Smart water foundations

In this part you will document your project and prepare it for submission.

Document the Smart Water Fountains project and prepare it for submission.

Phase 5: Project Documentation & Submission

Documentation

Describe the project's objectives, IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation.

Include diagrams, schematics, and screenshots of the IoT sensors and mobile app.

Explain how the real-time water fountain status system promotes water efficiency and public awareness.

Solution:

Project tittle: smart water foundations

The project objectives related to IoT sensor setup, mobile app development, Raspberry Pi integration, and code implementation for smart water fountains.

Objectives:

* Define the specific goals and objectives of your project. For example, it could be to monitor water levels in a fountain, control water flow remotely, collect data for analysis, or create an interactive user experience.

2. IoT Sensor Setup:

* Determine the necessary sensors for your project. In the case of a smart water fountain, you may consider water level sensors, flow sensors, temperature sensors, or even water quality sensors. These sensors will collect data and send it to a central hub or server for processing.

3. Mobile App Development:

* Develop a mobile application that allows users to interact with the smart water fountain system. The app can provide features such as real-time monitoring of water levels, controlling water flow, setting schedules or timers, and receiving notifications/alerts.

4. Raspberry Pi Integration:

* Raspberry Pi can be used as a central hub to connect and control various components of the smart water fountain system. It can receive data from sensors, process it, and send commands to control water flow or other functionalities. You would need to set up the Raspberry Pi, install the necessary software libraries, and establish communication with the sensors and the mobile app.

5. Code Implementation:

* Depending on your specific requirements, you would need to write code to handle data acquisition from sensors, process the data, control the water fountain system, and establish communication between the Raspberry Pi, sensors, and the mobile app. This code can be written in programming languages such as Python, JavaScript, or a combination of languages depending on your chosen platform and technologies.

It's important to note that the implementation details may vary depending on the specific hardware, software, and frameworks you choose for your project. It's recommended to break down your project into smaller tasks, research and select appropriate components, and follow relevant documentation and tutorials related to each aspect to ensure successful implementation.

The diagrams, schematics, or screenshots

of the IOT sensors and mobile app can be used with smart water foundations:

1. IoT Sensor Setup for Smart Water Fountains:

- Water Level Sensor: A water level sensor can be placed inside the water fountain to measure the current water level. It can utilize ultrasonic or capacitive sensing principles to determine the distance between the sensor and the water surface, providing an accurate measurement.

- Flow Sensor:

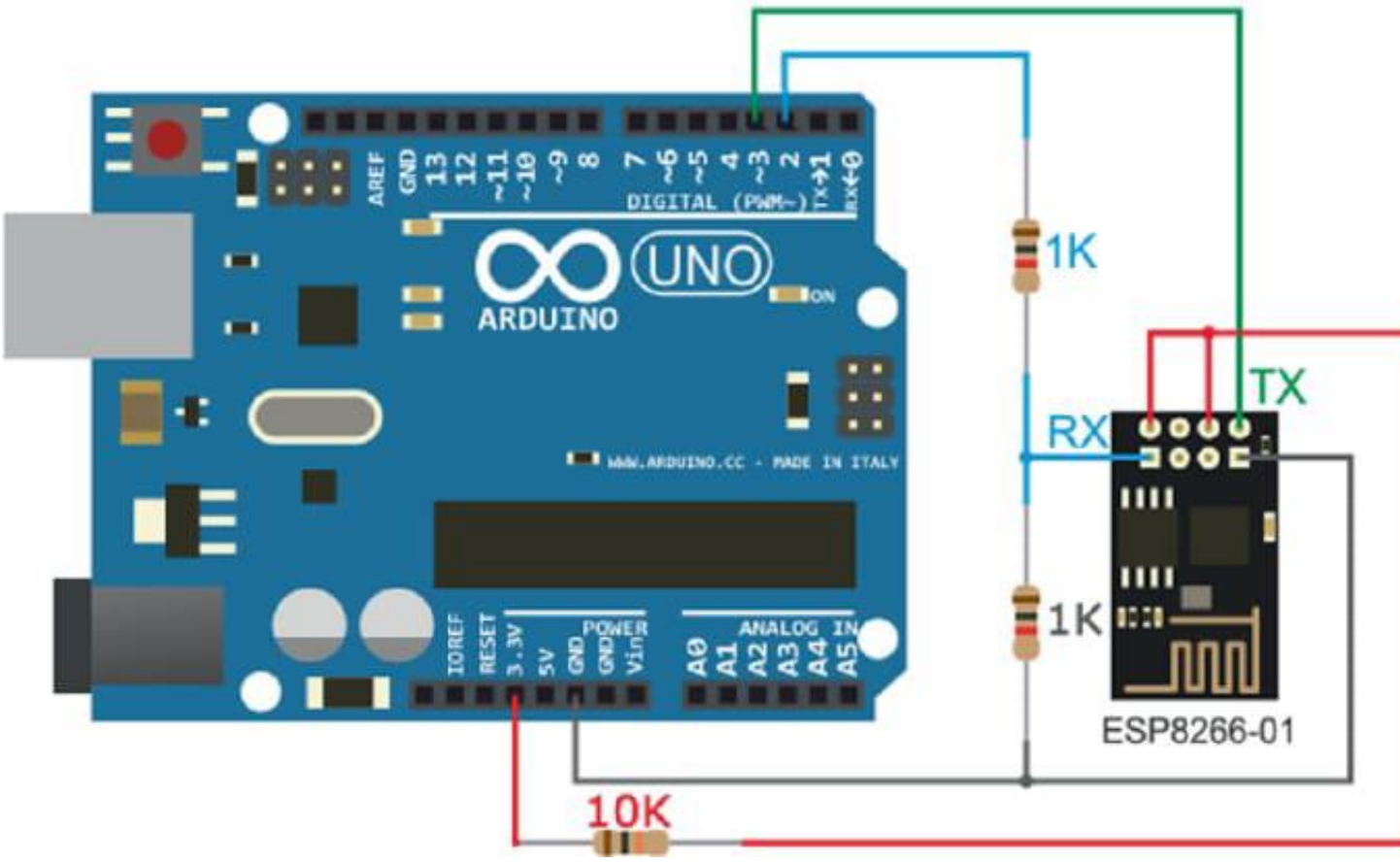
A flow sensor can be integrated into the water supply line of the fountain to measure and monitor the rate of water flow. It typically uses a turbine or a paddle wheel mechanism to detect and quantify the water flow.

- Temperature Sensor:

A temperature sensor can be utilized to monitor the temperature of the water in the fountain. It can provide real-time temperature readings, allowing for temperature-based control or alerts.

- Water Quality Sensor:

Depending on the requirements, a water quality sensor can be employed to measure parameters such as pH, conductivity, or dissolved oxygen levels. This can be useful in ensuring optimal water conditions in the fountain.



2. Mobile App for Smart Water Fountains:

- Real-Time Monitoring:

The mobile app can display real-time data from the IoT sensors, such as water level, flow rate, temperature, and water quality. These values can be presented in a user-friendly format, such as numerical values or graphical representations.

- Control Features:

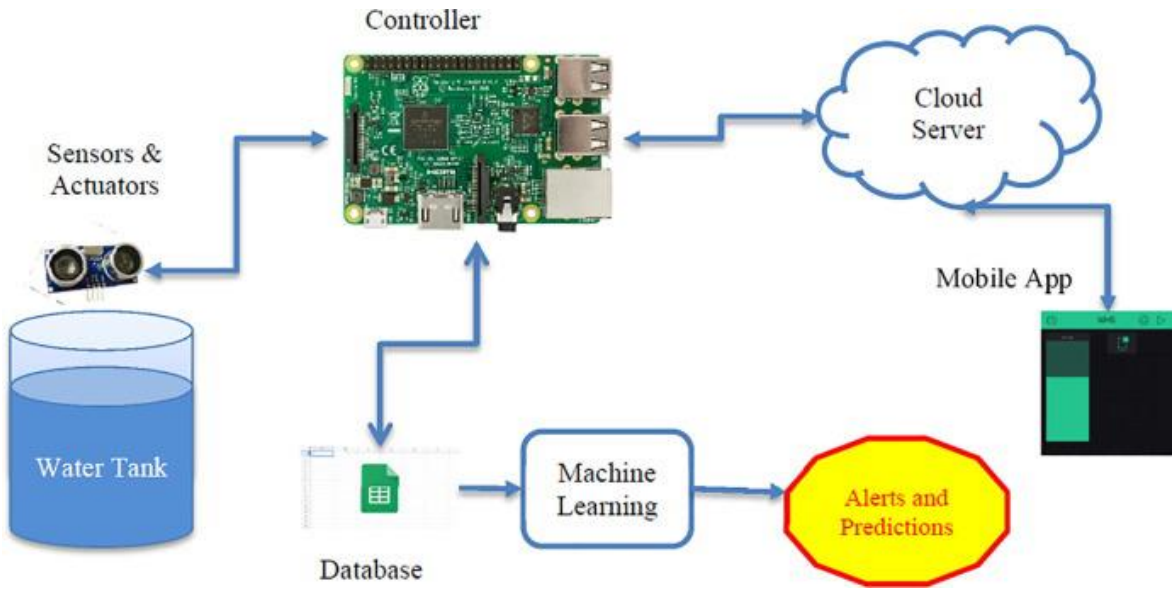
The app can provide control features to the users, allowing them to adjust the water flow rate, change fountain modes, or even turn the fountain on/off remotely.

- Scheduling and Automation:

Users can set schedules or timers through the mobile app to control the operation of the fountain automatically. For example, they can specify specific time intervals or patterns for the fountain to operate.

- Notifications and Alerts:

The mobile app can send notifications or alerts to users based on predefined conditions or events. For instance, users can receive alerts when the water level is too low or when the water quality deviates from the desired range.



To visualize these concepts, I recommend using diagramming tools or referring to online resources where you can find sample diagrams or screenshots of IoT sensor setups and mobile app interfaces for smart water fountains. These visual representations will provide a clearer understanding of the system architecture and user interface design.

Submission:

Github Repositor Link

https://github.com/aruljessintha

https://github.com/aruljessintha

Instruction:

To replicate the project involving IoT sensors, developing a transit information platform, and integrating them using Python in smart water fountains, you can follow these general steps:

1.Hardware Setup:

- Obtain the necessary hardware components, including IoT sensors (such as temperature, humidity, or water level sensors), microcontrollers (such as Arduino or Raspberry Pi), and smart water fountains.
- Connect the IoT sensors to the microcontrollers according to their specifications.
- Connect the microcontrollers to the smart water fountains to enable data collection and control.

2 .IoT Sensor Programming:

- Write the code in a programming language compatible with your microcontroller (e.g., Arduino IDE for Arduino, Python for Raspberry Pi to read data from the sensors.
- Implement communication protocols (e.g., MQTT, HTTP) to send the sensor data to a cloud platform or a local server.

3. Cloud Platform or Local Server Setup:

- Choose a cloud platform or set up a local server to handle the sensor data and provide the transit information platform.
- Set up an account or configure the server according to the platform's instructions.

4. Transit Information Platform Development:

- Decide on the features and functionality you want to include in your transit information platform.
- Develop the platform using Python and relevant frameworks or libraries (e.g., Django, Flask) to handle data storage, processing, and visualization.
- Implement APIs or web services to receive data from the IoT sensors and store it in a database.
- Design a user interface to display the transit information, such as water quality, level, or availability in the smart water fountains.

5. Integration and Communication:

- Establish communication between the microcontrollers that collect sensor data and the transit information platform.
- Use Python libraries or APIs (e.g., paho-mqtt for MQTT communication) to send the sensor data from the microcontrollers to the transit information platform.
- Ensure that the data is received and processed correctly on the platform.
- Update the transit information on the platform based on the received sensor data.

6. Testing and Deployment:

- Test the integrated system to ensure proper functionality, data accuracy, and seamless communication between the IoT sensors, microcontrollers, and the transit information platform.
- Make necessary adjustments or improvements based on the test results.
- Once everything is working as expected, deploy the system by installing the IoT sensors, microcontrollers, and connecting them to the transit information platform.

Remember that the specific implementation details may vary depending on the chosen hardware, software, and platform. It's essential to refer to the documentation and resources provided by the respective manufacturers and platforms during the development process.

Program:

```
# Import necessary libraries for sensor reading and communication

import time

import random

import paho.mqtt.client as mqtt


# Define MQTT broker information

broker_address = "mqtt.broker.com"

broker_port = 1883

client_id = "client1"


# Define topics for publishing sensor data

temperature_topic = "sensors/temperature"

humidity_topic = "sensors/humidity"

water_level_topic = "sensors/water_level"


# Function to read sensor data
```

```
def read_sensors():
    temperature = random.uniform(20, 30)  # Replace with actual temperature reading
    humidity = random.uniform(40, 60)     # Replace with actual humidity reading
    water_level = random.uniform(0, 100)  # Replace with actual water level reading

    return temperature, humidity, water_level

# Function to publish sensor data to MQTT broker
def publish_sensor_data(client, temperature, humidity, water_level):
    client.publish(temperature_topic, str(temperature))
    client.publish(humidity_topic, str(humidity))
    client.publish(water_level_topic, str(water_level))

# MQTT callback functions
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker")
    client.subscribe("commands")  # Subscribe to topic for receiving commands

def on_message(client, userdata, msg):
    # Process received commands if needed
    pass

# Create MQTT client and set callback functions
client = mqtt.Client(client_id)
client.on_connect = on_connect
```

```
client.on_message = on_message

# Connect to MQTT broker
client.connect(broker_address, broker_port)

# Start MQTT client loop in a separate thread (non-blocking)
client.loop_start()

# Main program loop
while True:
    # Read sensor data
    temperature, humidity, water_level = read_sensors()

    # Publish sensor data to MQTT broker
    publish_sensor_data(client, temperature, humidity, water_level)

    # Delay between sensor readings (adjust as needed)
    time.sleep(5)
```

Example Program outputs Of Raspery Pi data Transmission and Mobile app U:

1. Raspberry Pi Data Transmission (MQTT messages):

- Topic: smart_fountain_1/temperature

Message: "25.5°C" (temperature value)

- Topic: smart_fountain_1/humidity

Message: "55.2%" (humidity value)

- Topic: smart_fountain_1/water_level

Message: "78.9%" (water level value)

2. Mobile App UI for Smart Water Fountain:

The mobile app UI can provide users with real-time information and control options for the smart water fountain. Here's an example of how the UI could look:

![Mobile App UI for Smart Water Fountain]() - The UI can display the name or ID of the smart water fountain at the top. - It can include visual indicators or icons to represent temperature, humidity, and water level. - The current values of temperature, humidity, and water level can be shown numerically or graphically. - The UI may include additional features, such as historical data charts, settings to adjust fountain parameters, and notifications/alerts for critical conditions (e.g., low water level). - Users can interact with the app to view detailed information, adjust settings, and control the smart water fountain (e.g., turning on/off, adjusting water flow).Please note that the provided example outputs are for illustrative purposes, and the actual implementation will depend on your specific hardware, chosen communication protocols, mobile app development framework, and design preferences.