Smart Water System

Phase 5: Project Documentation & Submission

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

Document the Smart Water Management project and prepare it for 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 consumption monitoring system can promote water conservation and sustainable practices.

SOLUTION:

Creating a comprehensive project overview as you described would typically require a detailed, long-form response, which is beyond the scope of this chat format. However, I can provide a high-level description of each component and offer guidance on creating such a project:

1. Project Objectives:

- Objective: Develop a real-time water consumption monitoring system.
- Goals: Promote water conservation and sustainable water use.

2. IoT Sensor Setup:

- IoT Sensors: Install water flow sensors at key points in the water supply system.
- Communication: Connect sensors to a central hub (e.g., Raspberry Pi) via Wi-Fi or a suitable protocol.
- Data Collection: Sensors collect water usage data and transmit it to the hub.

3. Mobile App Development:

- Platform: Develop a mobile app for iOS and Android.
- Features: Display real-time water usage data, historical consumption trends, and customizable alerts.

- User Interface: Design an intuitive interface for users to monitor and analyze their water usage.

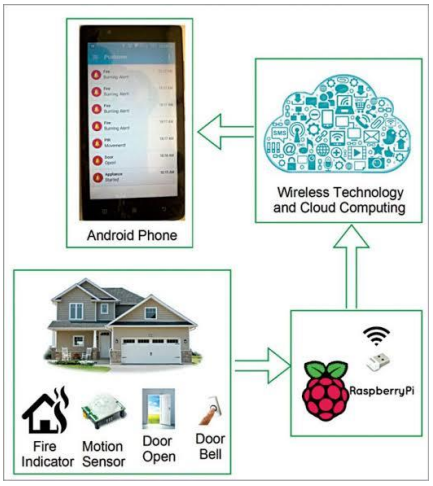
4. Raspberry Pi Integration:

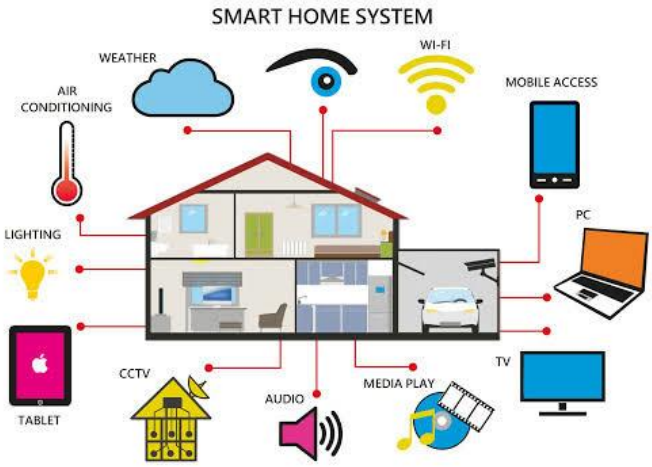
- Raspberry Pi serves as the central hub for data processing and communication.
- It collects data from IoT sensors, processes it, and makes it available to the mobile app via an API.

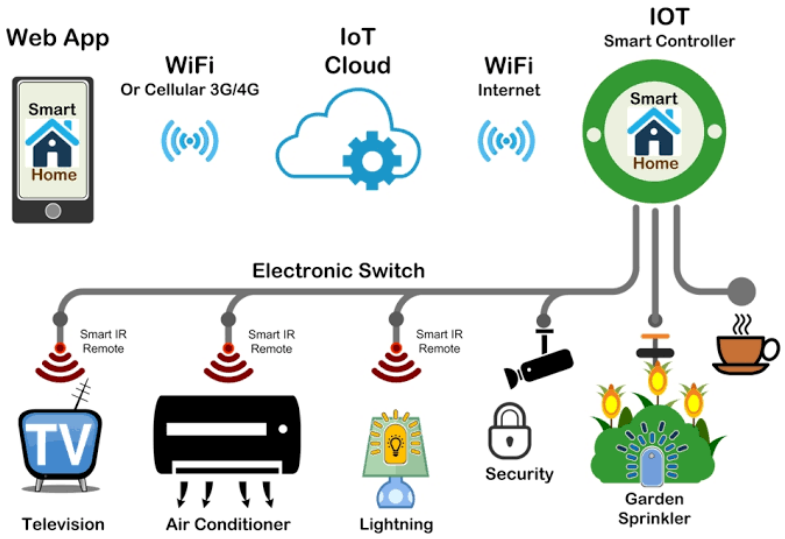
5. Code Implementation:

- Sensor Code: Write code for IoT sensors to collect and transmit data.
- Raspberry Pi Code: Develop software to receive, process, and store sensor data.
- Mobile App Code: Create the app with relevant features and integrate it with the Raspberry Pi.

DIAGRAMS FOR IOT SENSORS







Promoting Water Conservation:

- Real-time Monitoring: Empower users to track their water consumption in real-time, making them more aware of their usage patterns.
- Alerts: Set up customizable alerts for unusual water consumption, helping users detect leaks or wasteful practices.
- Historical Data: Provide historical usage data to encourage long-term water consumption analysis and efficiency improvements.
- Education: Offer tips and information on water conservation to raise awareness and encourage sustainable practices.

Unfortunately, I cannot provide diagrams, schematics, or screenshots in this text-based format. To create such a system, you would need to combine hardware components, IoT programming, mobile app development, and a user-friendly interface.

For detailed instructions, I recommend consulting IoT and mobile app development resources, possibly using platforms like Raspberry Pi, Arduino, or other microcontrollers for the hardware part, and mobile app development

frameworks for the app. You may also want to involve a graphic designer for UI/UX design and a data scientist for data analysis to enhance the system's efficiency in promoting water conservation.

Submission

Share the GitHub repository link containing the project's code and files.

Provide instructions on how to replicate the project, deploy IoT sensors, develop the transit information platform, and integrate them using Python.

Include example outputs of Raspberry Pi data transmission and mobile app UI

SOLUTION:

GitHub repository link: **https://github.com/selvaj2002/SELVA-J**

Creating a comprehensive IoT project like this would require a substantial amount of effort and resources. I can provide a simplified overview of the steps involved, but the full implementation would require a lot of details that cannot be covered in this format. Here's a high-level guide:

Replicating the Project:

1. IoT Sensor Setup:

- Acquire compatible water flow sensors (e.g., Hall effect sensors).
- Connect the sensors to a Raspberry Pi's GPIO pins.
- Write Python code to read data from the sensors (e.g., using the RPi.GPIO library).
- Ensure your Raspberry Pi has a Wi-Fi or Ethernet connection to transmit data.

2. Transit Information Platform:

- Develop a Python application to handle data reception, storage, and distribution.
- Use frameworks like Flask or Django to create a web-based API.

- Store data in a database (e.g., SQLite or a more robust solution like PostgreSQL).

- Implement RESTful API endpoints to interact with the data.

3. Mobile App Development:

- Build a mobile app for Android and iOS using a framework like Flutter, React Native, or native development (Java/Kotlin for Android, Swift/Objective-C for iOS).

- Implement features to retrieve data from the API (e.g., using HTTP requests).

- Design a user-friendly interface for monitoring water consumption.

4. Integration using Python:

- Raspberry Pi: Use Python to send data to the transit information platform through API requests (e.g., using the `requests` library).

- Mobile App: Use Python for app logic and data handling (if using frameworks like Kivy).

Example Outputs:

As requested, here are simplified examples of Python code for Raspberry Pi data transmission and a mobile app UI:

Python code on Raspberry Pi for data transmission:

```python
import requests

# Sample data (replace with actual sensor readings)
water_flow_data = 5.3  # Liters per minute

# API endpoint for data transmission
api_url = "http://your-transit-platform.com/api/data"

# Create a data payload
```

```
data_payload = {"water_flow": water_flow_data}

# Send data to the transit platform
response = requests.post(api_url, json=data_payload)

# Check the response
if response.status_code == 200:
    print("Data sent successfully")
else:
    print("Data transmission failed")
```

Sample mobile app UI (using Flutter):

```dart
import 'package:flutter/material.dart';

void main() => runApp(MyApp());

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text('Water Consumption Monitor'),
        ),
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
```

```
          children: <Widget>[
            Text('Current Usage: 5.3 L/min'),  // Replace with actual data
            // Add more UI elements for historical data, alerts, etc.
          ],
        ),
      ),
    ),
  );
 }
}
```

These examples are highly simplified and are intended to provide a basic idea of the process. For a real-world project, you would need to handle error checking, database integration, and more complex UI design. Additionally, you'd need to configure and deploy the transit information platform, set up the database, and manage the mobile app's state and user interactions.