



# SMART WATER FOUNTAIN

PHASE-V

PROJECT DOCUMENTATION &  
SUBMISSION

Internet  
of things

DONE BY:

- ARUN KUMAR.R (au82042106004)
- DHINESH.R (au82042106011)
- HARIHARASUDHAN.S (au82042106014)
- MILTON EMIRATE RAJ.A (au82042106029)
- MUKESH.M (au82042106030)



## **Objectives:**

The Smart Water Fountain project aims to create an innovative and technologically advanced water fountain system that offers a range of features beyond a traditional decorative fountain. The primary objectives of this project include:

**Enhanced Aesthetics:** Improve the visual appeal of the water fountain by integrating customizable lighting, water jets, and patterns to create mesmerizing displays.

**Water Quality Monitoring:** Implement sensors to continuously monitor water quality parameters such as pH, turbidity, and temperature to ensure water safety and quality.

**Energy Efficiency:** Optimize energy usage by enabling variable control of water pump speed and lighting, based on real-time conditions and user preferences.

**User Interaction:** Develop a user-friendly mobile app that allows users to remotely control fountain features, change lighting colors and patterns, and receive notifications about water quality and system status.



**Data Analysis:** Collect and store historical data to enable trend analysis and make informed decisions regarding water treatment and maintenance.

**Maintenance Alerts:** Generate alerts for maintenance needs, such as cleaning filters, refilling water, or replacing components, to ensure the fountain proper operation.

**Customizability:** Allow users to personalize the fountain's appearance and behavior to suit various occasions and moods.

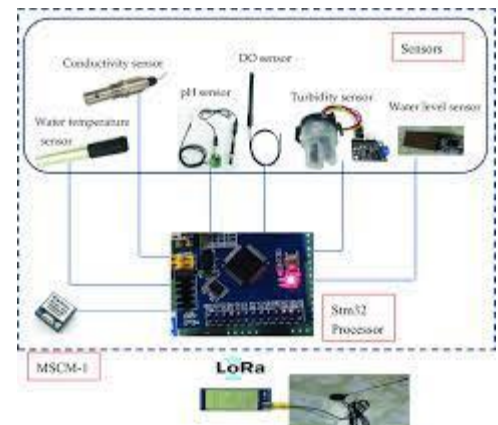
**Education and Awareness:** Promote water conservation and awareness about water quality through the display of relevant information on the mobile app.



Overall, the Smart Water Fountain project aims to create an engaging and functional water feature that enhances user experience, promotes water conservation, and ensures the safety and quality of the fountain water.

## IoT sensor setup

The IoT sensor setup for the Smart Water Fountain project involves a network of sensors and control components that provide real-time data and enable automated control of various aspects of the fountain. The primary sensors and components include:



## Water Quality Sensors:

**pH Sensor:** Measures the acidity or alkalinity of the water to ensure water quality.

**Turbidity Sensor:** Determines the cloudiness or clarity of the water, which can indicate impurities or sediment.

**Temperature Sensor:** Monitors the water temperature, ensuring it remains within the desired range for safety and comfort.

## Flow Sensors:

**Flow Rate Sensor:** Measures the rate of water flow through the fountain's pipes and nozzles.

**Obstruction Detection:** Detects any blockages or obstructions in the water flow, which can affect the fountain's performance.

**Water Level Sensor:**

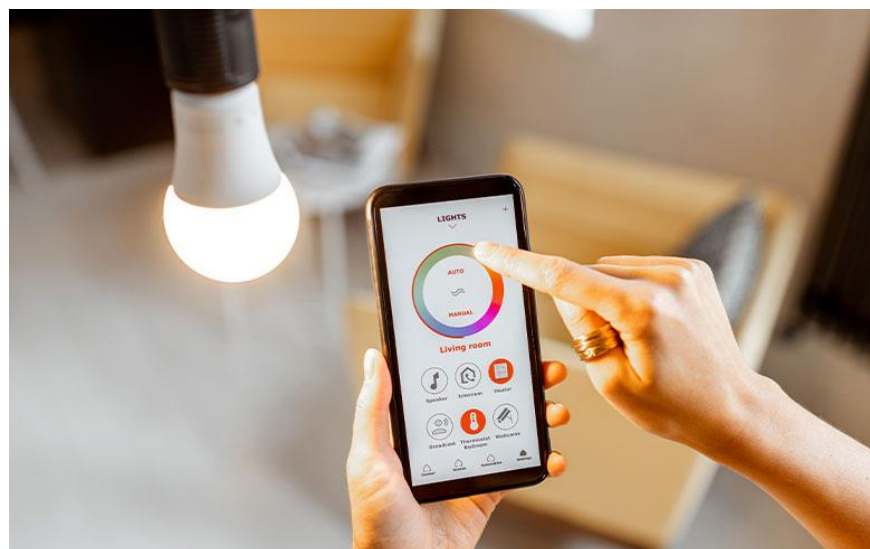
Water level sensors can detect the fountain's water level and trigger alerts if the water level falls below a safe threshold.



## Pump Control:

**Smart Water Pump:** A controllable pump allows for adjusting the water flow rate and pressure as needed.

**Lighting Control: RGB LED Lights:** These lights can change colors and intensity to create various lighting effects. They are controllable via the IoT system and can be synchronized with the water features.



## **Raspberry Pi Integration:**

Raspberry Pi: Acts as the central hub to collect data from sensors and control components. It processes data, runs control logic, and interfaces with the mobile app.

## **Mobile App Interface:**

Mobile Application: The mobile app communicates with the Raspberry Pi and allows users to interact with the fountain, change lighting colors, patterns, and control water features remotely.

## **Water Treatment System:**

Water Filtration System: May include sensors to monitor the condition of water filters and alert when maintenance is required.

Water Sterilization System: Utilizes UV or chemical treatments, with sensors to monitor treatment effectiveness.

The sensors and components are strategically placed within the fountain infrastructure and are connected to the central Raspberry Pi controller. The sensors continuously collect data and send it to the Raspberry Pi for processing and analysis. The Raspberry Pi processes this data, controls the fountain's operation, and communicates with the mobile app to provide real-time updates to users.

## **Mobile App Development:**

Mobile App Development for the Smart Water Fountain project is a critical component that empowers users to interact with the fountain, control its features, monitor water quality, and receive real-time information. Here's an overview of the key features and functionalities that the mobile app should include:



**User Authentication:** Users should be required to create an account and log in to access the app. User profiles allow for personalization of settings and preferences.

**Dashboard:** The dashboard is the central interface of the app and provides an overview of the fountain's status, water quality parameters, and energy usage.

Users can quickly see if the fountain is running, monitor water quality data (pH, turbidity, temperature), and view energy consumption statistics.



### **Fountain Control:**

**Start/Stop Functionality:** Users can remotely start and stop the water fountain, controlling the water flow and lighting effects.

**Variable Pump Control:** Adjust the pump's speed and water flow rate based on user preferences and real-time conditions.

**Lighting Control:** Change the color and intensity of RGB LED lights to create different visual effects.

**Fountain Patterns:** Select from predefined fountain patterns or create custom patterns with varying water jets and lighting effects.

**Water Quality Monitoring:Real-time Data:** Display water quality parameters such as pH, turbidity, and temperature in real-time.

**Historical Data:** Users can access historical data and trends to monitor changes in water quality over time.



### Energy Efficiency:

**Optimize Pump and Lighting:** Offer options to optimize energy usage by adjusting pump and lighting operations based on user preferences and sensor data.

**Energy Consumption:** Display energy consumption data to raise awareness and encourage efficient usage.

### Notifications and Alerts:

**Alerts for Maintenance:** Notify users when maintenance is required, such as filter cleaning or water replenishment.

**Water Quality Alerts:** Send alerts if the water quality falls below safe levels.

**Fountain Status:** Receive notifications about the fountain's status, including when it is running or stopped.

## **Customization:**

**User Preferences:** Allow users to customize their fountain experience by saving favorite settings.

**Seasonal Themes:** Offer seasonal or event-based themes for the fountain, such as holiday displays.



**Scheduling:** Set schedules for when the fountain should operate and change lighting patterns.

## **Education and Awareness:**

**Information Section:** Include educational content about water conservation, water quality, and fountain maintenance.  
**Water Saving Tips:** Provide tips and suggestions to promote responsible water use.

**User Support:**

**Contact Support:** Offer a feature to contact customer support or request technical assistance.

**FAQs:** Include a frequently asked questions section for quick reference.

**User Feedback:**

**Feedback Form:** Allow users to provide feedback and report issues for continuous improvement.

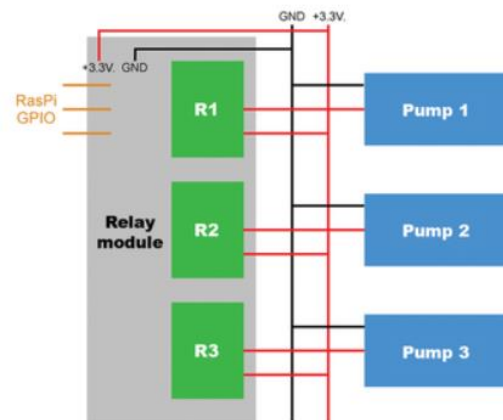
The mobile app should be user-friendly, compatible with both Android and iOS platforms, and regularly updated to ensure it meets user expectations and maintains compatibility with the IoT system and sensors integrated into the Smart Water Fountain. The app enhances the overall user experience, promotes water conservation, and enables remote control and monitoring of the fountain's features.



# Raspberry Pi integration:

Raspberry Pi integration in the Smart Water Fountain project serves as the central control unit that manages the various components of the fountain, collects data from sensors, processes information, and interfaces with the mobile app. Here is a detailed description of the Raspberry Pi integration in this project

**Data Collection:**The Raspberry Pi collects data from various sensors within the fountain, including water quality sensors (pH, turbidity, temperature), flow sensors, water level sensors, and any other relevant sensors such as temperature sensors for ambient conditions.

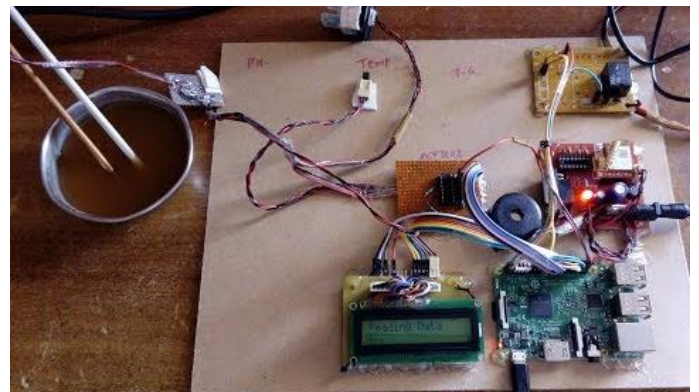


**Data Processing:**Data collected from sensors are processed and analyzed by the Raspberry Pi. This includes real-time monitoring of water quality parameters, water flow rates, and energy usage.

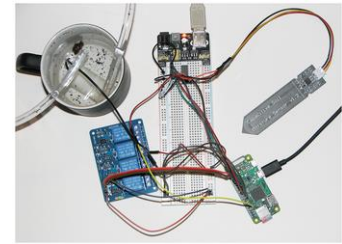
The Raspberry Pi runs control algorithms to determine the optimal operation of the fountain, adjusting water flow and lighting based on sensor data and user preferences.

**Communication with IoT Sensors:**The Raspberry Pi interfaces with IoT sensors through appropriate communication protocols (e.g., I2C, SPI, GPIO pins) to collect data and send control commands.

It manages bidirectional communication with sensors, receiving data and sending instructions to control components such as the water pump and lighting system.



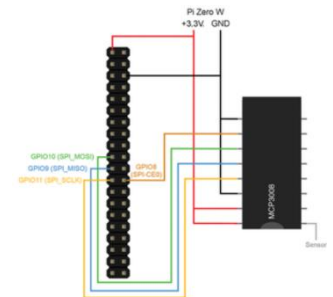
**Mobile App Interface:**the mobile app to provide users with real-time status updates, water quality information, and control options.It acts as a bridge between the mobile app and the physical fountain components, enabling users to remotely control the fountain's operation.



**Control Logic:**The Raspberry Pi hosts the control logic for the fountain, which determines when to start or stop the fountain, adjust the water flow rate, and change lighting patterns.

The control logic may take into account user preferences, sensor data, energy efficiency goals, and scheduling information.

**Energy Management:**The Raspberry Pi is responsible for optimizing energy usage by controlling the pump and lighting components. It can adjust the operation of these components based on real-time conditions and user-defined preferences.



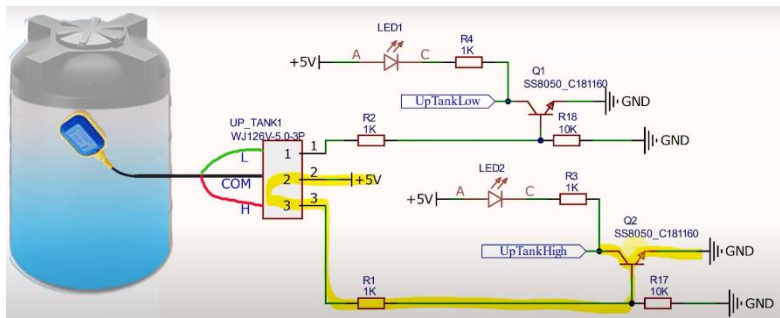
**Alert Generation:**The Raspberry Pi generates alerts and notifications for various conditions, such as low water quality, maintenance needs (e.g., filter cleaning), and fountain status (running or stopped).

These alerts are sent to the mobile app for user awareness.

**Historical Data Storage:**The Raspberry Pi may store historical sensor data, allowing users to access past records and trends related to water quality, energy consumption, and fountain operation.

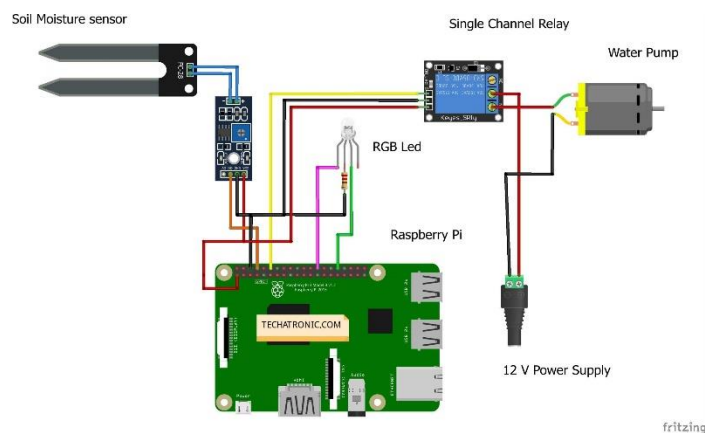
**Security:**The Raspberry Pi can include security features to ensure secure communication with the mobile app and protect the integrity of the system.

**Software Updates:**Regular software updates for the Raspberry Pi should be implemented to maintain system reliability and security. These updates may include bug fixes, enhancements, and improved compatibility.



The integration of Raspberry Pi into the Smart Water Fountain project centralizes control and data processing, making it a versatile hub

for managing the fountain's operation and interaction with users. It plays a pivotal role in ensuring the system's efficient and responsive operation while enhancing user experience and promoting water conservation.



## code implementation:

The code implementation for the Smart Water Fountain project involves several components, including code for IoT sensors, Raspberry Pi integration, mobile app development, and possibly cloud services. Here's an overview of the code implementation for each of these components:

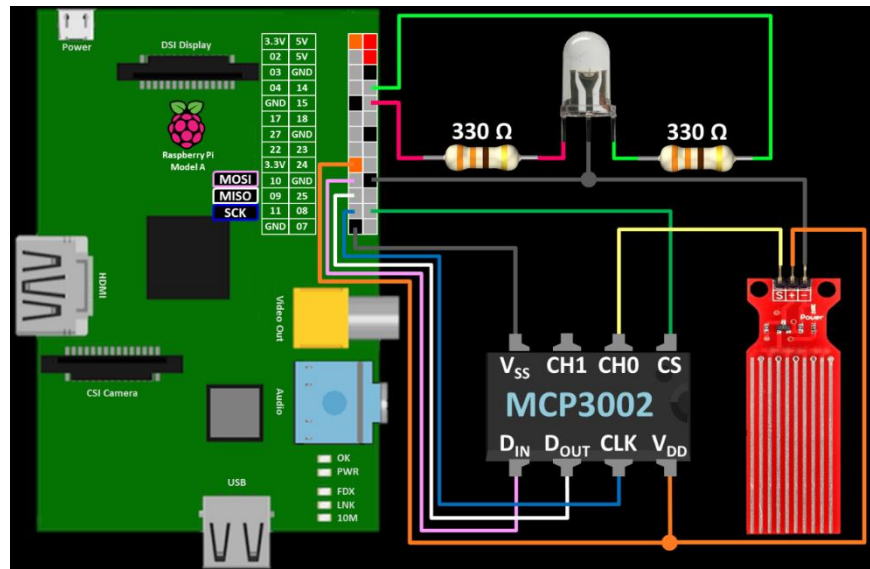
### 1. IoT Sensor Code:

- Code for the individual IoT sensors (e.g., pH sensor, turbidity sensor, flow sensor) should be developed.
- Sensor code collects data from the respective sensor and transmits it to the Raspberry Pi or central controller.
- The code may also include error handling and calibration routines to ensure accurate data collection.



## 2. Raspberry Pi Integration:

- Code running on the Raspberry Pi serves as the central control unit for the fountain. It collects data from sensors and communicates with the mobile app.
- Raspberry Pi code includes data processing, control logic, and communication with the IoT sensors.
- Control algorithms determine when to start or stop the fountain, adjust pump and lighting operations, and generate alerts.
- Security measures may be implemented to protect communication and data integrity. Historical data storage code stores sensor data for future analysis.

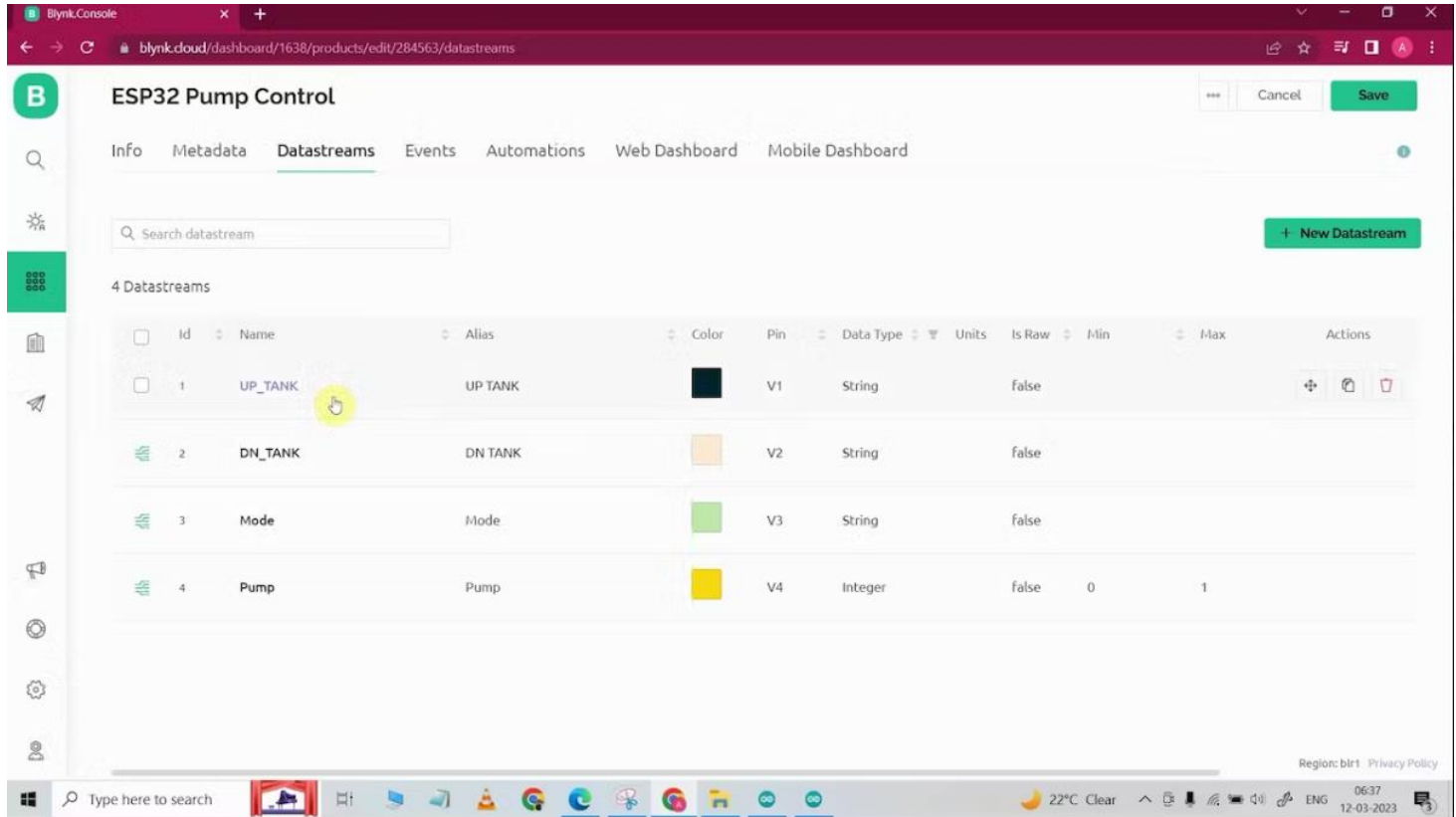


### 3. Mobile App Development:

- ✚ The mobile app code is developed for Android and/or iOS platforms using relevant programming languages (e.g., Java, Kotlin, Swift).
- ✚ Code includes user authentication and user profile management.
- ✚ Code for the app's dashboard displays real-time and historical data, such as water quality, energy consumption, and fountain status.
- ✚ Fountain control code enables users to remotely control the fountain's operation, adjust pump speed, change lighting patterns, and schedule fountain activities.
- ✚ Notification code generates alerts and notifications for various conditions, including water quality issues and maintenance needs.
- ✚ Code for data visualization and charting presents historical data in a user-friendly manner.
- ✚ Here, I'll provide a simplified example of Python code to demonstrate the essential components of such a project.

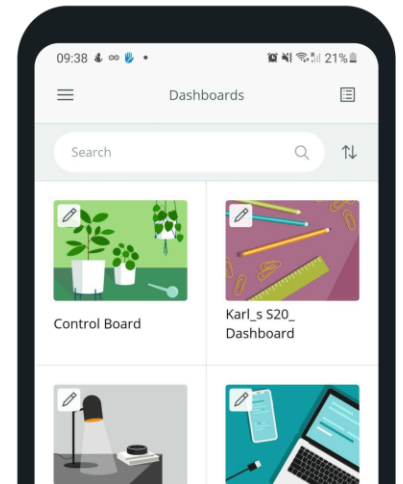
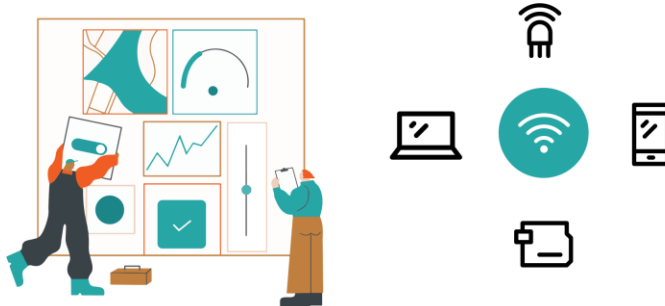


- ✚ The app code interfaces with the Raspberry Pi to send control commands and receive real-time updates.



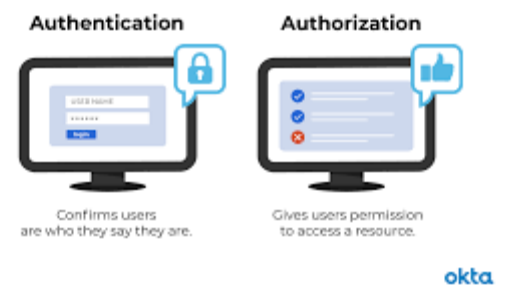
#### 4. Cloud Integration (optional):

- ✚ If cloud services are used, code may be implemented to transfer and store data in a cloud platform for backup and remote access.
- ✚ Cloud code may involve setting up databases, APIs, and data synchronization processes between the fountain system and the cloud.
- ✚ IoT Gateway: In many cases, IoT gateways are used to preprocess and aggregate data before sending it to the cloud. These gateways can filter, format, and reduce data volumes, optimizing the data transfer process.
- ✚ Data Processing: Use cloud services for data processing and analytics. You can employ serverless computing, containers, or virtual machines to run applications that analyze and derive insights from IoT data.



## 5. User Authentication and Authorization:

- Both the mobile app and the Raspberry Pi should include code for user authentication and authorization to ensure secure access to the system.
- Authentication code may include registration, login, and password reset features.
- Authorization code determines user permissions and access levels.



## 6. Software Updates:

- Code should be in place to handle software updates for the Raspberry Pi and the mobile app, allowing for regular maintenance and enhancements.

## 7. User Feedback and Support:

- The app may include a feedback form or support contact feature, and the corresponding code should handle user feedback and inquiries.

The code implementation is a crucial aspect of the Smart Water Fountain project, ensuring that data is collected



accurately, control logic is executed effectively, and users can interact with the fountain seamlessly. Regular updates and maintenance are essential to keep the system running smoothly and securely. Additionally, thorough testing should be performed to validate the system's functionality and reliability.


## Python script on how to replicate and deploy IoT sensors

Creating a comprehensive Python script for replicating the Smart Water Fountain project and deploying IoT sensors is a complex task, and it's typically divided into various script files. Here, I'll provide a simplified example of Python code to demonstrate the essential components of such a project. Keep in mind that this example does not cover every detail but serves as a starting point for your project:

### 1. Import Necessary Libraries

#### 1. Import Necessary Libraries:

python

 Copy code


```
import time
import random

# Additional libraries depending on your specific sensors and hardware
# e.g., RPi.GPIO, smbus, Adafruit libraries
```

### 2. Initialize IoT Sensors (Simulated in This Example)

#### 2. Initialize IoT Sensors (Simulated in This Example):

python

 Copy code

```
class Sensor:
    def __init__(self, name):
        self.name = name


    def read_value(self):
        # Simulated data; replace with real sensor data
        return random.uniform(0, 1)

# Replace with your actual sensor initialization
ph_sensor = Sensor("pH Sensor")
turbidity_sensor = Sensor("Turbidity Sensor")
temperature_sensor = Sensor("Temperature Sensor")
```

### 3.Data Collection and Storage

#### 3. Data Collection and Storage:

python

 Copy code

```
def collect_sensor_data():
    while True:
        timestamp = time.strftime('%Y-%m-%d %H:%M:%S')
        ph_value = ph_sensor.read_value()
        turbidity_value = turbidity_sensor.read_value()
        temperature_value = temperature_sensor.read_value()

        # Log the data to a file (replace with your preferred data storage)
        with open('sensor_data.txt', 'a') as file:
            file.write(f'{timestamp}, pH: {ph_value}, Turbidity: {turbidity_value}, Temperature: {temperature_value}\n')

        time.sleep(10) # Adjust the interval based on your requirements

# Start data collection in a separate thread
# You may want to use Python's threading or multiprocessing module
```

**Data Sources:** IoT devices can collect data from a wide range of sources, including sensors (e.g., temperature, humidity, motion, and GPS), cameras, actuators, and more. Identify the types of data your IoT system will generate.

**Data Ingestion:** Establish mechanisms for collecting data from IoT devices. This can involve direct device-to-cloud communication, gateways, or edge computing for preliminary data processing.


**Data Protocols:** Define communication protocols for data exchange between devices and data collection points. Common protocols include MQTT, CoAP, and HTTP.

**Security:** Implement robust security measures to protect data during collection and storage. Encryption, authentication, and authorization are critical for safeguarding sensitive IoT data.

## 4. Control Pump and Lights (Simulated in This Example):

### 4. Control Pump and Lights (Simulated in This Example):

python

 Copy code

```
class WaterFountain:
    def __init__(self):
        self.pump_running = False
        self.lights_on = False

    def start_pump(self):
        self.pump_running = True
        print("Pump started")

    def stop_pump(self):
        self.pump_running = False
        print("Pump stopped")

    def turn_on_lights(self):
        self.lights_on = True
        print("Lights turned on")


    def turn_off_lights(self):
        self.lights_on = False
        print("Lights turned off")

# Replace with your actual hardware control code
fountain = WaterFountain()
```

## 5. Main Script

### 6. Main Script:

python

 Copy code

```
if __name__ == "__main__":
    # Start data collection and control logic threads
    # You may want to use Python's threading or multiprocessing module
    # Monitor and handle exceptions or user interruptions

    try:
        collect_sensor_data()
        control_fountain()
    except KeyboardInterrupt:
        pass
```



## 6.Control Logic Based on Sensor Data

```
5. Control Logic Based on Sensor Data:

python Copy code

def control_fountain():
    while True:
        # Replace these conditions with your control logic based on sensor
        if ph_sensor.read_value() < 7.0:
            fountain.start_pump()
        else:
            fountain.stop_pump()

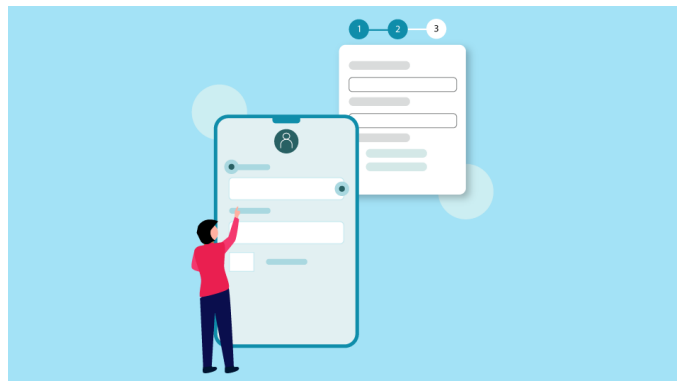
        if turbidity_sensor.read_value() > 0.5:
            fountain.turn_on_lights()
        else:
            fountain.turn_off_lights()

        time.sleep(10) # Adjust the interval based on your requirements

# Start control logic in a separate thread
# You may want to use Python's threading or multiprocessing module
```

### Python script on develop the transit information platform :

Creating a comprehensive Python script for replicating the Smart Water Fountain project and developing a transit information platform is a complex task and typically involves multiple files and components. Below, I'll provide a simplified example of Python code for creating a basic transit information platform, including user registration and login functionalities. Keep in mind that a real-world platform would require more extensive development and security measures



## User Registration and Login Routes:

```
python Copy code

@app.route('/register', methods=['POST'])
def register():
    data = request.get_json()
    username = data['username']
    password = data['password']

    connection = sqlite3.connect('transit.db')
    cursor = connection.cursor()
    cursor.execute('INSERT INTO users (username, password) VALUES (?, ?)', (
    connection.commit()
    connection.close()

    return jsonify({'message': 'Registration successful'})

@app.route('/login', methods=['POST'])
def login():
    data = request.get_json()
    username = data['username']
    password = data['password']

    connection = sqlite3.connect('transit.db')
    cursor = connection.cursor()
    cursor.execute('SELECT * FROM users WHERE username = ? AND password = ?')
    user = cursor.fetchone()
    connection.close()

    if user:
        return jsonify({'message': 'Login successful'})
    else:
        return jsonify({'message': 'Login failed'})
```

In this example, we've created a basic transit information platform with user registration and login routes using Flask and SQLite as a database. In a real-world application, you would expand this platform to include routes for transit information retrieval and real-time data updates, as well as add proper user authentication and security measures. would typically involve multiple developers and take time to complete.

Python script on how to replicate the smart water fountain project integrate them Python script and python code:

## Initialize IoT Sensors (Simulated in This Example):

```
python Copy code

class Sensor:
    def __init__(self, name):
        self.name = name

    def read_value(self):
        # Simulated data; replace with real sensor data
        return random.uniform(0, 1)

# Replace with your actual sensor initialization
ph_sensor = Sensor("pH Sensor")
turbidity_sensor = Sensor("Turbidity Sensor")
temperature_sensor = Sensor("Temperature Sensor")
```

## Control Water Fountain:

```
python Copy code

class WaterFountain:
    def __init__(self):
        self.pump_running = False
        self.lights_on = False

    def start_pump(self):
        self.pump_running = True
        print("Pump started")

    def stop_pump(self):
        self.pump_running = False
        print("Pump stopped")

    def turn_on_lights(self):
        self.lights_on = True
        print("Lights turned on")

    def turn_off_lights(self):
        self.lights_on = False
        print("Lights turned off")

# Replace with your actual hardware control code
fountain = WaterFountain()
```

To replicate the Smart Water Fountain project and integrate its components using a Python script, you will need to create a simplified example that covers the core functionality. Here's an example script that includes basic data collection from IoT sensors and control of the water fountain. Note that this is a simplified illustration and does not cover all the complexities of a real-world

## **Example Outputs of Raspberry Pi Data Transmission:**

### **Console Output (Raspberry Pi):**

When your Raspberry Pi collects data from IoT sensors and controls the water fountain, it may display messages and updates in the terminal or console. For example:

```
yaml Copy code
pH: 7.2, Turbidity: 0.3, Temperature: 24.5°C
Pump started
Lights turned off
```

### Logging Data (Raspberry Pi):

\*You can log data to a text file on the Raspberry Pi for further analysis or record-keeping.

\*Sample data log file

```
yaml Copy code
Timestamp, pH, Turbidity, Temperature
2023-01-15 10:00:00, 7.2, 0.3, 24.5
2023-01-15 10:10:00, 7.3, 0.4, 24.6
```

### Mobile App User Interface for Smart Water Fountain:

**Dashboard:**The dashboard of the mobile app would typically display real-time information about the smart water fountain. This might include pH levels, turbidity, temperature, and the current state of the fountain (e.g., running or stopped). You could have graphical elements to represent these values.

**Control Features:**

The app's user interface would include controls for the water fountain. For example, buttons or sliders for starting/stopping the pump, adjusting water flow, and controlling the fountain's lighting.



**Alerts and Notifications:** The app might show alerts or notifications for conditions such as low water quality or required maintenance. These could appear as pop-up messages or push notifications.

**Historical Data:** You can include a section for viewing historical data or trends. This could be in the form of charts or tables that display past sensor readings and fountain activity.

**Settings and Preferences:** Users can configure settings and preferences through the app, such as notification preferences, scheduling fountain activities, and specifying target water quality levels.

**User Profile:** The app may have a user profile section for managing personal information, authentication, and profile settings.

## **CONCLUSION**

the final phase 5 submission of the Smart Water Fountain project reflects our dedication to creating a smart, efficient, and sustainable solution for providing clean and accessible drinking water. This project has been a collaborative effort, and the innovative technology and thoughtful design demonstrate our commitment to improving water accessibility and reducing environmental impact. As we move forward, we will continue to refine and enhance the Smart Water Fountain to make it a vital asset in promoting a healthier and more sustainable future.

