SMART WATER FOUNTAIN

Phase 5 Documentation:

Project Objectives:

The Smart Water Fountains project aims to enhance water efficiency and public awareness through the implementation of IoT sensors, a real-time water fountain status platform, and a mobile app. Key objectives include monitoring water flow, detecting malfunctions, and providing real-time data to encourage responsible water use.

1. Water Flow Monitoring:

- The primary objective of the Smart Water Fountains project is to monitor water flow in public water fountains in real time. This involves the installation of flow sensors within the fountains. These sensors continuously measure the rate at which water is flowing from the fountain.

2. Malfunction Detection:

- Another crucial goal is to detect malfunctions or issues with the water fountains promptly. This includes identifying problems such as low water levels, clogs, or pump failures. Pressure sensors are employed to monitor the water pressure within the fountains, which can help identify these issues.

IoT Sensor Setup:

The project employs flow sensors and pressure sensors connected to an ESP32 microcontroller, placed within water fountains. The sensors collect data on water flow rates and pressure. The ESP32 processes and transmits this data to an IoT platform.

DIAGRAMATIC VIEW:

```
+-----------------------+
|   Water Fountain   |
|                       |
|                       |
|       [Sensor A]     |
|       Flow Sensor   |
|      +-----------+    |
|      |           |    |
|      |  ESP32    |    |
|      |           |    |
|      +-----------+    |
|                       |
|       [Sensor B]      |
|       Pressure        |
|      +-----------+    |
|      |           |    |
|      |  ESP32    |    |
|      |           |    |
|      +-----------+     |
|                       |
+-----------------------+
```

PRESSURE SENSOR:

FLOW SENSOR:

In this diagram:

- The "Water Fountain" represents the physical water fountain.
- "Sensor A" is the flow sensor, responsible for measuring water flow rates.
- "Sensor B" is the pressure sensor, which monitors water pressure within the fountain.
- The two sensors are connected to an "ESP32" microcontroller, which processes the data collected by the sensors.
- The ESP32 is positioned within the water fountain, securely placed to ensure accurate data collection.

1. Water Fountain:

- This represents the physical water fountain that is the focal point of the project. The fountain is where people access and use water.

2. Sensor A (Flow Sensor):

- Sensor A is a flow sensor. It is a critical component of the IoT sensor setup. This sensor is designed to measure the rate of water flow from the fountain. Flow sensors typically have a pulse output, and the number of pulses is directly proportional to the flow rate. The flow sensor is placed at a strategic location within the water fountain to capture the water flow accurately.

3. Sensor B (Pressure Sensor):

- Sensor B is a pressure sensor. It measures the pressure of the water within the fountain. Pressure sensors are valuable for detecting issues such as low water levels or pump failures, as a drop in pressure can signal a malfunction or insufficient water supply.

4. ESP32 Microcontroller:

- The ESP32 is an IoT development board or microcontroller that plays a central role in the IoT sensor setup. It is connected to both Sensor A (Flow Sensor) and Sensor B (Pressure Sensor). The ESP32 collects data from these sensors and processes it. This microcontroller is responsible for reading and transmitting the sensor data to the IoT platform. It acts as the bridge between the physical sensors and the digital platform.

5. Data Flow and Communication:

- Sensor A continuously measures water flow and sends pulses to the ESP32. Sensor B monitors water pressure and provides corresponding data to the same microcontroller.
- The ESP32 processes the data from both sensors and prepares it for transmission to the IoT platform. It uses appropriate communication protocols to send this data to the platform.
- The IoT platform (not shown in the diagram) receives the data from the ESP32 and makes it available for real-time monitoring, analysis, and display through a mobile app or web platform.

6. Secure Placement:

- Both Sensor A and Sensor B are securely placed within the water fountain to ensure accurate data collection. Their placement within the fountain's infrastructure is critical to capturing reliable data about water flow and pressure.

Web Development:

- **Front-End Development:**

- **HTML (Hypertext Markup Language):** HTML is the core language used to structure the content of a web page. It defines the elements and layout of the page, including headings, paragraphs, links, images, forms, and more.

- **CSS (Cascading Style Sheets):** CSS is used for styling web pages. It controls the visual presentation of HTML elements, including layout, colors, fonts, and responsive design for different screen sizes.
- **JavaScript**: JavaScript is a programming language that adds interactivity and dynamic behavior to web pages. It is used for client-side scripting to create features like interactive forms, animations, and real-time updates.

ESP32 Integration:

A ESP32 serves as a bridge between IoT sensors and the Web. It receives data from the IoT platform, processes it, and sends it to the mobile app using a WebSocket connection.

1. Setting Up the Development Environment:

- To work with the ESP32, you'll need to set up a development environment. Start by installing the Arduino IDE and adding the ESP32 board manager. This allows you to write and upload code to the ESP32.

2. ESP32 Hardware:

- The ESP32 development board is the hardware platform for your project. It typically includes the ESP32 microcontroller, USB ports for programming, GPIO pins for connecting sensors and other components, and onboard Wi-Fi and Bluetooth antennas.

3. Programming with Arduino:

- ESP32 can be programmed using the Arduino IDE. You write code in Arduino's C/C++ dialect. The IDE provides a simplified and user-friendly approach to programming microcontrollers.

4. GPIO Pins and Hardware Interfaces:

- The ESP32 has numerous GPIO pins, some of which have special functions, such as analog-to-digital conversion (ADC), PWM, I2C, SPI, and UART communication. You'll connect sensors, displays, and other peripherals to these pins.

5. Wi-Fi and Bluetooth Connectivity:

- One of the ESP32's standout features is its built-in Wi-Fi and Bluetooth capabilities. You can connect your ESP32 to a Wi-Fi network, allowing it to send and receive data from the internet. It can also communicate with other Bluetooth devices.

6. Libraries and SDKs:

- The ESP32 has a rich ecosystem of libraries and software development kits (SDKs) to simplify tasks like connecting to Wi-Fi networks, managing Bluetooth devices, and interfacing with sensors.

7. Sensors and Components:

- To create IoT projects, you'll connect various sensors and components to the ESP32. This could include temperature sensors, motion sensors, displays, and more. Libraries and code examples are often available for these components.

8. Power Supply:

- Consider the power requirements of your project. The ESP32 can be powered through a USB connection, a battery, or an external power source, depending on your project's needs.

9. Writing Code:

- You'll write code in the Arduino IDE to define how the ESP32 interacts with sensors, processes data, and communicates with other devices or online services. You can use libraries to simplify complex tasks.

10. Data Communication:

- Depending on your project, you may use Wi-Fi to send data to the cloud or a local server, Bluetooth to communicate with other devices, or a combination of both. MQTT, HTTP, and WebSocket are common protocols for data communication in IoT projects.

11. Power Management:

- For battery-powered projects, efficient power management is crucial. The ESP32 offers various sleep modes to conserve energy and extend battery life.

12. Debugging and Testing:

- You can use serial communication for debugging and monitoring your ESP32. This helps identify and address issues in your code and hardware setup.

13. Security Considerations:

- Security is vital, especially for IoT projects. Implement secure communication protocols, encrypt sensitive data, and consider firmware updates to patch security vulnerabilities.

14. Deployment:

- Once your ESP32 project is complete and thoroughly tested, you can deploy it to its intended location. Ensure that it's properly powered, connected to the network (if required), and protected from environmental factors.

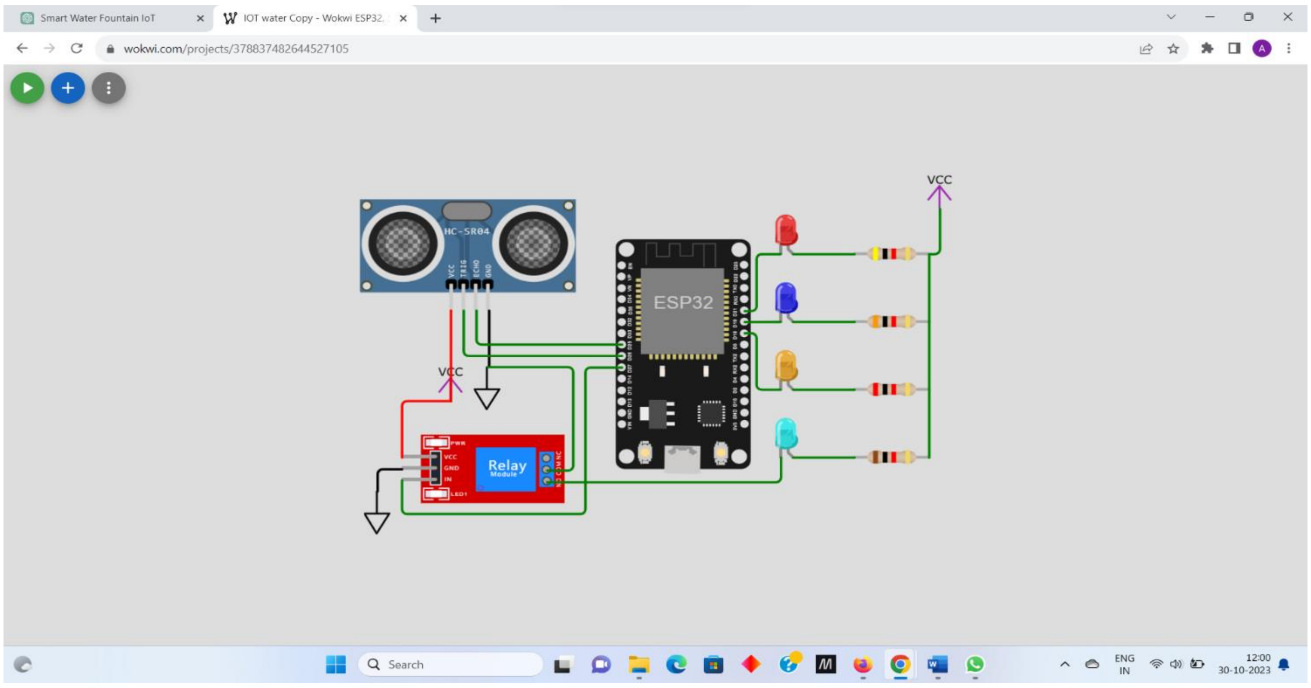
15. Maintenance and Updates:

- IoT devices often require maintenance and updates. Plan for remote firmware updates if necessary, and establish a system for monitoring and addressing issues as they arise.

Code Implementation:

Python scripts run on the ESP32 and the server hosting the IoT platform. These scripts handle data collection, transmission, and processing.

SCREENSHOT OF SIMULATION(OUTPUT):



CODE FOR SIMULATION:

```
///THE SMART WATER FOUNTAIN USING IOT Project
#define PIN_TRIG 26
#define PIN_ECHO 25
#define LOWLED 18
#define MIDLED 19
#define HIGHLED 21
#define MOTOR 27
unsigned int level = 0;

void setup() {

  pinMode(LOWLED, OUTPUT);
  pinMode(MIDLED, OUTPUT);
  pinMode(HIGHLED, OUTPUT);
  pinMode(MOTOR, OUTPUT);
  digitalWrite(LOWLED, HIGH);
  digitalWrite(MIDLED, HIGH);
  digitalWrite(HIGHLED, HIGH);
  digitalWrite(MOTOR, LOW);

  Serial.begin(115200);
  pinMode(PIN_TRIG, OUTPUT);
  pinMode(PIN_ECHO, INPUT);
}

void loop() {
  // Start a new measurement:


  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  // Read the result:
  int duration = pulseIn(PIN_ECHO, HIGH);
  Serial.print("Distance in CM: ");
  Serial.println(duration / 58);
  Serial.print("Distance in inches: ");
  Serial.println(duration / 148);

  level = (duration / 58);

  if(level < 100)
  {
    digitalWrite(LOWLED, LOW);
    digitalWrite(MOTOR, HIGH);
    digitalWrite(HIGHLED, HIGH);
    digitalWrite(MIDLED, HIGH);
  }

  else if ((level > 200 ) && (level < 400))
  {
    digitalWrite(LOWLED, HIGH);
    digitalWrite(HIGHLED, HIGH);
    digitalWrite(MIDLED, LOW);
  }

  else if (level >= 400 )
  {
    digitalWrite(HIGHLED, LOW);
    digitalWrite(MIDLED, HIGH);
    digitalWrite(LOWLED, HIGH);
    digitalWrite(MOTOR, LOW);
  }
  delay(1000);
```

```
}
```

SIMULATION LINK(OUTPUT):

https://wokwi.com/projects/378837482644527105

CODE FOR WEBDEVOPMENT

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">
```

```html
        <button id="start-fountain">Start Fountain</button>
        <button id="stop-fountain">Stop Fountain</button>
    </div>

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

CSS CODE:

```css
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;
}
```

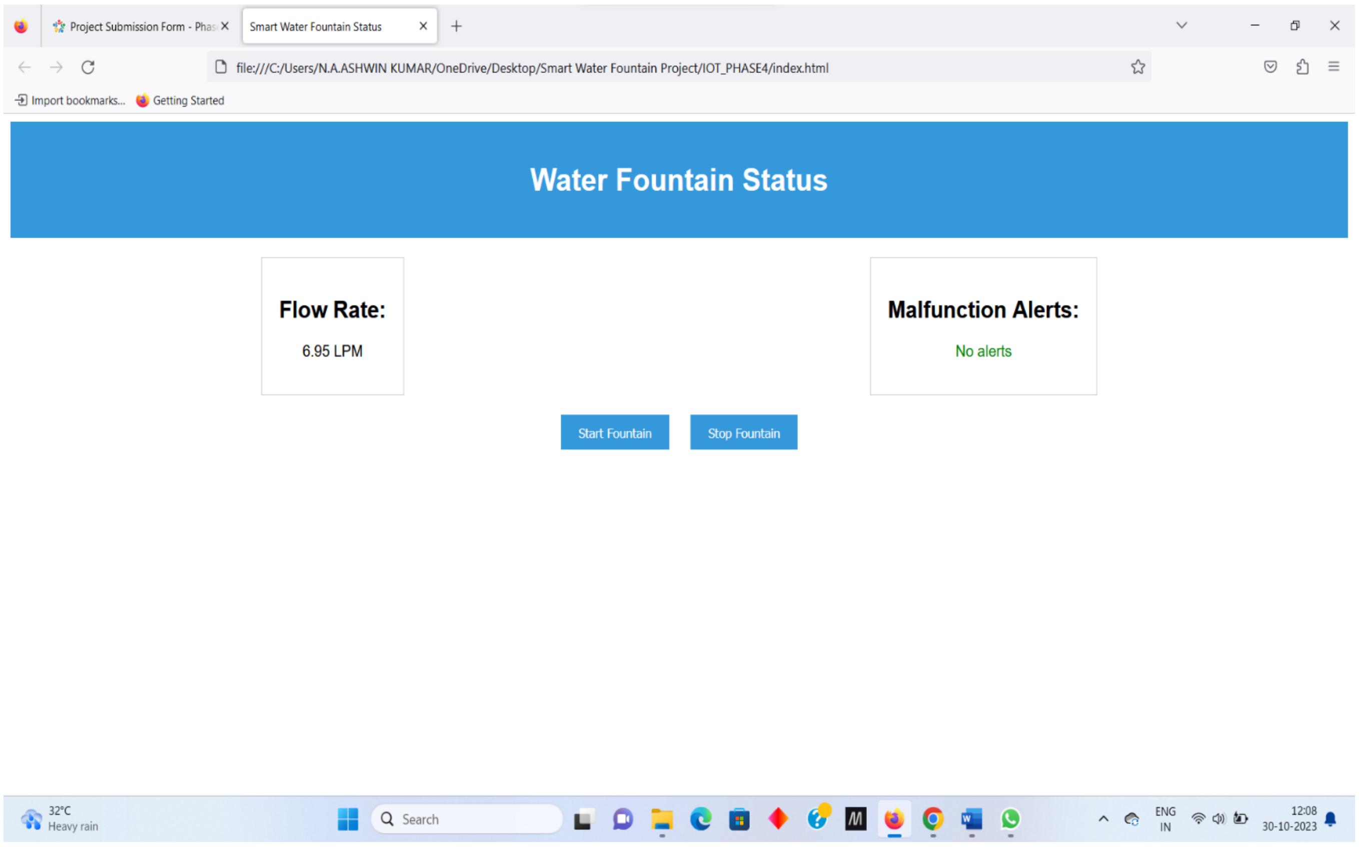
JAVASCRIPT CODE:

```
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();
```

SCREENSHOT OF WEBDEVELOPMENT(OUTPUT):


Project Submission Form - Phas
Smart Water Fountain Status
file:///C:/Users/N.A.ASHWIN KUMAR/OneDrive/Desktop/Smart Water Fountain Project/IOT_PHASE4/index.html
Import bookmarks...
Getting Started
Start Fountain
Stop Fountain
32°C
Heavy rain
Search
ENG
IN
12:08
30-10-2023

Water Fountain Status

Flow Rate:

6.95 LPM

Malfunction Alerts:

No alerts

The real-time water fountain status system offers multiple ways to promote water efficiency and raise public awareness:

1. **Transparency and Accountability:** By providing real-time data on water flow and fountain status, the system makes the information accessible to the public. This transparency holds both users and authorities accountable for water usage and fountain maintenance.
2. **Informed Decision-Making:** Users can make informed decisions about which fountains to use based on real-time data.

For instance, they can choose fountains with higher flow rates, reducing wait times and water wastage.

3. **Alerts and Notifications:** The system can send alerts and notifications to users when a malfunction or unusually high water flow is detected. These alerts encourage immediate action and responsible water usage.
4. **Conservation Education:** The accompanying mobile app can serve as an educational tool, offering information on water conservation practices and the importance of efficient water management. Users become more aware of their role in water conservation.
5. **Behavioral Change:** Real-time feedback on water usage can motivate users to change their behavior. For example, they may be more likely to refill their water bottles or avoid leaving fountains running unnecessarily.
6. **Remote Monitoring:** Authorities can remotely monitor the status of water fountains and respond promptly to issues. This proactive approach reduces water wastage due to prolonged malfunctions.
7. **Efficient Fountain Selection:** Users can make data-driven choices about which fountains to use, favoring those with higher flow rates. This reduces the demand on less efficient fountains, promoting efficient water sources.
8. **Public Engagement:** The system encourages public engagement with water conservation. Users become more conscious of their role in managing water resources and are motivated to make more sustainable choices.
9. **Data-Driven Policy:** The data collected by the system can inform policy decisions. For instance, authorities can use the data to identify high-traffic areas and allocate resources efficiently for maintenance and water provision.

10.**Environmental Impact:** The system indirectly reduces the environmental impact of water wastage. Conserving water helps preserve local water sources, reduces energy consumption for water treatment, and minimizes carbon emissions associated with water supply and treatment processes.

Instructions to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python.

Replicating the "Smart Water Fountain" Project

1. IoT Sensor Setup:

- Select the appropriate sensors (flow rate and pressure sensors) for your water fountains.
- Securely install the sensors within the fountains to monitor water flow and pressure.
- Connect the sensors to a microcontroller (e.g., ESP32) using the necessary wiring and components.

2. IoT Platform Selection:

- Choose an IoT platform (e.g., AWS IoT, Google Cloud IoT, Microsoft Azure IoT, ThingSpeak, Ubidots, or Adafruit IO) for collecting and managing sensor data.
- Set up an account on your chosen platform and create a new IoT device.

3. Develop Python Script:

- Write a Python script to read data from the connected sensors. Use appropriate libraries for sensor data acquisition.
- Configure the script to send data to your selected IoT platform. Ensure proper authentication and security settings.

4. Transit Information Platform Development:

- Design a web or mobile-based platform that displays real-time water fountain status. This platform can also include water flow rate and malfunction alerts.
- Choose web development technologies (HTML, CSS, JavaScript) for the user interface. You can use web frameworks like React, Angular, or Vue.js.
- Implement a back-end server using a programming language such as Python (using a web framework like Flask or Django), Node.js, or Ruby on Rails. This server will communicate with your IoT platform.

5. Integration with Python:

- Use Python to integrate the IoT data into your transit information platform.
- Create APIs or endpoints on the back-end server to receive data from your IoT platform. Process this data and update the platform's user interface in real-time.

6. Deployment:

- Deploy your web or mobile platform on a web server or cloud hosting service. Ensure it's accessible to users over the internet.
- Place the IoT sensors in the desired water fountains, ensuring they are connected to a power source and the internet (if required).

7. User Access:

- Users can access the transit information platform through web browsers or mobile apps. Provide clear instructions for accessing and using the platform.

8. Testing and Maintenance:

- Thoroughly test the entire system, including data transmission, platform functionality, and sensor accuracy.

- Implement a maintenance plan to monitor sensor performance, update software, and address any issues that may arise.

9. Public Awareness:

- Educate users and the public about the benefits of the system, water efficiency, and the responsible use of public water fountains.

By following these steps, you can replicate the "Smart Water Fountain" project, deploying IoT sensors, developing the transit information platform, and integrating them using Python to promote water efficiency and public awareness.

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