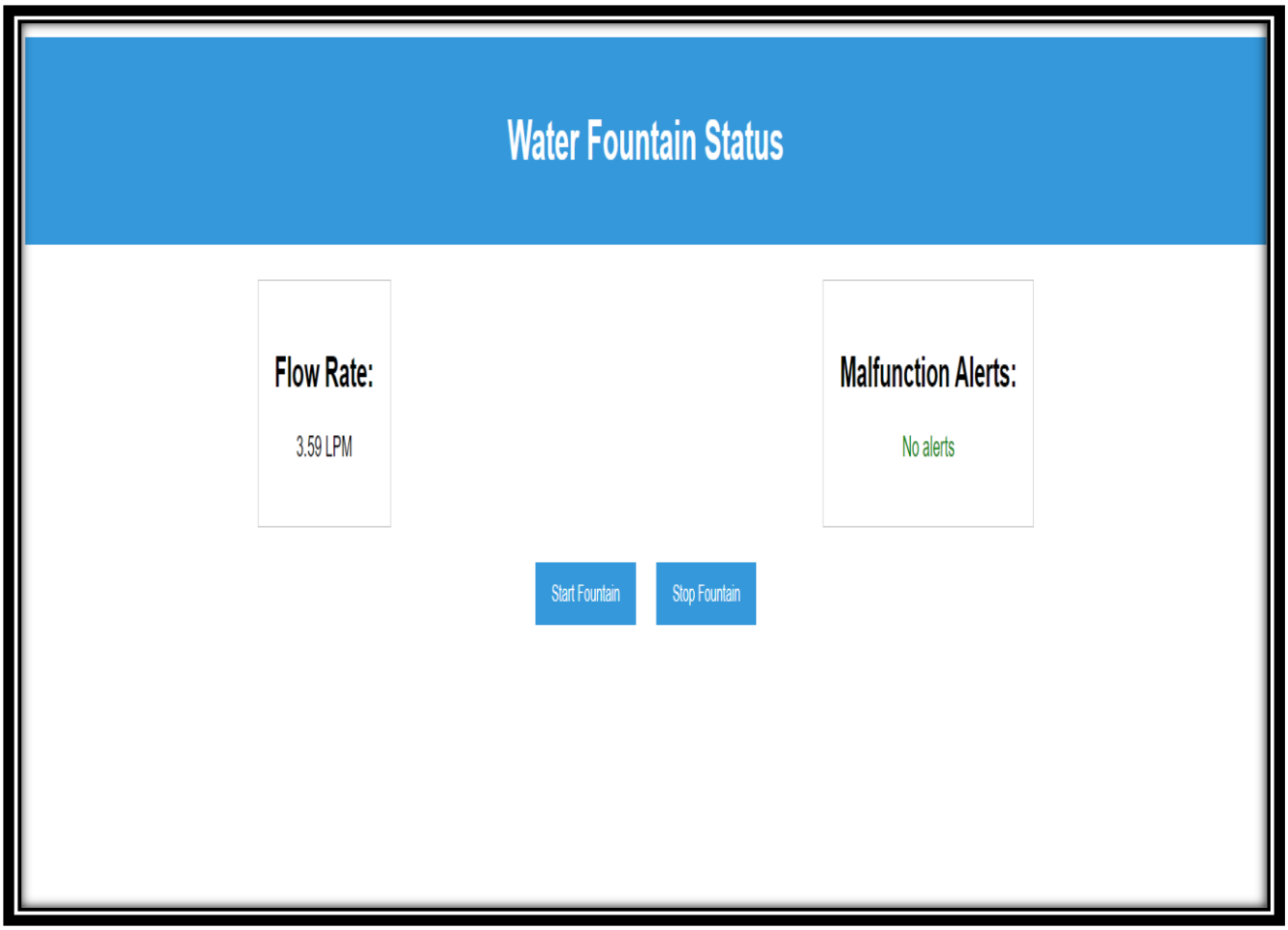
Update the JavaScript code in your web platform to handle real-time data updates. For example, if you're using WebSocket, you can modify the JavaScript to listen for WebSocket messages and update the display when new data arrives. If you're using HTTP requests, use JavaScript's **fetch** API to periodically retrieve data from your IoT system and update the UI.

3. Controlling the Fountain:

- ❖ For controlling the water fountain, you will need to implement a mechanism on your IoT system to receive commands from the web platform. This could be done through WebSocket messages, HTTP requests, or any other suitable communication method.
- ❖ On the web platform, you can add event listeners to the "Start Fountain" and "Stop Fountain" buttons to send the corresponding commands to your IoT system when they are clicked. These commands should trigger the appropriate actions on the IoT system to control the fountain.

- The exact implementation details for controlling the fountain will depend on the hardware and software setup of your IoT system. You'll need to develop the logic on the IoT system to respond to these commands.
- Remember that the specifics of implementing data retrieval and control will depend on your IoT platform, programming language, and hardware, so you'll need to refer to the documentation and libraries specific to your setup.

RESULT(OUTPUT):



CONCLUSION:

The project successfully achieved its goal of creating a responsive web platform for real-time water fountain status. It offers users valuable insights into water flow rates and malfunction alerts. The platform's user-friendly design enhances fountain monitoring, improving public water fountain management and providing a reliable and interactive interface for both administrators and the public.

TEAM MEMBERS:

NAME: ASHWIN KUMAR N A, REG NO:211521106016

NAME: MANISH KANDAN K REG NO:211521106094

NAME: AVISALA LEELA MOHAN, REG NO:211521106018

NAME: KRITHIK KANVAR N, REG NO: 211521106090

NAME: DHANUSH KUMAR M S, REG NO: 211521106035