KGiSL Institute of Technology

(Affiliated to ANNA University, Chennai and Approved by AICTE, New Delhi)

365, KGiSL Campus, Thudiyalur Road, Saravanampatti
Coimbatore – 641035

Department of Artificial Intelligence and Data Science

Name : **Isaac.S**

Register Number : **711721243035**

Regulation : **R-2021**

Branch : **B.Tech -Artificial Intelligence and Data Science**

Project Title : **Smart Water Fountain**

Semester/ Year : **V / III**

Project Title: Smart Water Fountains

Phase 4: Development Part 2

Hardware and Sensors:

- Select the appropriate IoT sensors for your project. In this case, you'll need flow rate sensors and pressure sensors.
- Choose microcontrollers or IoT development boards that can connect to the sensors and send data to the platform. Common options include Raspberry Pi, Arduino, or ESP8266/ESP32.

Wiring and Sensor Integration:

- Connect the flow rate and pressure sensors to the chosen microcontroller. Refer to the sensor datasheets and microcontroller documentation for wiring instructions. Ensure power and ground connections are made correctly.
- Write code on the microcontroller to read data from these sensors. Libraries and sample code are often available for specific sensors.

IoT Platform Selection:

- Choose an IoT platform to collect and manage data. Common platforms include AWS IoT, Google Cloud IoT, or Azure IoT.
- Create an account and set up a new IoT project on the chosen platform.

Python Script for Data Transmission:

Develop a Python script to read sensor data and send it to your selected IoT platform. The exact code will depend on the microcontroller and platform you're using.

Here's an example Python script for sending data to AWS IoT using a Raspberry Pi and Python:

```
import time
import boto3
import RPi.GPIO as GPIO  # Assuming you are using a Raspberry Pi
```

```
from AWSIoTPythonSDK.MQTTLib import AWSIoTMQTTClient

# AWS IoT Configuration
IoT_ENDPOINT = "your-iot-endpoint.amazonaws.com"
ROOT_CA = "root-CA.pem"
PRIVATE_KEY = "your-private-key.pem.key"
CERTIFICATE = "your-certificate.pem.crt"
TOPIC = "water-fountain-status"

# Initialize AWS IoT MQTT Client
myMQTTClient = AWSIoTMQTTClient("WaterFountainClient")
myMQTTClient.configureEndpoint(IoT_ENDPOINT, 8883)
myMQTTClient.configureCredentials(ROOT_CA, PRIVATE_KEY, CERTIFICATE)
myMQTTClient.configureOfflinePublishQueueing(-1)
myMQTTClient.configureDrainingFrequency(2)
myMQTTClient.configureConnectDisconnectTimeout(10)
myMQTTClient.configureMQTTOperationTimeout(5)

# Connect to AWS IoT
myMQTTClient.connect()

# Function to read sensor data
def read_sensors():
    # Replace with your code to read from flow rate and pressure sensors
    flow_rate = 0.0
    pressure = 0.0
    return flow_rate, pressure

try:
    while True:
        flow_rate, pressure = read_sensors()
        data = {
            "flow_rate": flow_rate,
            "pressure": pressure
        }
```

```
        myMQTTClient.publish(TOPIC, str(data), 1)
        time.sleep(5)  # Adjust the interval as needed

except KeyboardInterrupt:
    GPIO.cleanup()
    myMQTTClient.disconnect()
```

Front-End:

HTML Structure:

The HTML structure of the Smart Water Fountain Status platform defines the layout and presentation of real-time data. This section guides you through the structure of the HTML file (index.html) that powers the front-end.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>Smart Water Fountain Status</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <div class="status-container">
        <h1>Water Fountain Status</h1>
        <div class="flow-rate">
            <h2>Flow Rate: <span id="flow-rate-value">0</span> GPM</h2>
        </div>
        <div class="malfunction-alert">
            <h2>Malfunction Alert: <span id="malfunction-status">No</span></h2>
        </div>
    </div>
    <script src="app.js"></script>
</body>
</html>
```

3.2 CSS Styles

The CSS styles define the visual design and layout of the platform. This section provides insights into the styles applied through the styles.css file.

Body Styling: Defines the font family and provides a clean appearance for the platform.

Status Container Styling: Provides central alignment and margin for the main status container.

Flow Rate and Malfunction Alert Styling: Defines the styling for displaying flow rate and malfunction status, with an emphasis on readability.

3.3 JavaScript

The JavaScript file (script.js) adds interactivity to the platform. This section explains how the script fetches data from the server and updates the platform in real-time.

Updating Flow Rate: The script includes a function to update the water flow rate displayed on the platform.

Updating Malfunction Status: A function to update the malfunction status displayed on the platform.

Fetching and Updating Data: Details on how the script fetches data from the server's API endpoint and uses it to update the displayed data.

Periodic Updates: The script sets up periodic data updates to keep the platform's information current.

Back-End (Node.js)

Server Script

The Node.js server script (app.js) serves as the backbone of the Smart Water Fountain Status platform. This section outlines the purpose and functionality of the server script.

The server script covers the following aspects:

Server Initialization: Information on initializing the Node.js server using Express and specifying the listening port.

Simulated Data: An explanation of the simulated data source used for demonstration purposes. Mention the need to replace this with a real data source.