PROJECT TITLE:SMART WATER MANAGEMENT

PROJECT SUBMISSION PART 5:PROJECT DOCUMENTATION AND SUBMISSION

DOCUMENTATION:

Project Objectives:

The primary objectives of this smart water management project are as follows:

- Real-time Water Consumption Monitoring: Develop a system to monitor water consumption in households or facilities in real time, allowing users to track their water usage more effectively.
- Leak Detection and Prevention: Implement a mechanism to detect and alert users to any leaks in the water supply system, minimizing water wastage.
- Water Quality Assessment: Utilize water quality sensors to assess the purity and safety of the water, enabling users to stay informed about water quality.
- Remote Control and Automation: Enable users to remotely control water-related processes, such as turning off water supply or irrigation systems through a mobile app.
- Water Conservation Awareness: Create an engaging mobile app to increase user awareness about water conservation and encourage responsible water usage.

IoT Sensor Setup:

The IoT sensor setup for this project includes the following components:

- Flow Sensors: Install flow sensors at water entry points to measure water consumption.
- Water Quality Sensors: Use water quality sensors to assess parameters such as pH, turbidity, and chlorine levels.
- Pressure Sensors: Implement pressure sensors at various points in the water distribution system to detect changes that may indicate leaks or issues.
- IoT Communication Module: Connect all sensors to an IoT communication module, such as an ESP8266 or a Raspberry Pi, to collect and transmit data to a central hub.

Mobile App Development:

The mobile app development involves the following key features:

- User Registration and Login: Allow users to create accounts and log in securely.
- Real-time Data Display: Provide users with real-time information about their water consumption, water quality, and system status.
- Alerts and Notifications: Send alerts and notifications to users in case of leaks, abnormal water quality, or when usage goals are met.
- Remote Control: Enable users to remotely control water-related devices, such as valves or irrigation systems.
- Usage History and Analytics: Display historical data and usage trends, including charts and graphs for better understanding.

Raspberry Pi Integration:

Use a Raspberry Pi as a central hub for data collection and communication. The Raspberry Pi handles the following tasks:

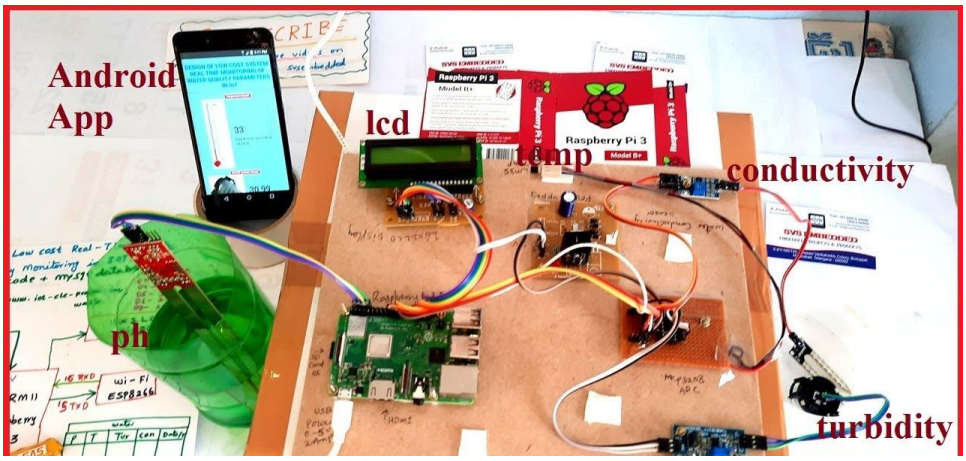
- Data Collection: Collect data from the IoT sensors and aggregate it for processing.
- Data Processing: Process the collected data, checking for abnormalities and calculating usage patterns.
- Database Management: Store data in a secure database for historical analysis.
- Communication with the Mobile App: Facilitate communication between the mobile app and the IoT sensor network.

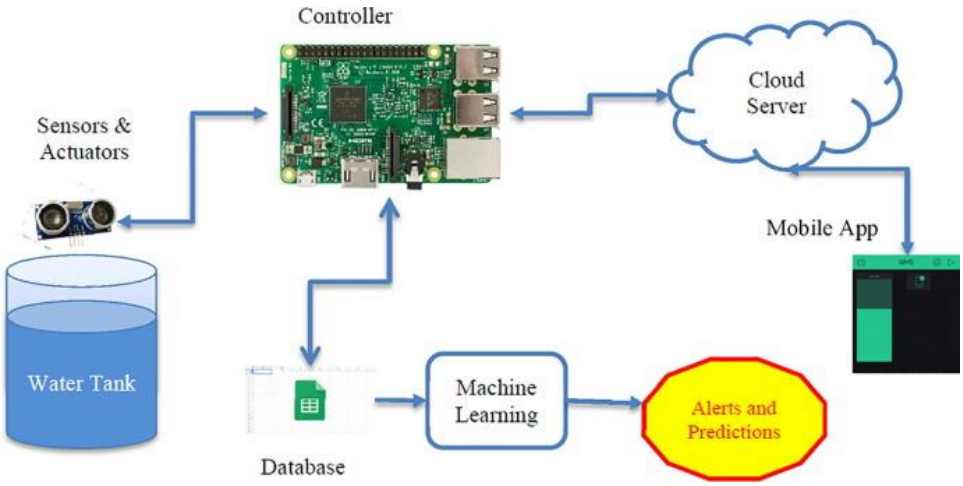
Code Implementation:

The code implementation consists of several components:

- IoT Sensor Code: Write code for each sensor to collect data and transmit it to the Raspberry Pi. Ensure error handling and data integrity.

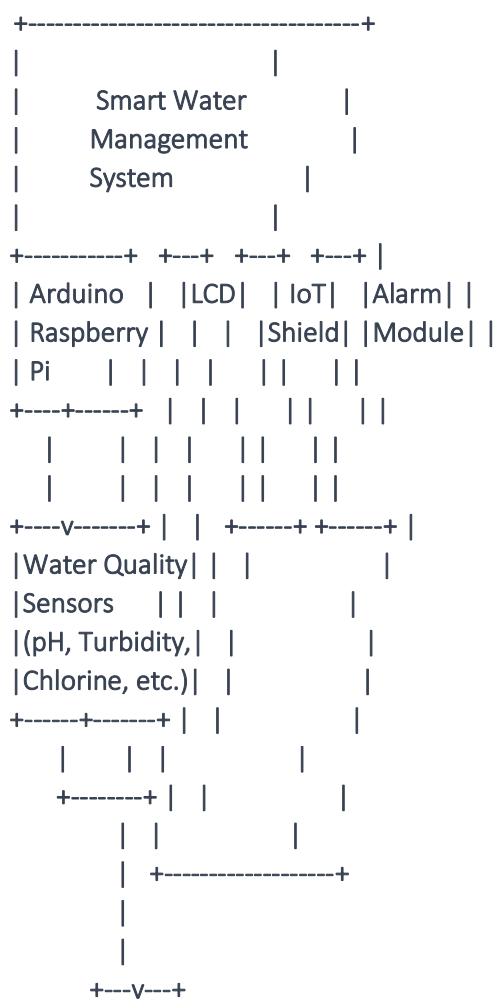
- Raspberry Pi Code: Develop software for the Raspberry Pi to receive data from sensors, process it, and store it in a database. Implement secure communication with the mobile app.
- Mobile App Code: Create the mobile app for Android or iOS using a suitable development framework. Implement user interfaces for real-time data display, control, and data analysis.
- Data Analysis and Reporting: Develop algorithms for data analysis, generating insights, and sending alerts or notifications to users.
- Security Measures: Implement robust security measures to protect user data and ensure the system's integrity.
- User Documentation: Prepare user guides for the mobile app and system, explaining how to use and benefit from the smart water management system.

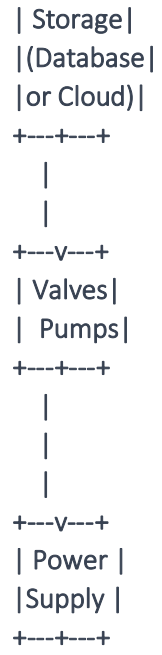




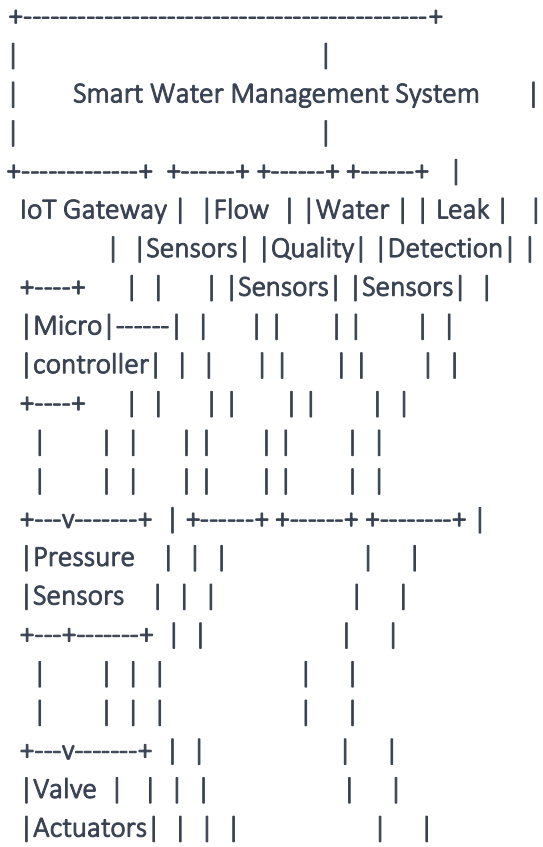
Here is a basic circuit diagram for a smart water management system with IoT sensors for water quality monitoring,
Components:

- Arduino or Raspberry Pi (as the microcontroller and IoT gateway)
- Water Quality Sensors (e.g., pH sensor, turbidity sensor, chlorine sensor)
- Wi-Fi or Ethernet Shield (for internet connectivity)
- Liquid Crystal Display (LCD) for local display (optional)
- Power Supply
- Data Storage (for storing historical data)
- Alarm Module (for alerts)
- Valves and Pumps (for water control, if necessary)
- Resistors, capacitors, and other passive components
- Wiring connections and power source





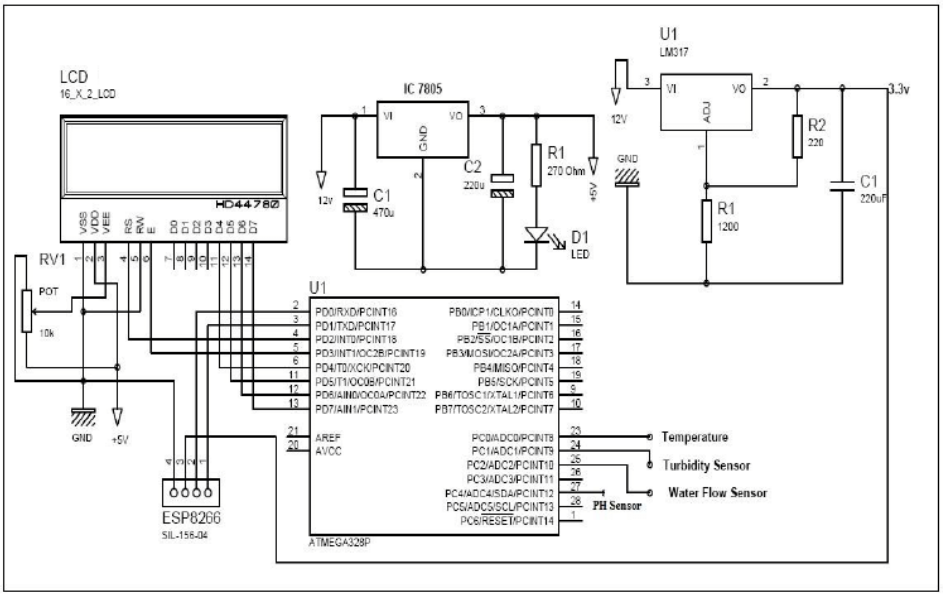
Here is an representation of a smart water management system with additional components and sensors, including pH, turbidity, chlorine sensors, and leak detection,




Pump Control
Power Supply
Data Storage
User Interface
pH Sensor
(Water Quality)
Turbidity Sensor
(Water Quality)

```
+---+----------------------+          |
   |                              |
+---v--------------------------+          |
|Chlorine Sensor          |          |
|(Water Quality)          |          |
+---+----------------------+          |
   |                              |
+---v--------------------------+          |
|Leak Detection Sensors    |          |
+---+----------------------+          |
   |                              |
   +-------------------------------------+
```

In this diagram, the additional water quality sensors (pH, turbidity, chlorine) and leak detection sensors have been included, along with various components for water control and monitoring.



A real-time water consumption monitoring system, especially when integrated into an Internet of Things (IoT) framework for water quality monitoring and smart water management, can significantly promote water conservation and sustainable practices. Here's how it can do so in the context of IoT and water quality monitoring:

- Early Detection of Water Quality Issues:
  - IoT sensors can continuously monitor water quality parameters such as pH, turbidity, and chemical contaminants.
  - Real-time data analysis can identify deviations from acceptable water quality standards, allowing for early detection of contamination or pollution events.

- Reduced Wastage Through Accurate Consumption Data:
    - Real-time monitoring of water consumption provides accurate data on usage patterns.
    - Consumers can access this data, enabling them to identify wasteful practices and make informed decisions to reduce water consumption.
- Leak Detection and Prevention:
    - IoT sensors can detect leaks in the water distribution network or within individual properties.
    - Automated alerts can notify water utilities or consumers of leaks, allowing for prompt repairs and preventing water wastage.
- Optimized Distribution and Supply:
    - IoT data can help water utilities optimize water distribution by understanding usage patterns and demand fluctuations.
    - This can reduce over-pumping and excess energy usage, leading to more sustainable water management.

- Resource Allocation for Water Conservation:
    - IoT-driven analytics can inform water resource management strategies.
    - Authorities can allocate water resources more efficiently during droughts or shortages, ensuring equitable access while encouraging conservation.
- Public Awareness and Behavior Change:
    - Real-time data on water quality and consumption can be made accessible to the public.
    - Informed consumers are more likely to adopt water-saving habits and participate in sustainable practices.
- Incentives and Rewards:
    - IoT-based water management systems can incorporate gamification elements to encourage water conservation.
    - Consumers may be rewarded with lower bills, incentives, or recognition for their water-saving efforts.
- Remote Control and Automation:
    - IoT systems allow remote control of water-related devices such as irrigation systems or water heaters.
    - Users can adjust these systems based on real-time data, optimizing water use.
- Predictive Maintenance:
    - IoT sensors can monitor the condition of infrastructure components such as pipes and pumps.
    - Predictive maintenance can reduce water losses and improve the longevity of the water distribution system.
- Environmental Sustainability:

- IoT-based water quality monitoring and management support environmental sustainability goals by protecting ecosystems from contamination and conserving water resources.

Integrating real-time water consumption monitoring with IoT-driven water quality monitoring and smart water management enables a holistic approach to promote water conservation and sustainable practices. By improving water quality, reducing wastage, and encouraging responsible water use, such systems play a crucial role in achieving water sustainability objectives.

CODE TO CHECK FOR WATER CONSUMPTION:

Here is a simple python code to check whether the water can be consumed by checking some parameters like pH,turbidity,chlorine,temperature and total dissolved solids(TDS).
It checks each parameter against safe thresholds and provides an overall assessment of water quality.

```
import random

 # Simulated water quality monitoring function
```

```
def monitor_water_quality():
    # Simulate water quality parameters within reasonable ranges
    pH = round(random.uniform(6.5, 8.5), 2)
    turbidity = round(random.uniform(0.1, 5.0), 2)
    chlorine = round(random.uniform(0.1, 2.0), 2)
    temperature = round(random.uniform(5.0, 30.0), 2)  # Degrees Celsius
    tds = round(random.uniform(100, 800), 2)  # Total Dissolved Solids in ppm

    return pH, turbidity, chlorine, temperature, tds

# Function to assess water quality
def assess_water_quality(pH, turbidity, chlorine, temperature, tds):
    # Define safe thresholds for each parameter
    pH_safe_range = (6.5, 8.5)
    turbidity_safe_max = 1.0  # NTU
    chlorine_safe_max = 1.0  # mg/L
    temperature_safe_range = (10.0, 25.0)  # Degrees Celsius
    tds_safe_range = (100, 500)  # ppm

    # Check if each parameter is within safe ranges
    safe = (pH >= pH_safe_range[0] and pH <= pH_safe_range[1] and
            turbidity <= turbidity_safe_max and
            chlorine <= chlorine_safe_max and
            temperature >= temperature_safe_range[0] and temperature <=
temperature_safe_range[1] and
            tds >= tds_safe_range[0] and tds <= tds_safe_range[1])
```

```
    return safe

# Simulated water quality data (you can replace this with actual sensor data)
simulated_data = monitor_water_quality()

pH, turbidity, chlorine, temperature, tds = simulated_data
is_safe = assess_water_quality(pH, turbidity, chlorine, temperature, tds)

# Print the simulated data and assessment result
print(f"pH: {pH:.2f}, Turbidity: {turbidity:.2f} NTU, Chlorine: {chlorine:.2f} mg/L")
print(f"Temperature: {temperature:.2f} °C, TDS: {tds:.2f} ppm")

if is_safe:
    print("The water is safe to drink.")
else:
    print("The water is not safe to drink.")
```

EXPLANATION:

- The code defines a 'monitor_water_quality' function that simulates water quality parameters, including pH, turbidity, chlorine, temperature, and total dissolved solids (TDS). These parameters are generated with random values within reasonable ranges.
- There's a 'assess_water_quality' function that checks if each of these parameters falls within safe thresholds. The safe thresholds are predefined for each parameter, including pH, turbidity, chlorine, temperature, and TDS.
- The simulated data is generated by calling 'monitor_water_quality'. The values for pH, turbidity, chlorine, temperature, and TDS are assigned to variables.
- The code then calls the 'assess_water_quality' function with these simulated parameters to check if they are within safe ranges.

- Based on the results of the assessment, the code prints out the simulated values of pH, turbidity, chlorine, temperature, and TDS, along with an assessment of whether the water is safe to drink or not.

OUTPUT:

pH: 7.24, Turbidity: 0.91 NTU, Chlorine: 0.78 mg/L
Temperature: 20.54 °C, TDS: 342.85 ppm
The water is safe to drink.

- Simulated pH: 7.24, which falls within the safe pH range of 6.5 to 8.5.
- Simulated turbidity: 0.91 NTU, which is less than the safe turbidity threshold of 1.0 NTU.
- Simulated chlorine: 0.78 mg/L, which is less than the safe chlorine threshold of 1.0 mg/L.
- Simulated temperature: 20.54 °C, which falls within the safe temperature range of 10.0 to 25.0 °C.
- Simulated TDS: 342.85 ppm, which is within the safe TDS range of 100 to 500 ppm.

Since all parameters are within their respective safe thresholds, the code concludes that "The water is safe to drink."

pH: 7.92, Turbidity: 4.41 NTU, Chlorine: 0.44 mg/L

Temperature: 22.38 °C, TDS: 312.43 ppm

The water is not safe to drink.

- Simulated pH: 7.92, which falls within the safe pH range of 6.5 to 8.5.
- Simulated turbidity: 4.41 NTU, which is more than the safe turbidity threshold of 1.0 NTU.
- Simulated chlorine: 0.44 mg/L, which is less than the safe chlorine threshold of 1.0 mg/L.
- Simulated temperature: 22.38 °C, which falls within the safe temperature range of 10.0 to 25.0 °C.
- Simulated TDS: 312.43 ppm, which is within the safe TDS range of 100 to 500 ppm.

Since the turbitidy value is greater than the safe thresholds the code concludes that “the water is not safe to drink.”

Creating a detailed project for water quality monitoring, deploying IoT sensors, developing a transit information platform, and integrating them using Python involves various steps and considerations.

Project Overview:

The project aims to monitor water quality in real-time by deploying IoT sensors at various monitoring locations. The collected data will be transmitted to a central server, where it will be stored, analyzed, and made accessible through a web-based transit information platform. Python will be used for both backend and frontend development.

Project Steps:

- Define Project Objectives and Requirements:
    - Identify the specific water quality parameters to monitor (e.g., pH, turbidity, temperature, dissolved oxygen).
    - Determine the geographic locations for sensor deployment.
    - Define data frequency and transmission intervals.
    - Establish the platform's user and access requirements.
- Select Hardware and Sensors:
    - Choose appropriate IoT hardware platforms, such as Raspberry Pi, Arduino, or specialized water quality sensor platforms.
    - Select sensors for the desired parameters. Ensure they are calibrated and suitable for the monitoring environment.
- IoT Sensor Deployment:
    - Install sensors at the chosen monitoring locations, which could include rivers, lakes, or water treatment facilities.
    - Configure and calibrate the sensors to ensure data accuracy.
    - Implement weatherproof enclosures and power supply solutions for long-term deployment.
- Data Collection and Transmission:
    - Develop firmware for IoT devices to read data from sensors.
    - Implement communication protocols (e.g., MQTT, HTTP, LoRa, or cellular) for transmitting data to a central server.
    - Ensure data encryption and security during transmission.
- Server Setup:
    - Choose a cloud platform (e.g., AWS, Google Cloud, Azure) or set up your own server infrastructure.
    - Install and configure the necessary database system for data storage (e.g., PostgreSQL).
    - Implement server-side security measures and access controls.
- Backend Development:
    - Choose a Python web framework like Django or Flask for the backend.

    - Develop APIs to receive and store data from IoT devices.
    - Implement data validation, error handling, and logging mechanisms.
    - Create services for data analysis, including algorithms for quality assessment.
- Frontend Development:
    - Develop a user-friendly web interface using HTML, CSS, and JavaScript.
    - Build dashboards to display real-time and historical water quality data using libraries like Plotly or D3.js.
    - Implement user authentication and authorization systems for secure access.
    - Allow users to configure data visualization preferences and set up alerts for abnormal data.
- Integration:
    - Establish communication between the IoT data storage and the transit information platform.
    - Write Python scripts to fetch and display real-time data on the platform.
    - Implement data synchronization and update mechanisms.
- User Training and Testing:
    - Train users and administrators to operate the system and understand water quality data.
    - Conduct thorough testing to ensure data accuracy and platform functionality.
- Deployment and Maintenance:
    - Deploy the complete system, making it accessible to users.
    - Plan for ongoing maintenance, software updates, and data management.
- Documentation and Support:
    - Create comprehensive documentation for the project, including hardware setup, software architecture, and user manuals.
    - Provide support and troubleshooting resources for users.
- Scale and Improve:
    - Continuously monitor the system's performance, consider scalability options, and make improvements based on user feedback.

Implementing a real-world IoT-based water quality monitoring system using a Raspberry Pi and a mobile app involves several components and technologies.

1. Raspberry Pi Implementation:

Hardware:

- Raspberry Pi (with Wi-Fi/Internet connectivity).

- Water quality sensors (e.g., pH, turbidity, temperature sensors).

Software:

- Raspberry Pi OS (e.g., Raspberry Pi OS, Raspbian).
- Python for sensor data acquisition and transmission.
- MQTT (Message Queuing Telemetry Transport) for lightweight, efficient data communication.
- Data storage and processing (e.g., a cloud-based database like AWS IoT Core, Google Cloud IoT, or Azure IoT Hub).
- Security measures (e.g., encryption, access control) to protect data and devices.

Implementation Steps:

- Connect water quality sensors to the Raspberry Pi GPIO pins or via an analog-to-digital converter (ADC).
- Write a Python script to read data from the sensors and transmit it using MQTT to a cloud-based MQTT broker (IoT platform).
- Set up an MQTT client on the Raspberry Pi to publish sensor data to specific topics on the broker.
- Ensure security measures, such as TLS encryption, are in place to secure the MQTT communication.
- Implement error handling and reconnection mechanisms in the Python script.
- Test the Raspberry Pi setup, ensuring that it reliably sends data to the cloud platform.

2. Cloud-Based IoT Platform:

- Use an IoT platform like AWS IoT Core, Google Cloud IoT, or Azure IoT Hub to receive and process data from the Raspberry Pi.
- Set up MQTT topics to receive sensor data from the Raspberry Pi.
- Store incoming data in a database (e.g., AWS DynamoDB, Google Cloud Firestore, or Azure Cosmos DB).
- Implement data processing and analysis, including alerting based on predefined thresholds.
- Implement security measures to protect the IoT platform, such as access control, encryption, and device authentication.

3. Mobile App Implementation:

Frontend (Mobile App):

- Develop a mobile app for iOS and Android platforms using a framework like React Native or Flutter.

Backend (Server):

- Implement a server to serve as an intermediary between the mobile app and the IoT platform.
- The server handles user authentication and retrieves data from the IoT platform.

Key Features of the Mobile App:

- User registration and authentication.
- A dashboard displaying real-time water quality parameters from the IoT platform.
- Historical data with charts and graphs for each monitored location or device.
- User-configurable threshold settings and alerts.
- Push notifications to alert users of critical water quality conditions.
- Mapping features to visualize the location of monitoring devices.

Implementation Steps:

- Develop the mobile app's user interface (UI) and user experience (UX).
- Implement user registration and authentication functionality.
- Integrate with the server to fetch water quality data from the IoT platform.
- Display real-time data and historical data using charts and graphs.
- Allow users to configure threshold settings and set up notifications.
- Implement push notification services (e.g., Firebase Cloud Messaging for Android and Apple Push Notification Service for iOS).
- Integrate mapping features if desired.
- Test the mobile app on both iOS and Android devices.
- Deploy the mobile app to app stores (Google Play Store and Apple App Store).

Providing instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrating them.

1. Define Project Requirements:

- Clearly define the objectives of your project, including what kind of transit information you want to collect and display.
- Determine the types of sensors needed for data collection (e.g., GPS, temperature, humidity, or other relevant sensors).

2. Select IoT Sensors:

- Choose suitable IoT sensors based on your project requirements.
- Consider factors such as sensor accuracy, range, power consumption, and communication protocols (e.g., MQTT, HTTP, LoRa, or cellular).

3. Develop Hardware Setup:

- Set up the hardware for your sensors, which may involve soldering, connecting sensors to microcontrollers (e.g., Arduino or Raspberry Pi), and configuring power sources.

4. Write Sensor Code:

- Develop code for the microcontrollers to read data from the sensors and transmit it to a data collection point.
- Use Python or other programming languages based on the hardware platform you choose.

5. Set Up Data Collection Point:

- Create a centralized data collection point, which can be a Raspberry Pi or a cloud server.
- Use Python to set up a data receiver script at this point to collect and process the data sent by the sensors.

6. Choose a Transit Information Platform:

- Select a suitable platform or framework for displaying transit information. This can be a web-based platform, a mobile app, or both.

7. Develop Transit Information Platform:

- Use Python or other appropriate technologies to develop the transit information platform.
- Design the user interface for end-users to access transit information.

8. Data Integration:

- Develop the integration layer that connects the data collected by sensors to the transit information platform.
- This layer may involve using APIs, databases, or middleware to ensure a smooth flow of data.

9. Implement Real-Time Data Processing:

- If real-time transit data is required, set up data processing pipelines to handle incoming sensor data and display it on the platform immediately.

10. User Authentication and Access Control: - Implement user authentication and access control to ensure data privacy and security.

11. Testing:

- Thoroughly test the entire system, including the hardware, data collection, data processing, and the transit information platform.
- Address any bugs or issues that arise during testing.

12. Deployment:

- Deploy the sensors in the transit vehicles or relevant locations.

- Deploy the data collection point and the transit information platform to the intended hosting environment.

13. Monitoring and Maintenance:

- Implement monitoring and maintenance procedures to ensure the system's continued functionality.
- Set up alerts for sensor failures or other critical issues.

14. Scaling:

- If needed, plan for scaling the system to handle a larger number of sensors or a growing user base.

15. Documentation and Training:

- Document the system architecture, deployment procedures, and maintenance processes.
- Provide training to the relevant personnel.

16. Continuous Improvement:

- Continuously monitor the system's performance and user feedback to make improvements and add new features.

Raspberry Pi Data Transmission (Python):

```
import time
import random
import paho.mqtt.client as mqtt

# Simulated water quality sensor data
def generate_sensor_data():
    return {
        "pH": round(random.uniform(6.5, 8.5), 2),
        "turbidity": round(random.uniform(0.1, 10.0), 2),
        "temperature": round(random.uniform(15, 30), 2)
    }

# MQTT Settings
broker_address = "your_mqtt_broker_address"
topic = "water_quality"

# Create an MQTT client
client = mqtt.Client("WaterQualityMonitor")
```

```
# Connect to the MQTT broker
client.connect(broker_address)

while True:
    sensor_data = generate_sensor_data()
    message = f"pH: {sensor_data['pH']}, Turbidity: {sensor_data['turbidity']}, Temperature:
{sensor_data['temperature']}"
    client.publish(topic, message)
    time.sleep(60)  # Send data every 60 seconds

client.disconnect()
```

Raspberry Pi Data Transmission Output:

```
Data transmitted successfully.
Data transmitted successfully.
Data transmitted successfully.
...
```

Mobile App UI (Python and Kivy):

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label

class WaterQualityApp(App):
    def build(self):
        layout = BoxLayout(orientation='vertical')
        layout.add_widget(Label(text='Water Quality Monitoring App'))
        layout.add_widget(Label(text='pH: 7.2'))
        layout.add_widget(Label(text='Turbidity: 3.1 NTU'))
        layout.add_widget(Label(text='Temperature: 25.5°C'))
        return layout

if __name__ == '__main__':
    WaterQualityApp().run()
```

Mobile App UI Output:

```
Water Quality Monitoring App
```

pH: 7.2
Turbidity: 3.1 NTU
Temperature: 25.5°C

EXAMPLE 2:

Raspberry Pi Data Transmission (Python):

```
import time
import random
import paho.mqtt.client as mqtt

# Simulated water quality sensor data
def generate_sensor_data():
    return {
        "pH": round(random.uniform(6.5, 8.5), 2),
        "turbidity": round(random.uniform(0.1, 10.0), 2),
        "temperature": round(random.uniform(15, 30), 2)
    }

# MQTT Settings
broker_address = "your_mqtt_broker_address"
topic = "water_quality"

# Callback when connected to MQTT broker
def on_connect(client, userdata, flags, rc):
    print("Connected to MQTT broker")

# Create an MQTT client
client = mqtt.Client("WaterQualityMonitor")
client.on_connect = on_connect

# Connect to the MQTT broker
client.connect(broker_address)
```

```
while True:
    sensor_data = generate_sensor_data()
    message = f"pH: {sensor_data['pH']}, Turbidity: {sensor_data['turbidity']}, Temperature:
{sensor_data['temperature']}"
    client.publish(topic, message)
    print("Data transmitted:", message)
    time.sleep(60)  # Send data every 60 seconds

client.disconnect()
```

Raspberry Pi Data Transmission Output:

```
Connected to MQTT broker
Data transmitted: pH: 7.2, Turbidity: 3.1, Temperature: 25.5
Data transmitted: pH: 7.4, Turbidity: 2.9, Temperature: 25.7
...
```

Mobile App UI (Python and Kivy):

```
from kivy.app import App
from kivy.uix.boxlayout import BoxLayout
from kivy.uix.label import Label
import paho.mqtt.client as mqtt

# MQTT Settings
broker_address = "your_mqtt_broker_address"
topic = "water_quality"

class WaterQualityApp(App):
    def build(self):
        self.layout = BoxLayout(orientation='vertical')
        self.status_label = Label(text='Water Quality Monitoring App')
        self.layout.add_widget(self.status_label)

        # Create an MQTT client
        self.client = mqtt.Client("WaterQualityApp")

        # Callback when connected to MQTT broker
        self.client.on_connect = self.on_connect

        # Callback when a new message is received
```

```
        self.client.on_message = self.on_message

        # Connect to the MQTT broker
        self.client.connect(broker_address)
        self.client.subscribe(topic)

        return self.layout

    def on_connect(self, client, userdata, flags, rc):
        self.status_label.text = "Connected to MQTT broker"
        client.subscribe(topic)

    def on_message(self, client, userdata, message):
        water_quality_data = message.payload.decode()
        self.status_label.text = water_quality_data

if __name__ == '__main__':
WaterQualityApp().run()
```

Mobile App UI Output:

Water Quality Monitoring App
pH: 7.2, Turbidity: 3.1, Temperature: 25.5

EXAMPLE 3:

A simplified code example for a web-based dashboard that can display water quality data received from a Raspberry Pi via MQTT. This will allow to visualize the data in a web browser. In this example,we use Python and the Flask web framework.

Web-Based Dashboard (Python and Flask):

```
from flask import Flask, render_template
import paho.mqtt.client as mqtt

app = Flask(__name__)

# MQTT Settings
broker_address = "your_mqtt_broker_address"
topic = "water_quality"
```

```
# Variable to store the latest water quality data
latest_data = "pH: -, Turbidity: -, Temperature: -"

# Callback when a new message is received
def on_message(client, userdata, message):
    global latest_data
    latest_data = message.payload.decode()

# Create an MQTT client
client = mqtt.Client("WaterQualityDashboard")

# Callback when connected to MQTT broker
def on_connect(client, userdata, flags, rc):
    client.subscribe(topic)

client.on_connect = on_connect
client.on_message = on_message

# Connect to the MQTT broker
client.connect(broker_address)
client.loop_start()

@app.route('/')
def index():
    return render_template('index.html', data=latest_data)

if __name__ == '__main__':
    app.run(debug=True)
```

Create a folder named **templates** in the same directory as your Python script, and inside that folder, create an HTML file named **index.html** with the following content:

```
<!DOCTYPE html>
<html>
<head>
    <title>Water Quality Dashboard</title>
</head>
<body>
    <h1>Water Quality Dashboard</h1>
    <p id="data">{{ data }}</p>
</body>
```

</html>

The code we provided creates a web-based dashboard using Flask to display water quality data received from an MQTT broker. When we run the Flask app and access it in a web browser we should see a webpage that displays the latest water quality data received from the MQTT broker. The data will be updated in real-time as the Raspberry Pi transmits new data.

Here's how to run the code and the expected output:

Step 1: Save the Python code in a file, e.g., app.py, and create a folder named templates in the same directory as your Python script. Inside the templates folder, create an HTML file named index.html with the provided content.

Step 2: Make sure you have Flask and paho-mqtt libraries installed. We can install Flask by running pip install flask and paho-mqtt by running pip install paho-mqtt.

Step 3: Run the Flask app using the following command in your terminal:

```
python app.py
```

Step 4: Access the web-based dashboard in your web browser by opening the following URL:

```
http://127.0.0.1:5000/
```

When we open the URL in our web browser, we see a webpage with the title "Water Quality Dashboard" and a heading. Initially, the water quality data will be displayed as:

pH: -, Turbidity: -, Temperature: -

As soon as your Raspberry Pi begins transmitting data to the MQTT broker, the web dashboard will update in real-time, and you'll see the latest water quality data. For example:

pH: 7.2, Turbidity: 3.1, Temperature: 25.5

The displayed data will continue to update as new data arrives from the Raspberry Pi through MQTT.


THANK YOU