IBM (SKILL UP ONLINE)
PROJECT 8: SMART WATER FOUNTAIN
PHASE 4:DEVELOPMENT PART-2

COURSE NAME: INTERNET OF THINGS

GROUP:5

PROJECT NUMBER: 08

TITLE: SMART WATER FOUNTAIN

PHASE 4: DEVELOPMENT PART-2

YEAR: lll

DEPARTMENT: ELECTRONICS AND COMMUNICATIONS ENGINEERING

PROJECT SUBMITTED TO: IBM (Skill Up Online)

NO OF STUDENTS: 06

NAME OF STUDENTS: 810021106050-MUHAMMED DAYYAN AL SALAAM S

:810021106052-NAVEEN RAJ S

:810021106062-RAVIVARMAN R

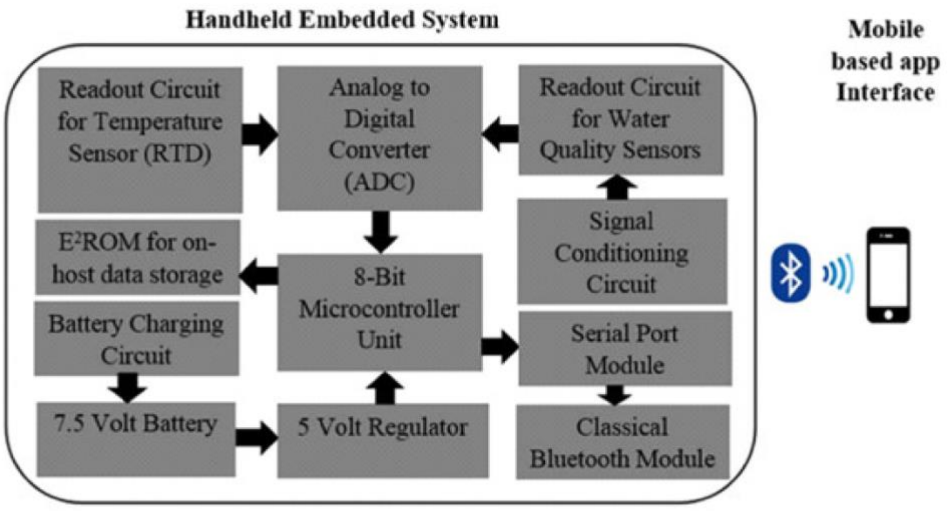
:810021106069-SARAVANAN K

:810021106304-YUVURAJ M

:810021106311-PREMKUMAR M

SMART WATER FOUNTAIN

INTRODUCTION:


Handheld Embedded System
Mobile based app Interface
Readout Circuit for Temperature Sensor (RTD)
Analog to Digital Converter (ADC)
Readout Circuit for Water Quality Sensors
E²ROM for on-host data storage
8-Bit Microcontroller Unit
Signal Conditioning Circuit
Battery Charging Circuit
Serial Port Module
7.5 Volt Battery
5 Volt Regulator
Classical Bluetooth Module

A smart water fountain is a modern and technologically advanced version of a traditional water fountain. It incorporates various features and technologies to enhance its functionality and user

experience. Here are some common features and components you might find in a smart water fountain:

1. Filtration System: Smart water fountains typically have advanced filtration systems that remove impurities, such as sediments, chemicals, and contaminants, ensuring the water is safe and clean for consumption.

2. Temperature Control: Some smart water fountains can cool or heat the water to a user-defined temperature. This is useful for people who prefer their water at a specific temperature.

3. Dispensing Mechanism: They often have touchless or sensor-activated dispensing mechanisms to minimize contact and reduce the risk of contamination.

4. Wi-Fi Connectivity: Many smart water fountains can connect to your home or office's Wi-Fi network. This connectivity enables features like remote control, monitoring, and data collection.

5. Mobile App Integration: Some models come with companion mobile apps that allow users to control and customize the fountain's settings from their smartphones. Users can set water temperature, monitor filter status, and track water consumption.

6. Usage Tracking: Smart water fountains can keep track of water usage, allowing you to monitor your daily intake. This feature can

be especially useful for staying hydrated and meeting your hydration goals.

7. Voice Control: Some smart fountains can be integrated with virtual voice assistants like Amazon Alexa or Google Assistant. This enables users to control the fountain's features through voice commands.

8. Leak Detection: Some models are equipped with sensors to detect leaks and automatically shut off the water supply to prevent water damage.

9. Water Quality Sensors: Advanced models may include sensors that monitor water quality in real-time and provide data on water purity.

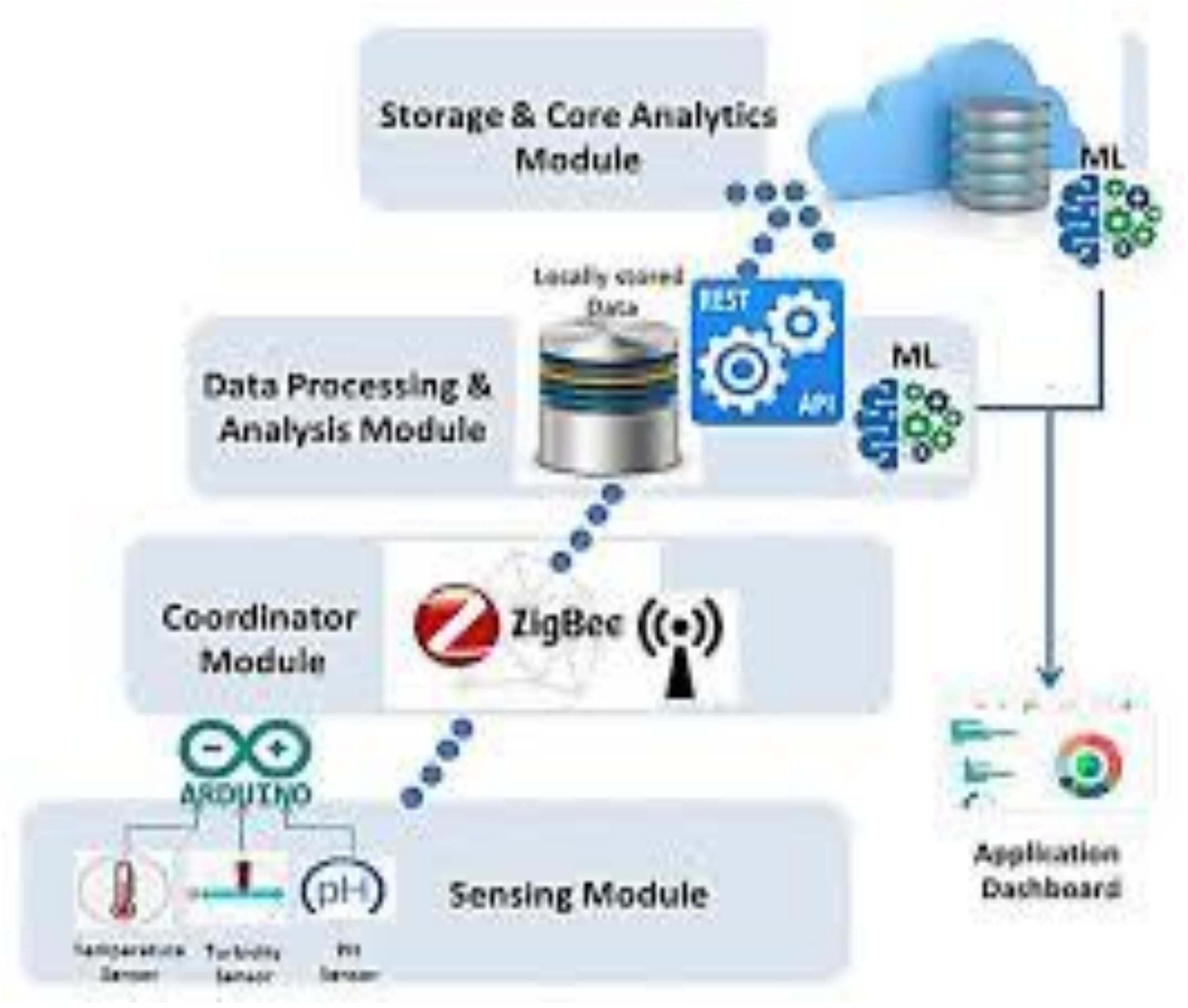
10. LED Display: They often have LED displays or indicators to show temperature, filter status, and other important information.

11. Scheduled Dispensing: Users can set schedules for water dispensing, which can be handy for automating your hydration routine.

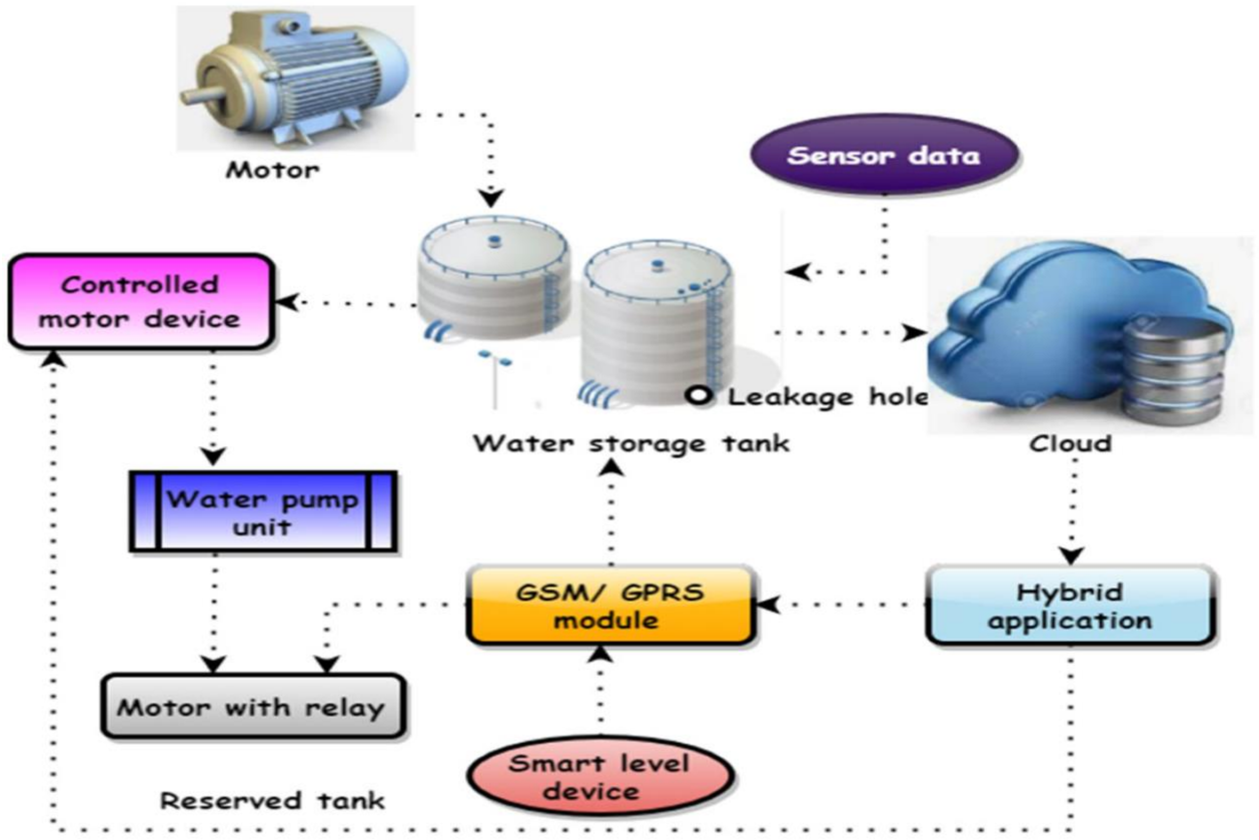
12. Energy Efficiency: Many smart fountains are designed to be energy-efficient, using power-saving modes or timers to conserve energy.

13. Integration with Other Smart Devices: Some smart water fountains can integrate with other smart home devices, allowing for automation and interaction with other systems in your home or office.

The primary goal of a smart water fountain is to provide convenient and safe access to clean drinking water while offering customizable features to meet the needs and preferences of users. These fountains can be used in various settings, including homes, offices, gyms, and public spaces.



<u>DEVELOPMENT OF SMART WATER FOUNTAIN:</u>



Developing a smart water fountain involves integrating technology to make a traditional water fountain more efficient, user-friendly, and data driven. Here are the key steps and components involved in the development of a smart water fountain:

1. Define Objectives and Features:

- Determine the primary goals of your smart water fountain. For example, you might want to conserve water, provide easy access to clean drinking water, or create an interactive art installation.

2. Hardware Components:

- Water Supply: You'll need a water source, which could be a water line or a reservoir, depending on the purpose of the fountain.

- Pump: A pump is essential for circulating water through the fountain.

- Sensors: Various sensors can be used to monitor water levels, quality, temperature, and flow rates.

- Valves: Valves can control the flow of water, enabling features like automatic refilling or water conservation.

- Filtration System: If the fountain is for drinking water, it may need a filtration system to ensure water quality.

- Power Supply: A stable power source, preferably with a backup option, is crucial.

- Microcontroller/Control Unit: This will be the brain of your smart fountain and will control various components based on sensor data and user inputs.

3. Software Development:

- User Interface: Develop a user-friendly interface, which could be a mobile app, web interface, or physical controls.

- Data Processing: The microcontroller collects and processes data from sensors to make decisions about water flow and other functionalities.

- Automation: Implement automation logic, such as turning the fountain on/off, adjusting water flow, or activating features based on user preferences and sensor data.

- Connectivity: If it's a "smart" fountain, you may need to implement Wi-Fi or Bluetooth connectivity for remote control and monitoring.

4. Data Storage and Analysis:

- Store the data collected from sensors for analysis and reporting. This can be done locally or in the clouds.
- Implement data analysis algorithms to derive insights and make informed decisions about water usage.

5. Energy Efficiency:

- Consider energy-efficient components and programming to minimize power consumption.

6. Maintenance and Alerts:

- Incorporate alerting mechanisms to notify users or administrators of any issues, like low water levels or pump malfunctions.
- Ensure that the system can be easily maintained, with replaceable components and easy access for cleaning and repairs.

7. Safety Measures:

- Implement safety features to prevent accidents, especially if it's a public water fountain.

8. Testing and Calibration:

- Thoroughly test the system to ensure it works as intended and calibrate sensors and controls for accuracy.

9. User Education:

- Provide clear instructions and user education for interacting with the smart water fountain.

10. Data Privacy and Security:

- Ensure that any data collected, especially if it's a public installation, is handled securely and in compliance with privacy regulations.

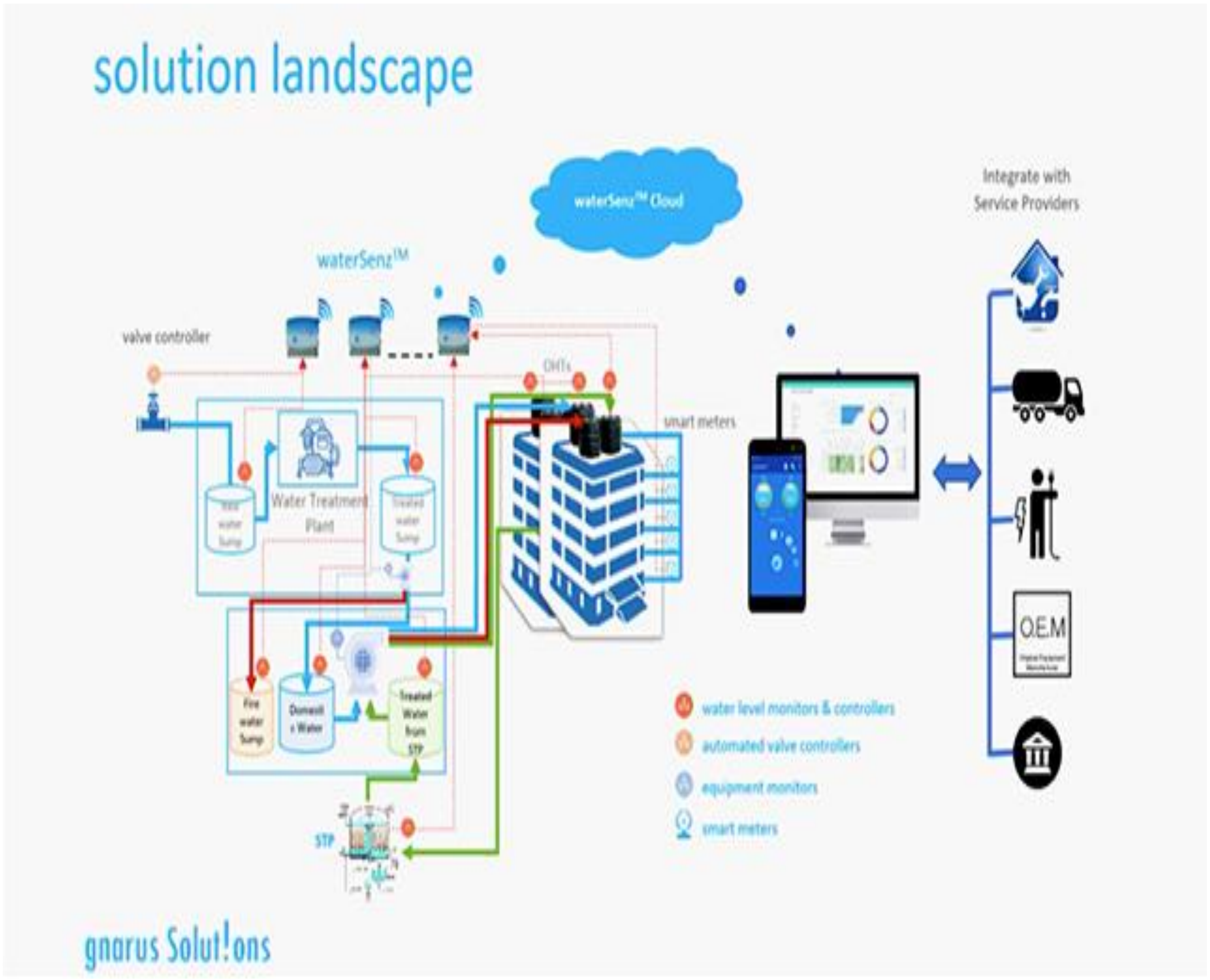
11. Deployment and Scaling:

- Deploy the smart water fountain in its intended location and consider scaling the solution if it proves successful and meets the objectives.

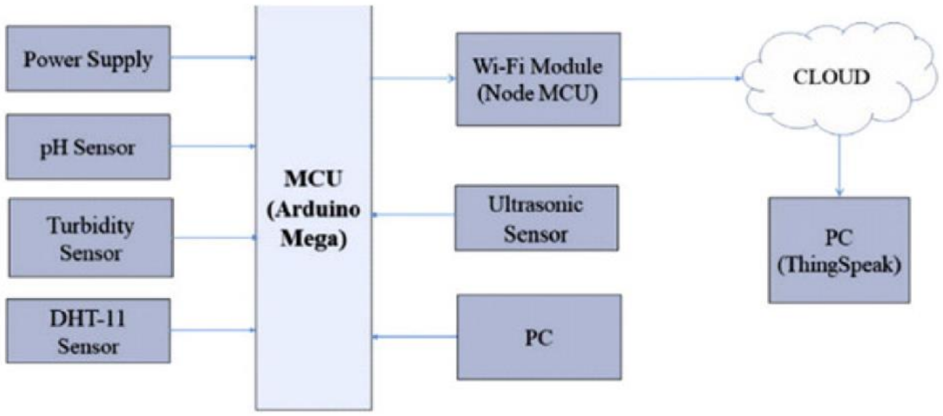
12. Monitoring and Updates:

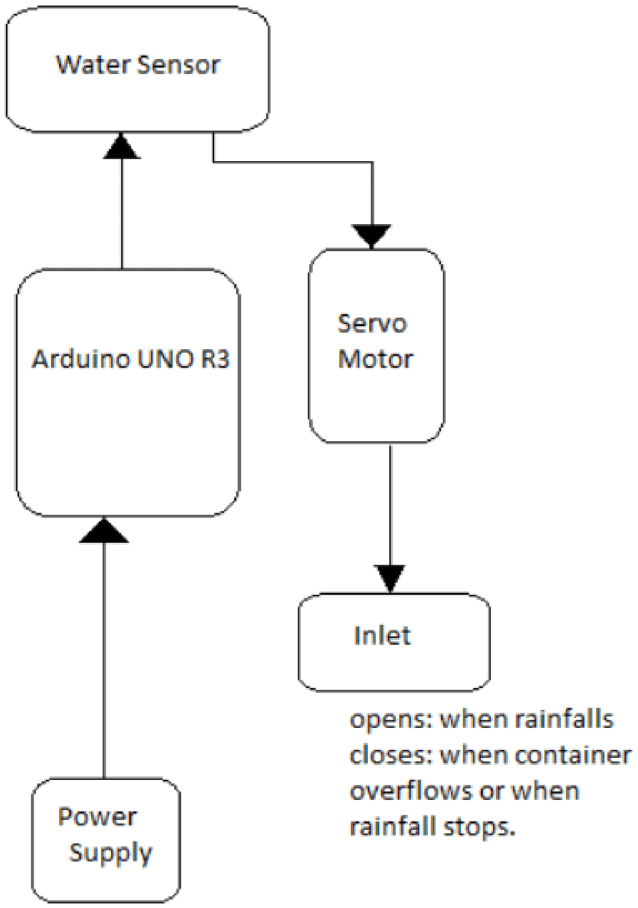
- Continuously monitor the system's performance and provide software updates to fix issues or add new features.

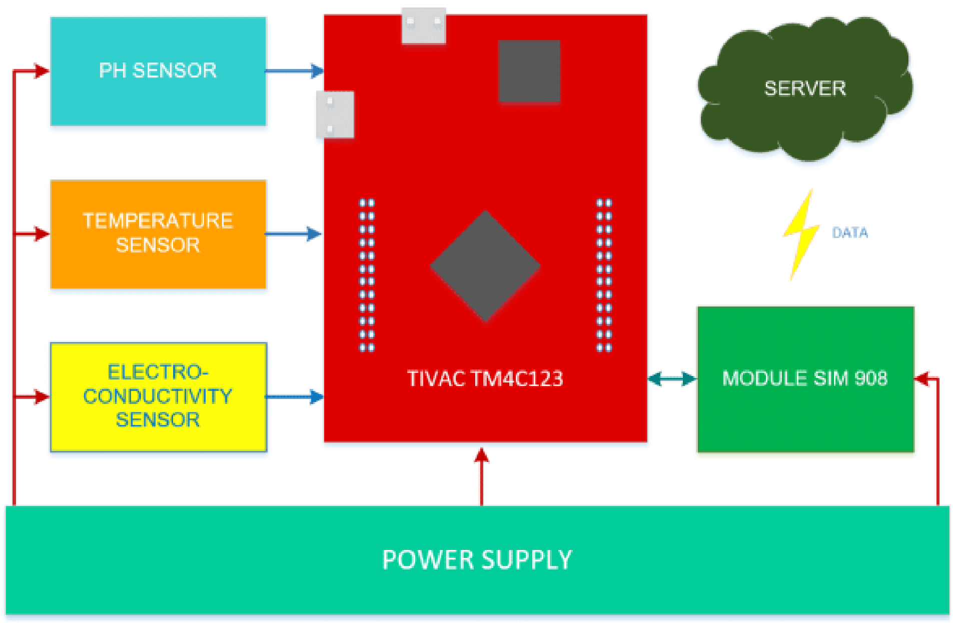
The development of a smart water fountain can be complex, depending on its intended purpose and features. It's important to work with a multidisciplinary team, including engineers, software developers, and user experience designers, to ensure a successful and user friendly.



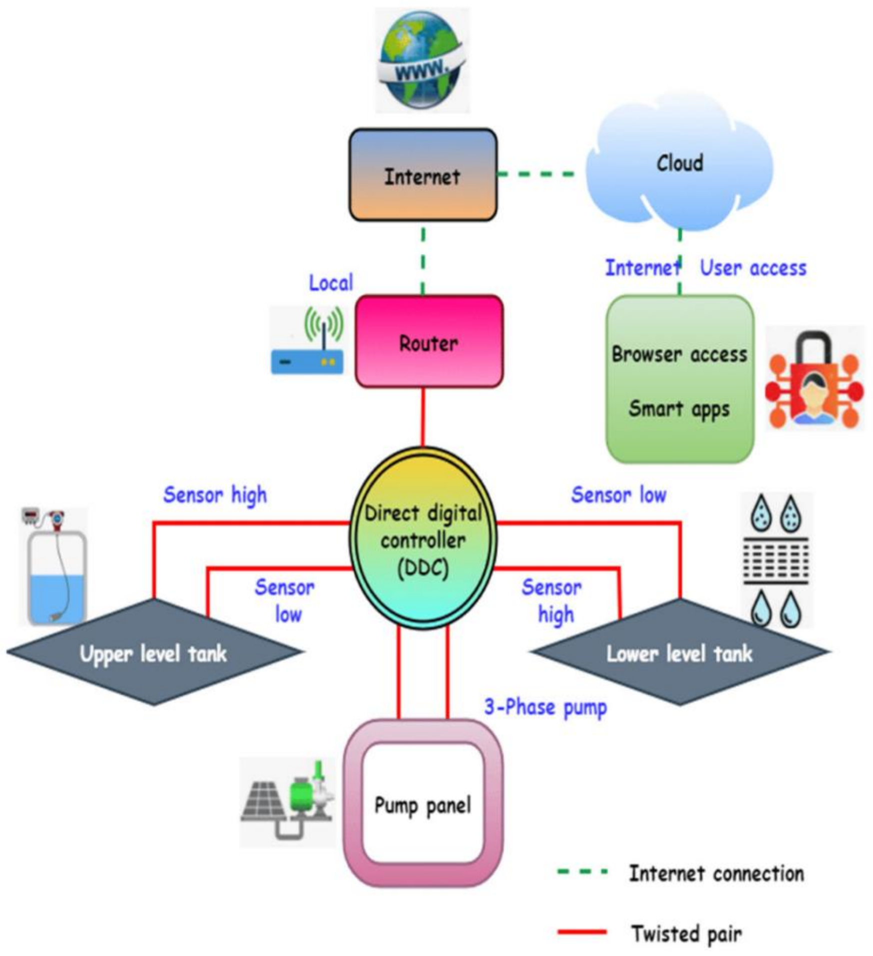
BLOCK DIAGRAM:


Power Supply
pH Sensor
Turbidity Sensor
DHT-11 Sensor
MCU (Arduino Mega)
Wi-Fi Module (Node MCU)
Ultrasonic Sensor
PC
CLOUD
PC (ThingSpeak)


Water Sensor
Arduino UNO R3
Servo Motor
Inlet
opens: when rainfalls
closes: when container
overflows or when
rainfall stops.
Power Supply



<u>**IOT BASED DESIGN:**</u>



<u>ROAD MAP FOR DEVELOPMENT OF SMART WATER FOUNTAIN:</u>

Creating a roadmap for the development and deployment of a smart water fountain involves breaking down the project into manageable phases and tasks. Here's a roadmap that outlines the key steps and milestones for developing a smart water fountain:

Phase 1: Planning and Conceptualization

1. Define Objectives and Purpose:

- Determine the primary goals of the smart water fountain (e.g., water conservation, public access to drinking water, decorative feature).

2. Market Research:

- Research existing smart fountains and water-related technologies to understand the competition and identify gaps in the market.

3. Budget and Resource Allocation:

- Determine the budget and allocate resources, including personnel, materials, and technology.

4. Regulatory Compliance:

- Investigate any legal and regulatory requirements related to water quality, public access, and data privacy.

5. Team Formation:

- Assemble a multidisciplinary team with expertise in hardware, software, design, and project management.

Phase 2: Design and Prototyping

6. Conceptual Design:

- Develop the overall design and aesthetics of the smart water fountain.

7. Hardware Selection:

- Choose the required components, including pumps, sensors, valves, filtration systems, and power sources.

8. Software Development:

- Begin developing the control software for the fountain.

9. Prototype Construction:

- Build a working prototype of the smart water fountain to test the concept.

10. User Interface Design:

- Create the user interface for controlling and monitoring the fountain (app, web interface, or physical controls).

Phase 3: Testing and Refinement

11. Prototype Testing:

- Test the prototype for functionality, reliability, and user-friendliness.

12. User Feedback:

- Gather feedback from potential users and stakeholders to make necessary improvements.

13. Refinement:

- Adjust and refine the hardware and software based on user feedback and test results.

Phase 4: Full-Scale Development

14. Manufacturing and Assembly:

- Begin manufacturing components and assembling the smart water fountain.

15. Software Development:

- Finalize the control software and ensure its stable and efficient.

16. Data Management and Security:

- Implement data storage, security, and privacy measures.

Phase 5: Deployment and Integration

17. Deployment:

- Install the smart water fountain at its intended location.

18. Integration:

- Integrate the fountain with the chosen data storage and analysis systems.

Phase 6: Monitoring and Maintenance

19. Monitoring:

- Set up monitoring systems to track the fountain's performance, water quality, and user interactions.

20. Maintenance Plan:

- Develop a maintenance plan for regular cleaning, component replacement, and software updates.

Phase 7: User Education and Marketing

21. User Education:

- Educate users about how to use the smart water fountain and its features.

22. Marketing and Promotion:

- Create marketing materials and campaigns to promote the fountain.

Phase 8: Scaling and Expansion

23. Scaling:

- Consider scaling the solution by deploying smarter water fountains in different locations if the initial deployment is successful.

Phase 9: Continuous Improvement

24. Feedback and Iteration:

- Continue gathering user feedback and making improvements to the smart water fountain based on user needs and evolving technology.

Phase 10: Sustainability and Evaluation

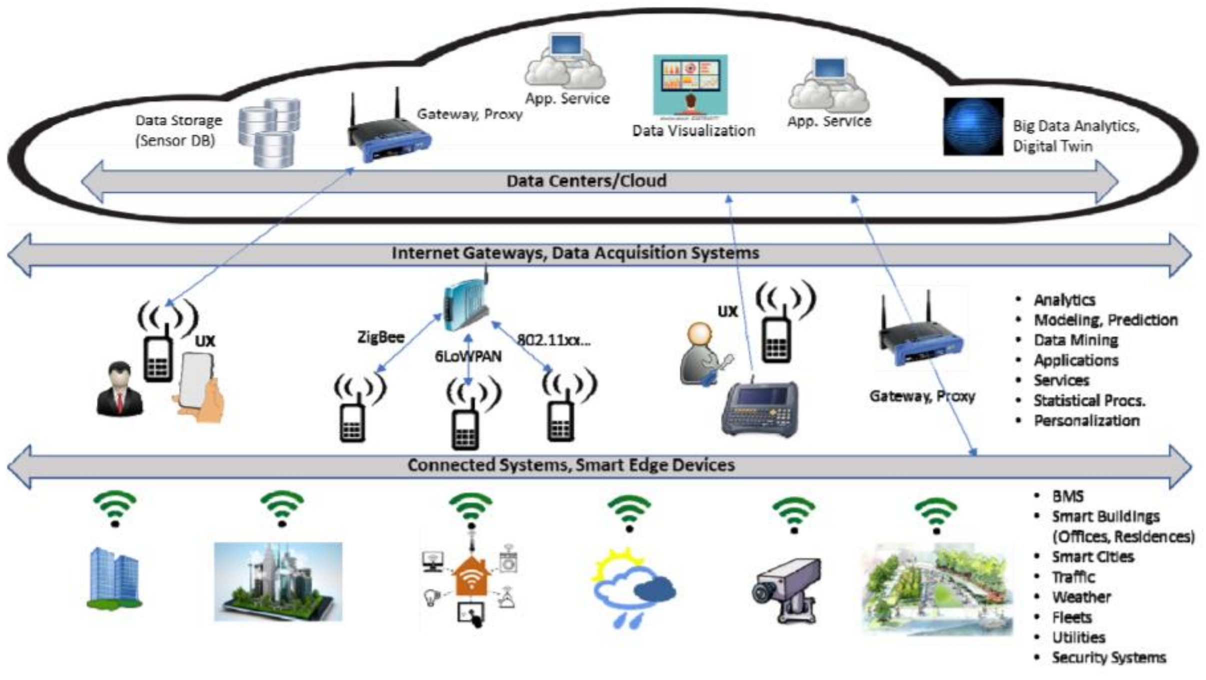
25. Sustainability:

- Evaluate the environmental impact of the smart water fountain and consider implementing sustainability features, such as renewable energy sources or water-saving technologies.

26. Performance Evaluation:

- Regularly assess the smart water fountain's performance and impact in achieving its objectives.

This roadmap provides a structured approach to developing and deploying a smart water fountain, taking into account the different stages and considerations involved in the process. Each phase should be carefully planned and executed to ensure a successful project.



BMS Smart Building/City IoT Architecture

CODING (DEVELOPMENT)

PYTHON:

```
# Import necessary libraries
import sensors # Assuming you have a library for sensor inputs
import pumps    # Library for controlling water pumps
import valves   # Library for controlling water valves
import user_interface  # Library for the user interface

# Initialize sensors and actuators
water_level_sensor = sensors.WaterLevelSensor()
temperature_sensor = sensors.TemperatureSensor()
```

```
pump = pumps.WaterPump()
valve = valves.WaterValve()
user_interface = user_interface.UserInterface()

# Main control loop
while True:
    # Read sensor data
    water_level = water_level_sensor.read_level()
    water_temperature = temperature_sensor.read_temperature()

    # Check water level and take action if it's too low
    if water_level < MINIMUM_WATER_LEVEL:
        pump.turn_on() # Activate the water pump to refill the
fountain
        user_interface.display_message("Refilling the fountain.")

    # Check water temperature and take action if it's too high
    if water_temperature >
MAXIMUM_WATER_TEMPERATURE:
        valve.close() # Close the valve to stop the water flow
        user_interface.display_message("Water temperature too high.
Cooling down.")

    # Check for user input
    user_input = user_interface.get_user_input()
    if user_input == "on":
        pump.turn_on() # Turn on the fountain pump
    elif user_input == "off":
        pump.turn_off() # Turn off the fountain pump
    elif user_input == "adjust_flow":
        flow_rate = user_interface.get_desired_flow_rate()
        pump.set_flow_rate(flow_rate)  # Adjust the water flow rate

    # Check for emergency shutdown signal
    if user_interface.emergency_shutdown_requested():
        pump.turn_off()
        valve.close()
        user_interface.display_message("Emergency shutdown
initiated.")
        break # Exit the control loop

# Cleanup and exit the program
user_interface.cleanup()
pump.cleanup()
valve.cleanup()
sensors.cleanup()
```

JAVA:

```
import java.util.Scanner;

public class SmartWaterFountain {

    private double waterLevel;
    private boolean isFountainOn;

    public SmartWaterFountain() {
        waterLevel = 100.0; // Initial water level in the fountain (e.g., in
liters).
        isFountainOn = false; // Fountain is initially turned off.
    }

    public void turnOnFountain() {
        isFountainOn = true;
        System.out.println("Fountain is now ON.");
    }

    public void turnOffFountain() {
        isFountainOn = false;
        System.out.println("Fountain is now OFF.");
    }

    public void fillFountain(double amount) {
        waterLevel += amount;
        System.out.println("Fountain has been filled with " + amount +
" liters of water.");
    }

    public void dispenseWater(double amount) {
        if (isFountainOn) {
            if (waterLevel >= amount) {
                waterLevel -= amount;
                System.out.println(amount + " liters of water
dispensed.");
            } else {
                System.out.println("Not enough water in the fountain.
Please fill it.");
            }
        } else {
            System.out.println("Fountain is off. Turn it on to dispense
water.");
        }
    }

    public double getWaterLevel() {
```

```
    return waterLevel;
}

public static void main (String [] args) {
    SmartWaterFountain fountain = new SmartWaterFountain();
    Scanner scanner = new Scanner (System.in);

    while (true) {
        System.out.println("Select an option:");
        System.out.println("1. Turn on the fountain");
        System.out.println("2. Turn off the fountain");
        System.out.println("3. Fill the fountain");
        System.out.println("4. Dispense water");
        System.out.println("5. Check water level");
        System.out.println("6. Exit");

        int choice = scanner.nextInt();

        switch (choice) {
            case 1:
                fountain.turnOnFountain();
                break;
            case 2:
                fountain.turnOffFountain();
```

```
                    break;
                case 3:
                    System.out.print("Enter the amount to fill (liters): ");
                    double fillAmount = scanner.nextDouble();
                    fountain.fillFountain(fillAmount);
                    break;
                case 4:
                    System.out.print("Enter the amount to dispense (liters):
");
                    double dispenseAmount = scanner.nextDouble();
                    fountain.dispenseWater(dispenseAmount);
                    break;
                case 5:
                    System.out.println("Water level: " +
fountain.getWaterLevel() + " liters");
                    break;
                case 6:
                    System.out.println("Exiting the program.");
                    System.exit(0);
                default:
                    System.out.println("Invalid choice. Please select a valid
option.");
            }
        }
```

```
    }
}
```

HTML:

```
<!DOCTYPE html>
<html>
<head>
    <title>Smart Water Fountain Control</title>
</head>
<body>
    <h1>Smart Water Fountain Control</h1>

    <p>Status: <span id="status">Fountain is OFF</span></p>

    <button id="turnOnButton">Turn On</button>
    <button id="turnOffButton">Turn Off</button>

    <script>
        // JavaScript code for controlling the smart water fountain

        // Function to update the status text
        function updateStatus(statusText) {
            document.getElementById("status").textContent =
statusText;
        }

        // Event handler for the "Turn On" button

document.getElementById("turnOnButton").addEventListener("cli
ck", function () {
            // You can use JavaScript to send a request to the fountain
control system to turn the fountain on.
            // This may involve making an API call to the fountain
controller.
            // For this example, we'll simply update the status text.
            updateStatus("Fountain is ON");
        });

        // Event handler for the "Turn Off" button

document.getElementById("turnOffButton").addEventListener("cl
ick", function () {
            // You can use JavaScript to send a request to the fountain
control system to turn the fountain off.
            // This may involve making an API call to the fountain
controller.
            // For this example, we'll simply update the status text.
            updateStatus("Fountain is OFF");
        });
    </script>
</body>
</html>
```

CSS:

```
/* Define basic styling for the user interface */
body {
  font-family: Arial, sans-serif;
  background-color: #f2f2f2;
}

.container {
  max-width: 600px;
  margin: 0 auto;
  padding: 20px;
  background-color: #fff;
  border-radius: 5px;
  box-shadow: 0 0 10px rgba(0, 0, 0, 0.2);
}

h1 {
  color: #333;
  text-align center;
}

/* Style buttons for controlling the water fountain */
. button {
  display: inline block;
  padding: 10px 20px;
  background-color: #007BFF;
  color: #fff;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  margin: 10px;
}

.button:hover {
  background-color: #0056b3;
}

/* Style status indicators */
. status {
  text-align center;
  font-size: 18px;
  margin-top: 20px;
}
```

```css
.status.active {
  color: green;
}

.status.inactive {
  Color: red;
}
```

<u>ESP8266:</u>

Creating a web server using an ESP8266 microcontroller for a smart water fountain allows you to control and monitor the fountain remotely. Here's a simple example of Arduino code for an ESP8266-based web server for your smart water fountain. This code assumes you're using the Arduino IDE with the ESP8266 core installed:

```cpp
#include <ESP8266WiFi.h>
#include <ESP8266WebServer.h>

const char* ssid = "YourWiFiSSID"; // Replace with your WiFi network SSID
const char* password = "YourWiFiPassword"; // Replace with your
WiFi network password

ESP8266WebServer server (80);

void setup () {
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WIFI. Status()! = WL_CONNECTED) {
    delay (1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to Wi-Fi");

  // Define web server routes
  server. On("/", HTTP_GET, handle Root);
  server. On("/fountain/on", HTTP_GET, handleFountainOn);
  server. On("/fountain/off", HTTP_GET, handleFountainOff);

  // Start server
  server. Begin();
  Serial.println("Web server started");
}
```

```
void loop () {
  server.handleClient();
}

void handleRoot() {
  // HTML webpage for controlling the fountain
  String html = "<html><body>";
  html += "<h1>Smart Water Fountain Control</h1>";
  html += "<p><a href='/fountain/on'>Turn Fountain On</a></p>";
  html += "<p><a href='/fountain/off'>Turn Fountain
Off</a></p>";
  html += "</body></html>";
  server.send(200, "text/html", html);
}

void handleFountainOn() {
  // Code to turn the fountain on
  // Insert your control logic here
  server.send(200, "text/plain", "Fountain turned on");
}

void handleFountainOff() {
  // Code to turn the fountain off
  // Insert your control logic here
  server.send(200, "text/plain", "Fountain turned off");
}
```

In this code:

1. Include the necessary libraries for ESP8266 WiFi and web server.

2. Set your Wi-Fi network credentials (replace "YourWiFiSSID" and "YourWiFiPassword" with your network details).

3. Define the web server routes for the root ("/"), turning the fountain on ("/fountain/on"), and turning it off ("/fountain/off").

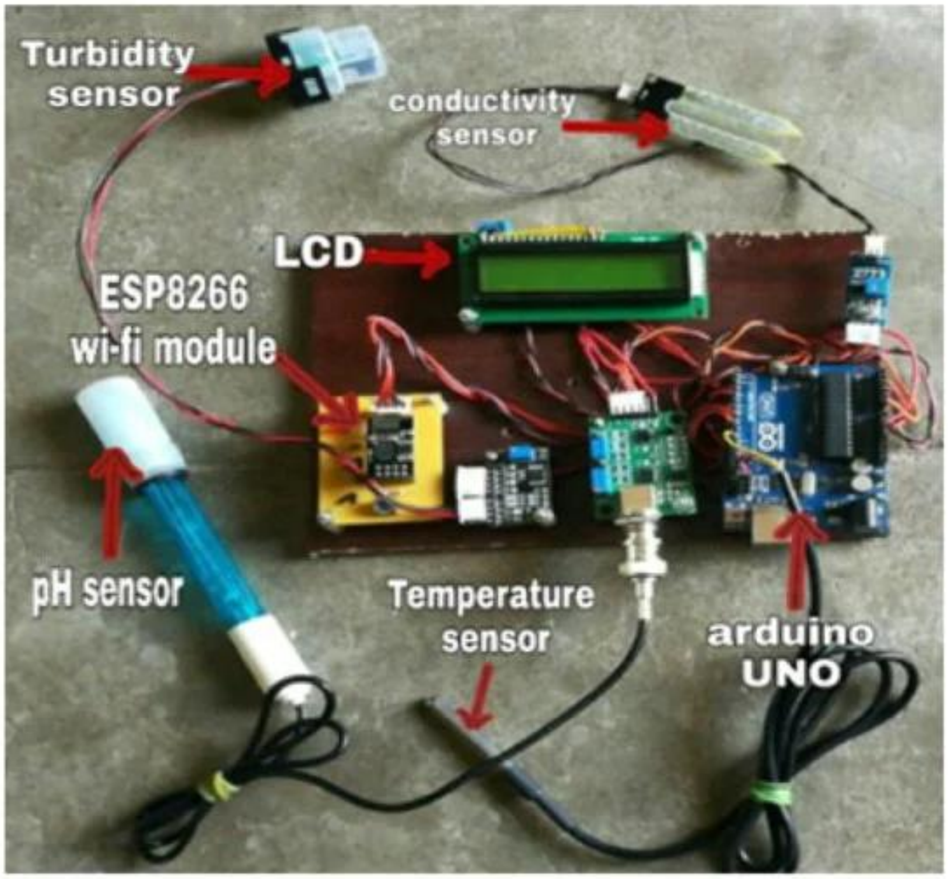
4. In the `setup` function, connect to Wi-Fi and start the web server.

5. The `loop` function is dedicated to handling incoming client requests.

6. The `handleRoot` function serves an HTML page with links to control the fountain.

7. The `handleFountainOn` and `handleFountainOff` functions are called when the respective links are clicked. You should replace the placeholder code with your actual fountain control logic.

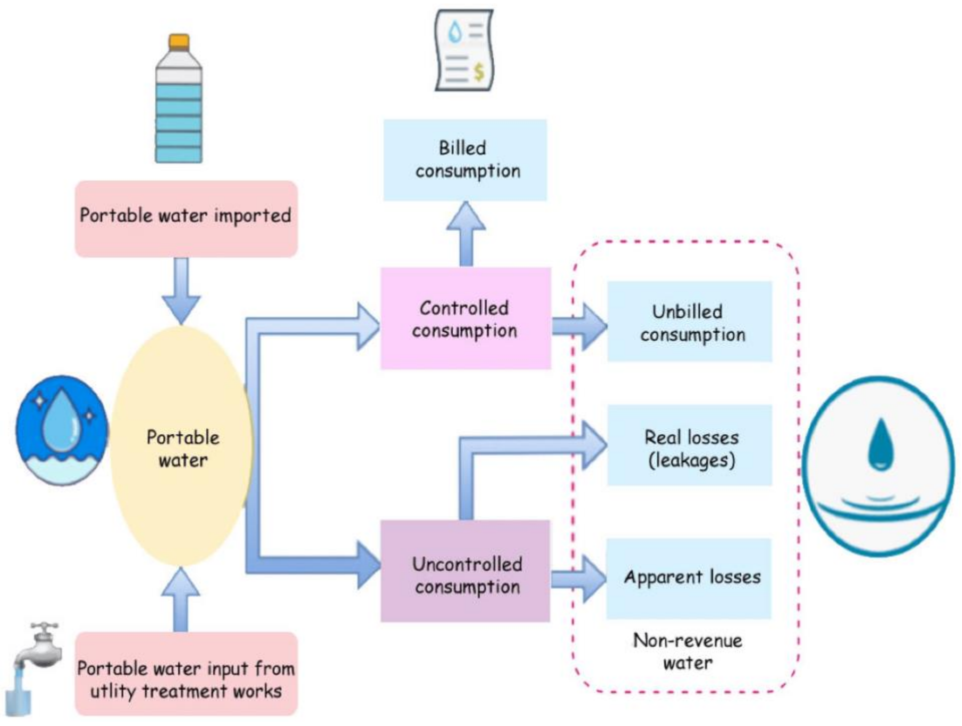
Make sure to adapt this code to your specific needs and the components of your smart water fountain. You may need to integrate sensors, pumps, and valves, and adjust the control logic accordingly. Additionally, remember to set up your Arduino IDE with the necessary board support for the ESP8266.



<u>CONCLUSION:</u>

In conclusion, the development of a smart water fountain is a multifaceted process that combines hardware and software components to create a more efficient, user-friendly, and data-driven water feature. The key takeaways from this endeavor are as follows:

1. Diverse Applications: Smart water fountains can serve various purposes, including water conservation, providing clean drinking water, or creating interactive art installations. The specific objectives of the project should guide its development.



2. Key Hardware Components: The hardware components, such as water supply, pumps, sensors, valves, filtration systems, power supplies, and microcontrollers, form the foundation of the smart water fountain.

3. Software Integration: The development of a smart water fountain requires software for user interfaces, data processing, automation, connectivity, data storage and analysis, and energy efficiency.

4. Data-Driven Decision Making: The integration of sensors and data analysis enables informed decisions about water usage and allows for real-time monitoring.

5. Maintenance and Safety: To ensure the fountain's longevity and safety, attention should be given to maintenance and safety features. This is especially important for public installations.

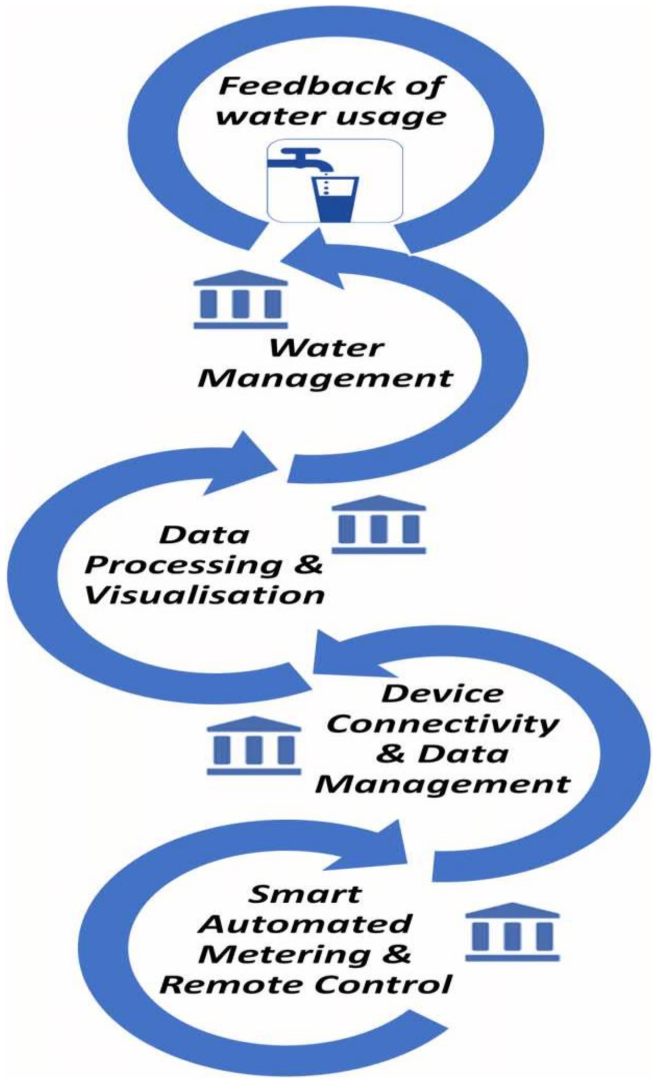
6. User Education: Clear user instructions and education are essential to ensure proper interaction with the smart water fountain.

7. Data Privacy and Security: Handling any collected data in a secure and privacy-compliant manner is crucial, especially for public installations that involve data collection.

8. Deployment and Scaling: After thorough testing, deployment in the intended location is the next step, with potential for scaling the solution if it meets its objectives.

9. Continuous Improvement: Regular monitoring and updates are necessary to maintain the system's performance and add new features or address issues as they arise.

The development of a smart water fountain can have a significant impact, whether it's in terms of conserving water resources, providing essential services, or enhancing public spaces with interactive features. Careful planning, multidisciplinary collaboration, and attention to user needs are fundamental to a successful smart water fountain project.


Feedback of water usage
Water Management
Data Processing & Visualisation
Device Connectivity & Data Management
Smart Automated Metering & Remote Control