**IBM NAAN MUDHALVAN**

**SKILL UP**

**PROJECT TITLE: SMART WATER SYSTEM**

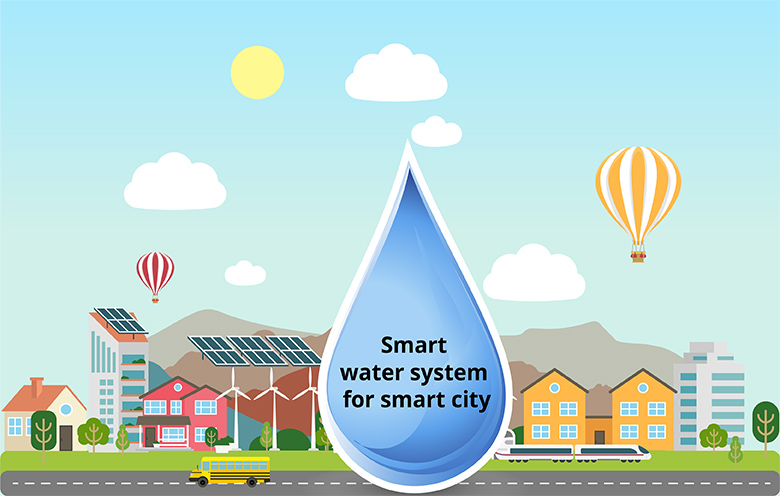
**COLLEGE: PERI INSTITUTE OF TECHNOLOGY**

**DEPT: ELECTRONICS AND COMMUNICATION ENGINEERING**

**DOMAIN: INTERNET OF THINGS (IOT)**

**Submitted By**

**VIJAYA SRI Y (au411521106060)**



**PHASE-3:**

* 1. **Introduction:**
* Currently drinking water is very prized for all the humans. In recent times water levels are very low and water in the lakes are going down. So its too important to find the solution for water monitoring & control system. IoT is a solution. In recent days, development in computing and electronics technologies have triggered Internet of Things technology.This paper present a low cost water monitoring system, which is a solution for the water wastage and water quality. Microcontrollers and sensors are used for that system. Ultrasonic Sensor is used to measuring water level. The other parameters like pH, TDS, and Turbidity of the water can be calculated using different corresponding sensors. This system use the flow sensor which can measure the water flow and if the necessary quantity of water flow through the pipe then water flow can be stopped automatically. The calculated values from the sensors can be processed by the Microcontrollers and uploaded to the internet through the Wi-Fi module (ESP 8266).
  1. **Objectives:**
* The objective of this project is to promote water conservation by providing real-time water consumption data to the public. This empowers individuals, communities, and industries to make informed decisions and take proactive measures to reduce water wastage and improve overall efficiency in water usage. By leveraging technology and data transparency, the project aims to foster a culture of responsible water management for a more sustainable future.
  1. **Outline of this Project:**
* This project aims to create a smart water management system using IoT technology. It will employ sensors, microcontrollers, and cloud platforms to monitor water resources efficiently. Users will have a user-friendly interface for monitoring and control, ensuring better water management.
  1. **Components Required:**

The components I used in this project are given below:

1. **ESP32 Processor:**



**ESP32** is a series of low-cost, low-power system on a chip microcontrollers  with integrated Wi-Fi and dual-mode Bluetooth.

1. **Ultrasonic sensor hc-sr04:**



The HC-SR04 Ultrasonic Distance Sensor is a sensor used for detecting the distance to an object using sonar. It's ideal for any robotics projects your have which require you to avoid objects, by detecting how close they are you can steer away from them.

1. **9V Battery:**



1. **Jumper Wires:**



1. **USB Cables:**



* 1. **Step by Step Procedure:**

**Step 1:** **Gather Hardware and Set Up Smart Meters**

• Acquire the necessary hardware, including ESP32 microcontrollers, smart water meters, and IoT sensors.

• Install the smart water meters at the water outlets in the parks and gardens.

**Step 2:** **Configure the ESP32**

• Program the ESP32 microcontrollers to interface with the smart water meters and IoT sensors. Use the Arduino IDE, PlatformIO, or your preferred development environment.

• Integrate the Blynk library into your ESP32 code to enable communication with the Blynk platform. Make sure to use the unique Blynk Auth Token for your project.

**Step 3**: **Set Up IoT Sensors**

• Attach IoT sensors to the smart water meters to monitor relevant data, such as water flow rates, temperature, and humidity. Configure these sensors to provide data to the ESP32.

**Step 4:** **Establish Internet Connectivity**

• Set up a Wi-Fi network in the parks and gardens to ensure the ESP32 processors can connect to the internet.

**Step 5**: **Develop the Blynk Interface**

• Create a Blynk project in the Blynk app builder that will serve as the public platform for displaying water consumption data.

• Design a user-friendly interface within the Blynk app, using widgets to visualize data, such as gauges, graphs, and labels.

**Step 6: Send Data to Blynk**

• Modify your ESP32 code to periodically collect data from the sensors and smart meters.

• Use the Blynk library to send this data to the Blynk project you created. Ensure the data is properly formatted and transmitted using the Blynk Auth Token.

**Step 7:** **Data Storage and Processing**

• Set up a cloud-based or on-premises server to receive, process, and store the data from the ESP32s. You can use platforms like AWS, Azure, or Google Cloud for this purpose.

**Step 8:** **User Awareness and Engagement**

• Launch an awareness campaign to promote the public platform. Encourage users to access the Blynk app to view real-time water consumption data and learn about water conservation efforts.

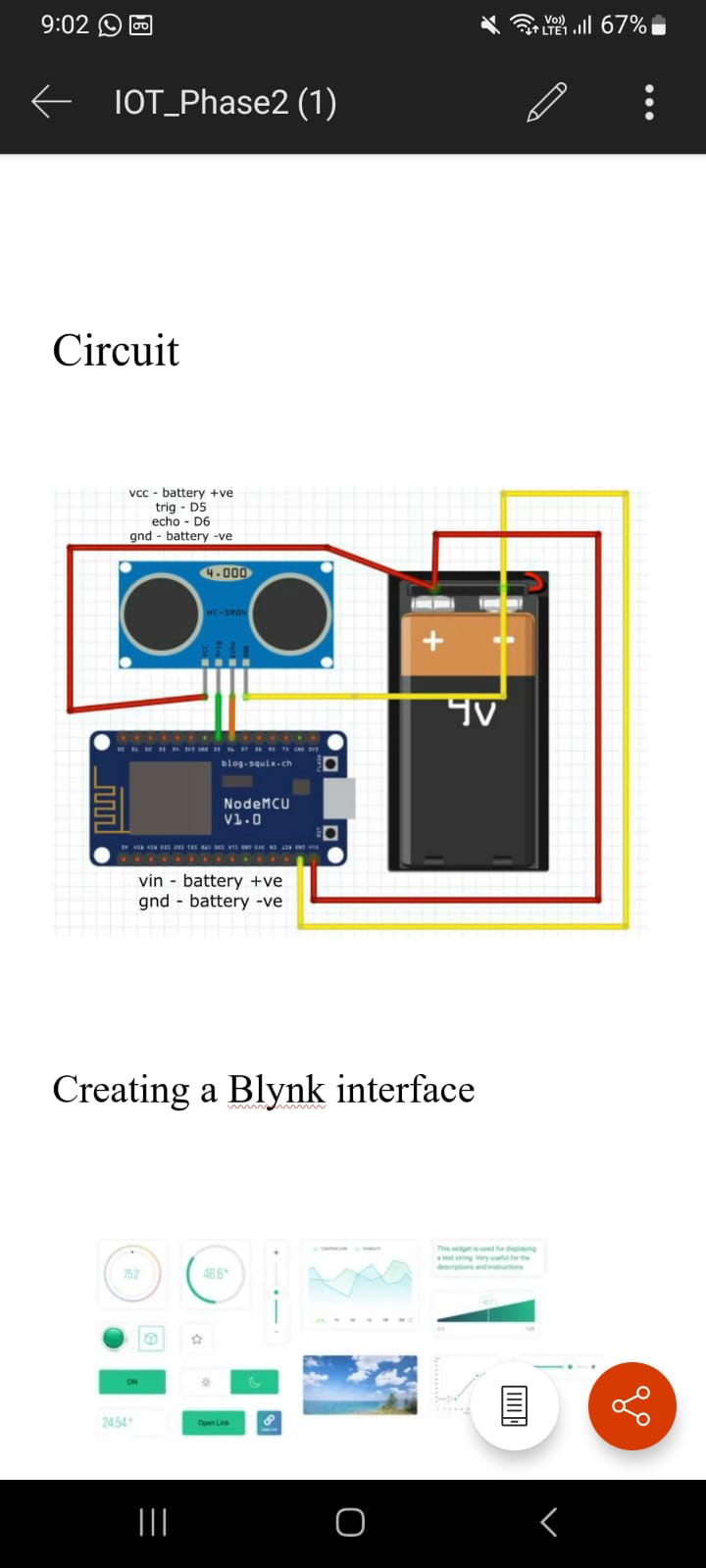
• Configure notifications and alerts within Blynk to engage users when unusual water consumption patterns are detected.

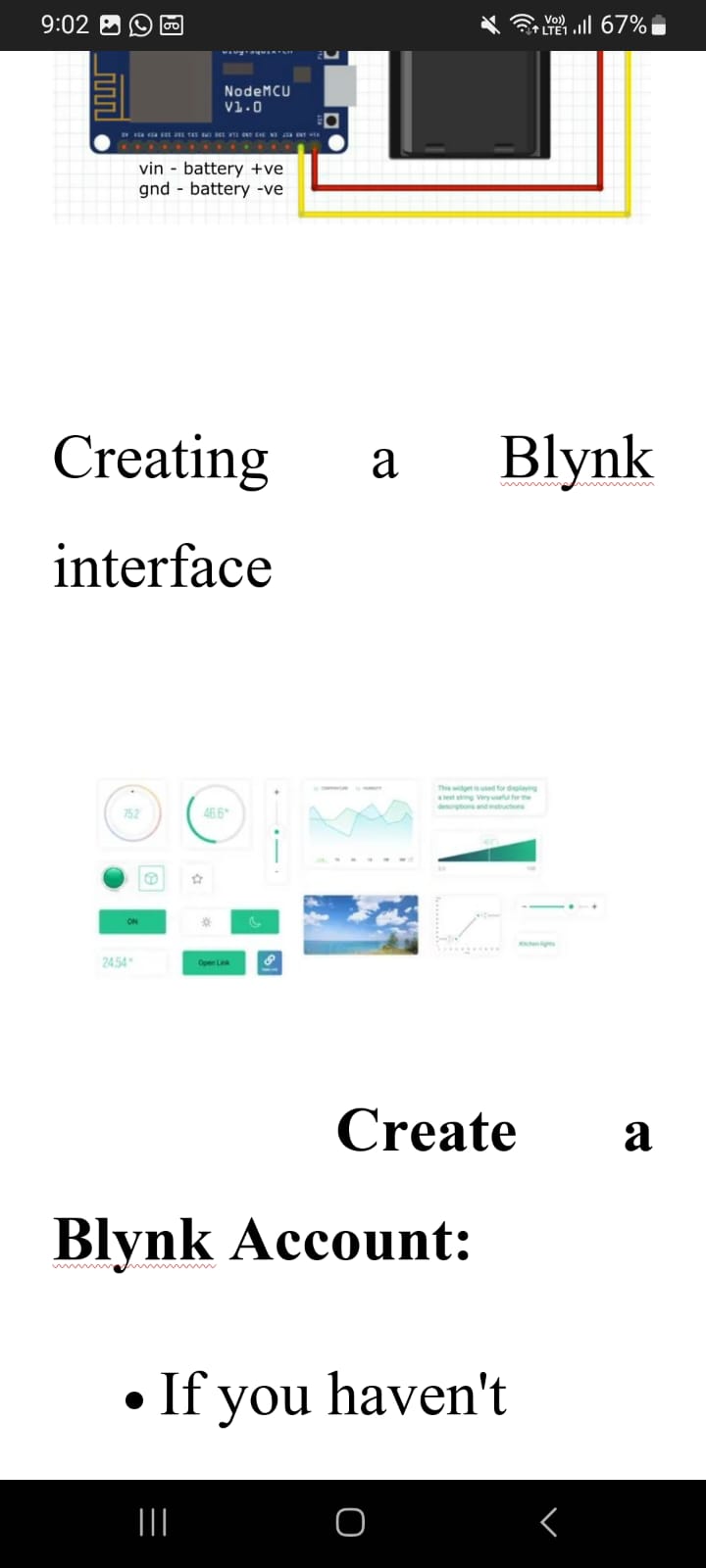
**Step 9:** **Monitor and Maintain**

• Continuously monitor the system's performance and the accuracy of the data. Make necessary adjustments and conduct regular maintenance.

• Update the Blynk project interface and the ESP32 code as needed to improve the system's functionality.

* 1. **Circuit:**



* 1. **Blynk Interface:**

**Create a Blynk Account:**

If you haven't already, start by creating an account on the Blynk platform.

**Create a Blynk Project:**

After logging in, create a new project within your Blynk account.

**Select Hardware Model:**

In your Blynk project settings, choose the hardware model you'll be using. For an ESP32, you can select "ESP32 Dev Board" or a similar option.

**Generate Blynk Auth Token**:

Each project will have a unique authentication token. Click on the "Device Info" button to generate a Blynk Auth Token for your project. You'll need this token to connect your hardware to the Blynk project.

**Add Widgets:**

Customize your project by adding widgets. Blynk offers a variety of widgets to choose from, including buttons, sliders, graphs, displays, and more. Each widget serves a specific purpose, such as displaying data or controlling devices.

**Configure Widgets:**

Configure each widget according to your project's requirements. This includes setting the widget type, pin, data range, labels, and more. Widgets are linked to your hardware's pins, so they should match your ESP32 pin configuration.

**Design the User Interface:**

Use the Blynk app builder to design the user interface. Drag and drop widgets onto the canvas, arrange them as desired, and set their properties.

**Set Data Frequency and Widgets Behavior:**

Define how often data should be updated for widgets that display information. Widgets like graphs can be configured to display historical data. Buttons and sliders can control hardware devices.

**Write Code for Hardware:**

In your ESP32 or hardware's code, incorporate the Blynk library. Use your Blynk Auth Token to establish a connection between your hardware and the Blynk project. Configure your code to send data to the specified pins that correspond to the widgets in your Blynk project.

**Test Your Project:**

Upload the code to your hardware and monitor the data and control functions through the Blynk app. Ensure that the data is being received and that the widgets function as expected.

**Fine-Tune and Iterate:**

Refine your Blynk interface as needed. You can modify the widget settings or layout to better suit your project's requirements. Iterate and test until the interface aligns with your project's goals.

**Public Access :**

If your project is intended for public access, you can share your Blynk project by making it accessible with a QR code or a unique link. Users can download the Blynk app and access your project by scanning the QR code or clicking the link.

**Secure Your Project (if needed):**

If your project deals with sensitive data, ensure data security and access control by setting up appropriate security measures within Blynk.

After creating this Blynk interface we are going to spread it to public use and make sure they are aware of it.

* 1. **Python Script:**

# Import necessary libraries

import time

import random

import requests # For HTTP requests

# Define the URL and authentication credentials for the data-sharing platform

BASE\_URL = "http://your\_data\_platform.com/api/endpoint"

API\_KEY = "your\_api\_key"

def read\_water\_consumption():

# Simulated function to read water consumption from the sensor

return random.uniform(1, 10) # Adjust this based on your actual sensor data

def send\_data\_to\_platform(consumption):

# Prepare data payload

data = {"water\_consumption": consumption}

# Add any additional data fields if required

# Set headers and authentication

headers = {

"Content-Type": "application/json",

"Authorization": f"Bearer {API\_KEY}"

}

# Make the HTTP request

response = requests.post(BASE\_URL, json=data, headers=headers)

# Check if the request was successful

if response.status\_code == 200:

print("Data sent successfully!")

else:

print(f"Error sending data. Status code: {response.status\_code}")

if \_\_name\_\_ == "\_\_main\_\_":

while True:

# Read water consumption data

consumption = read\_water\_consumption()

# Send data to the data-sharing platform

send\_data\_to\_platform(consumption)

# Adjust the time interval based on your requirements

time.sleep(60) # Send data every 60 seconds

* 1. **Program:**

#include <WiFi.h>

#include <BlynkSimpleEsp32.h>

// Wi-Fi and Blynk credentials

char ssid[] = "YourSSID";

char pass[] = "YourPassword";

char auth[] = "YourAuthToken";

// Pin configurations

#define TRIG\_PIN 14 // GPIO pin for the trigger pin of the ultrasonic sensor

#define ECHO\_PIN 12 // GPIO pin for the echo pin of the ultrasonic sensor

void setup() {

Serial.begin(115200);

Blynk.begin(auth, ssid, pass);

pinMode(TRIG\_PIN, OUTPUT);

pinMode(ECHO\_PIN, INPUT);

}

void loop() {

Blynk.run();

// Ultrasonic sensor reading

long duration, distance;

digitalWrite(TRIG\_PIN, LOW);

delayMicroseconds(2);

digitalWrite(TRIG\_PIN, HIGH);

delayMicroseconds(10);

digitalWrite(TRIG\_PIN, LOW);

duration = pulseIn(ECHO\_PIN, HIGH);

distance = (duration / 2) / 29.1; // Convert to centimeters

// Send the distance data to Blynk

Blynk.virtualWrite(V1, distance);

delay(1000); // Update frequency

}

* 1. **Conclusion:**
* Smart water management, enabled by IoT technology, revolutionizes how we monitor and conserve water. It provides real-time data, improves environmental sustainability, enhances public health, and strengthens communities. Ongoing collaboration, innovation, and investment are essential for a sustainable water future.