**Smart Water Fountain Phase-4**

Incorporating web technologies into your student IoT smart water fountain project can enhance its functionality and accessibility. You can use these technologies for data visualization, remote monitoring, and user interfaces. Here's how to do it:

**1. Web-Based User Interface:** Create a web-based user interface that allows users to monitor and control the smart water fountain from their computers or mobile devices. You can achieve this using technologies such as HTML, CSS, and JavaScript. Here are the steps:

* **HTML/CSS:** Design a user-friendly web page with buttons, status indicators, and data visualization elements. Use HTML for the structure and CSS for styling.
* **JavaScript:** Use JavaScript to add interactivity to the web page. You can send commands to the microcontroller and receive real-time sensor data using AJAX requests or WebSocket connections.
* **Responsive Design:** Ensure that your web interface is responsive, so it works well on various screen sizes and devices.

**2. Data Visualization:** Display data from your smart water fountain in a visually appealing manner. You can use charting libraries like Chart.js or D3.js to create graphs and charts that show real-time or historical data. For example, you can display water level trends, temperature fluctuations, or humidity changes over time.

**3. Remote Monitoring:** Implement remote monitoring capabilities so users can check the status of the fountain even when they are not physically present. Use web technologies to enable remote access. Here's how:

* **Web Hosting:** Host your web-based user interface on a web server, which allows users to access it from anywhere with an internet connection.
* **Secure Access:** Implement security measures like user authentication and SSL/TLS encryption to protect the remote access.
* **Real-Time Updates:** Use technologies like WebSockets to provide real-time updates on the status of the fountain.

**4. Data Storage and Cloud Integration:** Consider storing data in the cloud using services like AWS, Google Cloud, or Azure. This enables data backup, historical analysis, and easy access from anywhere.

**5. Notifications:** Use web technologies to send notifications to users. For example, you can use email alerts, SMS notifications, or push notifications through web browsers to inform users about critical events like low water levels or system failures.

**6. Mobile App Integration:** You can create a dedicated mobile app for your smart water fountain project using web technologies such as React Native or Ionic. This provides a more convenient and native-like experience for mobile users.

**7. Data Logging and Analytics:** Implement data logging and analytics to collect and analyze historical data. You can use databases and web-based analytics tools to gain insights into the performance of the water fountain over time.

**8. Documentation and Tutorials:** Make sure to document the web technologies and tools you use in your project. Create tutorials for future students or enthusiasts who want to replicate or build upon your work.

By incorporating web technologies, you can create a modern and user-friendly IoT smart water fountain project with a powerful web-based interface for control, monitoring, and data visualization. This approach not only enhances the project's functionality but also provides valuable skills in web development and IoT integration.

CODING:

// Include the necessary libraries

#include <Servo.h>

// Define the pins for the PIR sensor and water pump

const int pirPin = 2; // PIR sensor connected to digital pin 2

const int pumpPin = 8; // Water pump connected to digital pin 8

// Create a Servo object for controlling the water pump

Servo waterPump;

// Variables to store sensor state

int pirState = LOW;

int lastPirState = LOW;

void setup() {

// Initialize the PIR sensor pin as an input

pinMode(pirPin, INPUT);

// Initialize the water pump as a servo

waterPump.attach(pumpPin);

// Turn off the water pump initially

waterPump.write(0);

// Serial communication for debugging (optional)

Serial.begin(9600);

}

void loop() {

// Read the PIR sensor

pirState = digitalRead(pirPin);

// Check if motion is detected

if (pirState == HIGH) {

// Turn on the water pump

waterPump.write(90); // Adjust the angle to control the water flow

delay(5000); // Run the pump for 5 seconds (adjust as needed)

waterPump.write(0); // Turn off the water pump

delay(1000); // Delay before rechecking for motion

}

// Save the current state for the next iteration

lastPirState = pirState;

}

**Connecting mobile app with smart water fountain:**

To connect a mobile app to a smart water fountain, you'll need to create both the mobile app and the backend for controlling and monitoring the fountain. Here's a high-level overview of the steps involved:

**1. Develop the Mobile App:**

* Choose a mobile app development platform: You can develop a mobile app for iOS (using Swift) and Android (using Java or Kotlin) or use a cross-platform framework like Flutter or React Native for both platforms.
* Design the User Interface (UI): Create a user-friendly interface for controlling and monitoring the smart water fountain. Include features such as turning the fountain on/off, adjusting water flow, and checking the water level.
* Implement User Authentication: If multiple users can control the fountain, implement user authentication to ensure secure access.
* Integrate MQTT or WebSocket: Use MQTT or WebSocket for real-time communication between the mobile app and the fountain. The app should be able to send commands (e.g., "turn on" or "adjust flow") and receive updates (e.g., water level readings).
* Build the mobile app, test it on physical devices or emulators, and deploy it to app stores (Google Play and Apple App Store).

**2. Develop the Backend:**

* Create a backend server to act as an intermediary between the mobile app and the smart water fountain.
* Implement an MQTT broker or WebSocket server on the backend to facilitate communication between the app and the water fountain. You can use libraries like Eclipse Mosquitto for MQTT or libraries like Socket.IO for WebSocket.
* Set up endpoints for the mobile app to send commands and retrieve data. These endpoints will handle the communication with the water fountain.
* Implement security measures to protect the backend, such as user authentication and authorization.
* Handle device registration: When a user sets up a new smart water fountain, the mobile app should be able to register the fountain with the backend.

**3. Connect the Smart Water Fountain:**

* The smart water fountain (controlled by the Raspberry Pi or Arduino, as mentioned in the previous response) should be configured to communicate with the MQTT broker or WebSocket server running on your backend.

**4. Mobile App-Backend Communication:**

* In the mobile app, implement code to send commands to the backend server (e.g., "start water flow," "stop water flow") through MQTT or WebSocket.
* Receive updates from the fountain through MQTT or WebSocket (e.g., water level readings).

**5. Testing and Deployment:**

* Test the entire system by controlling the smart water fountain using the mobile app.
* Debug and optimize as needed.
* Once everything works correctly, deploy the backend to a server or cloud platform.

**6. Enhancements and Features:**

* Consider adding additional features such as scheduling water flow times, setting water flow patterns, or sending notifications/alerts to the mobile app when specific events occur (e.g., low water level).

By following these steps, you can create a mobile app that connects to a smart water fountain, allowing users to control and monitor the fountain remotely. The communication between the app and the fountain is facilitated through a backend server and a real-time messaging protocol like MQTT or WebSocket.

CIRCUIT DIAGRAM:



