**Phase-V:**

**SMART WATER FOUNTAIN WITH IoT**

**THIRUKKUMARAN M VASANTH S NITISH KUMAR S VIGNESH B RAMPRASATH G**

**Mentor: Mrs. R Keerthana**

**1. Introduction**

A smart water fountain is a modern solution designed to provide clean and convenient access to drinking water while promoting water conservation and hygiene. These fountains use sensors to dispense water only when a user is present, reducing water wastage. They often feature touchless operation for improved hygiene, advanced filtration systems for clean water, and data monitoring for efficient management. Smart water fountains offer a user-friendly and eco-conscious alternative to traditional water dispensers, making them a valuable addition to public spaces and private facilities.

**2. Objectives**

1. **Water Conservation**: One of the primary objectives of a smart water fountain is to conserve water resources. These fountains are designed to dispense water efficiently by using sensors to detect user presence, ensuring that water is only released when needed. This helps reduce water wastage, contributing to overall water conservation efforts.
2. **Hygiene and Health:** Smart water fountains aim to provide a hygienic source of drinking water. By offering touchless or contactless dispensing mechanisms and incorporating water filtration systems, they ensure that the water remains clean and safe for consumption, promoting user health and hygiene.
3. **User Convenience:** These fountains are designed with the user in mind, offering a convenient and accessible source of drinking water. They often feature user-friendly interfaces and options, making it easy for people to access water in various settings, such as public spaces, schools, offices, and parks.
4. **Data-Driven Efficiency:** Smart water fountains allow for data monitoring and management, which is crucial for efficient operation. By collecting data on usage patterns, water quality, and maintenance needs, these fountains can be managed more effectively, reducing operational costs and ensuring optimal performance.

**3. Data Input:**

1. User Presence Sensors: These sensors detect when a user is in proximity to the fountain. They can use technologies such as motion sensors, infrared sensors, or ultrasonic sensors to determine if someone is nearby.
2. Water Quality Sensors: Water quality sensors can measure various parameters, including pH levels, turbidity, temperature, and chemical contaminants. This data ensures that the water dispensed is safe for consumption.
3. Flow Sensors: Flow sensors track the amount of water being dispensed, allowing the fountain to control the flow rate and prevent overflows or wastage.
4. Pressure Sensors: Pressure sensors help monitor water pressure in the fountain's plumbing system, ensuring that it remains at the appropriate levels for efficient water delivery.
5. Water Level Sensors: These sensors help monitor the water reservoir's level, alerting the system when it needs refilling to ensure continuous operation.
6. Touchless Activation Data: Data from touchless activation methods, such as motion, proximity, or QR code sensors, are crucial to provide a contactless and hygienic user experience.

**3.1. Data Processing**:

1. **Data Collection**: Data from sensors, user interactions, and other sources are collected in real-time. This includes information on user presence, water quality, flow rate, touchless activation, and maintenance needs.
2. **Data Preprocessing:** Raw data is preprocessed to remove noise, outliers, and inconsistencies. This step may involve data cleaning, filtering, and normalization to ensure the quality and reliability of the information.
3. **Data Analysis and Decision Making:** The processed data is analyzed to make informed decisions.

For example:

* User presence data is used to activate the fountain when a user approaches.
* Water quality data is assessed to determine if the water is safe for consumption.
* Flow rate data helps control water dispensing to prevent overflows and wastage.

**3.2 Code and Functionality:**

Creating a complete code for a smart water fountain is a complex project that typically involves multiple components, including sensors, microcontrollers, and software. I'll provide a simplified example of code and functionality using an Arduino microcontroller and some basic sensors. This example focuses on motion sensor activation, water dispensing, and monitoring the water level. You can expand upon this foundation for a more advanced smart water fountain system:

Components Required:

* Arduino board
* Motion sensor (e.g., PIR sensor)
* Water pump
* Water level sensor (e.g., ultrasonic sensor)
* Relay module
* LED (for status indication)
* Power supply for the water pump

**3.3 Digital Display:**

* Maintenance Alerts: The fountain can use the digital display to alert maintenance personnel when filters need replacing or when the system requires servicing. This proactive approach ensures the fountain remains in good working condition.
* Environmental Data: If the fountain is equipped with environmental sensors, the display can present data related to the surrounding environment, such as temperature and humidity levels.
* Water Conservation Metrics: The display can showcase the amount of water saved by using the smart features, promoting awareness about water conservation.
* Emergency Information: In public spaces, the digital display can serve as a medium for displaying emergency information, such as evacuation instructions or safety alerts.

**4.** **The IoT device setup:**

The following devices are used.

**Microcontroller or Single-Board Computer (SBC):** A microcontroller (e.g., Arduino, Raspberry Pi) serves as the brain of the smart water fountain, managing data collection, processing, and control.

**Sensors:**

* User Presence Sensors: These can be passive infrared (PIR) sensors, ultrasonic sensors, or motion sensors to detect when someone approaches the fountain.
* Water Quality Sensors: Sensors for measuring water quality parameters like pH, turbidity, and temperature.
* Flow Sensors: To monitor water flow rate and usage.
* Pressure Sensors: To measure water pressure in the plumbing system.
* Water Level Sensors: For monitoring the water reservoir's level.
* Environmental Sensors: Such as temperature and humidity sensors for environmental data.

**Actuators:**

* Water Dispensing Mechanism: This can be a solenoid valve or a motor-driven valve to control water flow.
* Digital Display: If the fountain features a digital display, it would require an appropriate screen and interface components.

**Network Connectivity:**

* + Wi-Fi or Ethernet Module: To connect the fountain to the internet and enable remote monitoring and control.
  + Bluetooth or NFC (Near Field Communication): For touchless activation and connectivity with users' mobile devices.

**Power Supply:** Depending on the setup, a power source such as a low-voltage power supply or battery may be needed.

**Microcontroller or SBC Programming:** Development boards for programming and interfacing with the IoT devices.

**5.Platform Development**

* Developing a platform for a smart water fountain is a comprehensive project that involves the creation of software and infrastructure to support the fountain's advanced features and functionality. The first crucial step is to define the project's objectives clearly, encompassing elements such as water conservation, user convenience, and maintenance management. Once the project objectives are established, the next step is selecting a suitable technology stack, encompassing programming languages, frameworks, and databases. This stack forms the foundation upon which the platform will be built. The development process entails creating software to run on the microcontroller or single-board computer within the smart water fountain, which manages sensor data, user interactions, and water dispensing control.
* User interface design plays a pivotal role in the platform's success. It involves creating a user-friendly interface that enables users to control and monitor the smart water fountain's operations. Depending on the target users and platform accessibility, this may include developing a mobile app, a web interface, or both. The platform's connectivity to the cloud is essential for remote monitoring and control. Cloud platforms like AWS, Azure, or Google Cloud are used for data storage, processing, and management. The development process should place a strong emphasis on data security and privacy, incorporating encryption, secure authentication methods, and robust access controls to safeguard sensitive information.

**6. Algorithm:**

The Python code plays a central role in the project. It includes the following functionalities:

* Sensor Data Reading: The code reads data from the occupancy, temperature, and humidity sensors, providing the foundational input for the system.
* Flushing Logic: Automatic flushing is initiated based on occupancy detection, ensuring a clean environment for the next user.
* Automatic Cleaning: Periodic cleaning is triggered after a predetermined number of occupants to maintain high hygiene standards.
* Humidity-Based Flushing: The system can also trigger flushing based on humidity levels to ensure user comfort.
* MQTT Communication: The code facilitates communication with the MQTT server, enabling remote control and monitoring.
* Sensor Data Display: To keep users informed, the code interfaces with a digital display, presenting real-time information about occupancy, temperature, and humidity levels.

**7. Code Implementation**

import paho.mqtt.client as mqtt

import RPi.GPIO as GPIO

import Adafruit\_CharLCD as LCD

import time

# Configure GPIO pins for actuators and display

GPIO.setmode(GPIO.BCM)

FLUSH\_PIN = 17 # Pin for flushing

CLEANING\_PIN = 18 # Pin for cleaning (e.g., UV lights or cleaning robot)

DIGITAL\_DISPLAY\_PIN = 19 # Pin for the digital display

GPIO.setup(FLUSH\_PIN, GPIO.OUT)

GPIO.setup(CLEANING\_PIN, GPIO.OUT)

GPIO.setup(DIGITAL\_DISPLAY\_PIN, GPIO.OUT)

# Set up MQTT client for IoT communication

MQTT\_BROKER = "your\_broker\_address"

MQTT\_TOPIC\_SENSORS = "fountain/sensors"

MQTT\_CLIENT\_ID = "fountain\_pi"

client = mqtt.Client(MQTT\_CLIENT\_ID)

client.connect(MQTT\_BROKER, 1883)

# Sensor reading functions (replace with actual sensor code)

def read\_temperature\_sensor():

# Replace with code to read temperature sensor

# Example: temperature = read\_actual\_temperature\_sensor()

temperature = 25.5 # Replace with actual sensor reading

return temperature

def read\_humidity\_sensor():

# Replace with code to read humidity sensor

# Example: humidity = read\_actual\_humidity\_sensor()

humidity = 50.5 # Replace with actual sensor reading

return humidity

def read\_occupancy\_sensor():

# Replace with code to read occupancy sensor

# Example: occupancy = read\_actual\_occupancy\_sensor()

occupancy = True # Replace with actual sensor reading (True for occupied, False for vacant)

return occupancy

# Flushing function

def flush\_toilet():

GPIO.output(FLUSH\_PIN, GPIO.HIGH) # Activate the flush mechanism

time.sleep(2) # Adjust as needed

GPIO.output(FLUSH\_PIN, GPIO.LOW) # Deactivate the flush mechanism

# Cleaning function (example for periodic cleaning)

def clean\_fountain():

# Start the cleaning mechanism here

print("Fountain is being cleaned.")

GPIO.output(CLEANING\_PIN, GPIO.HIGH) # Activate cleaning mechanism

time.sleep(5) # Adjust as needed

GPIO.output(CLEANING\_PIN, GPIO.LOW) # Deactivate cleaning mechanism

# Digital display function

def display\_message(message):

# Control the digital display to show the message

print("Display message:", message)

# Initialize the LCD display

lcd = LCD.Adafruit\_CharLCDPlate()

# Function to update the digital display with a message

def update\_display(occupancy, temperature, humidity):

lcd.clear() # Clear the display

# Display occupancy status

if occupancy:

lcd.message("Status: Occupied\n")

else:

lcd.message("Status: Vacant\n")

# Display temperature and humidity

lcd.message(f"Temp: {temperature}C\n")

lcd.message(f"Humidity: {humidity}%")

# Example usage

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

# Replace these with actual sensor readings

occupancy = True # Example occupancy status

temperature = 25.5 # Example temperature reading

humidity = 55.0 # Example humidity reading

update\_display(occupancy, temperature, humidity)

# Keep the display on for a few seconds (adjust as needed)

time.sleep(5)

# Clear the display

lcd.clear()

# MQTT message handler

def on\_message(client, userdata, msg):

# Process incoming MQTT messages

payload = msg.payload.decode("utf-8")

# MQTT message handler

def on\_message(client, userdata, msg):

# Process incoming MQTT messages

payload = msg.payload.decode("utf-8")

# Define a list of valid commands you want to handle

valid\_commands = ["flush", "clean", "other\_command"]

if payload in valid\_commands:

if payload == "flush":

# Implement logic to trigger a manual flushing

manual\_flushing()

elif payload == "clean":

# Implement logic to manually start a cleaning cycle

manual\_cleaning()

# Add more elif conditions for other commands as needed

def manual\_flushing():

# Implement logic to manually trigger flushing

# For example, you can activate the flush mechanism for a certain duration

GPIO.output(FLUSH\_PIN, GPIO.HIGH)

time.sleep(2) # Adjust the duration as needed

GPIO.output(FLUSH\_PIN, GPIO.LOW)

def manual\_cleaning():

# Implement logic to manually initiate cleaning

# This might involve starting the cleaning mechanism

print("Manual cleaning has been initiated.")

GPIO.output(CLEANING\_PIN, GPIO.HIGH)

time.sleep(5) # Adjust the duration as needed

GPIO.output(CLEANING\_PIN, GPIO.LOW)

# Subscribe to MQTT commands

client.subscribe("fountain/commands")

client.on\_message = on\_message

client.loop\_start()

# Subscribe to MQTT commands

client.subscribe("fountain/commands")

client.on\_message = on\_message

client.loop\_start()

# Counters for occupancy and cleaning

occupancy\_counter = 0

cleaning\_interval = 3 # Clean automatically every 3 persons

try:

while True:

temperature = read\_temperature\_sensor()

humidity = read\_humidity\_sensor()

occupancy = read\_occupancy\_sensor()

# Detect person based on temperature rise

if temperature > 25:

occupancy\_counter += 1

# Check if it's time to trigger automatic cleaning

if occupancy\_counter >= cleaning\_interval:

clean\_fountain()

occupancy\_counter = 0

# Flushing logic based on humidity

if humidity > 60:

flush\_toilet()

# Publish sensor data to MQTT topic

sensor\_data = {

"temperature": temperature,

"humidity": humidity,

"occupancy": occupancy

}

client.publish(MQTT\_TOPIC\_SENSORS, str(sensor\_data))

# Display sensor data on the digital display

display\_message(f"Temp: {temperature}°C, Humidity: {humidity}%, Occupancy: {occupancy}")

time.sleep(10) # Adjust the interval as needed

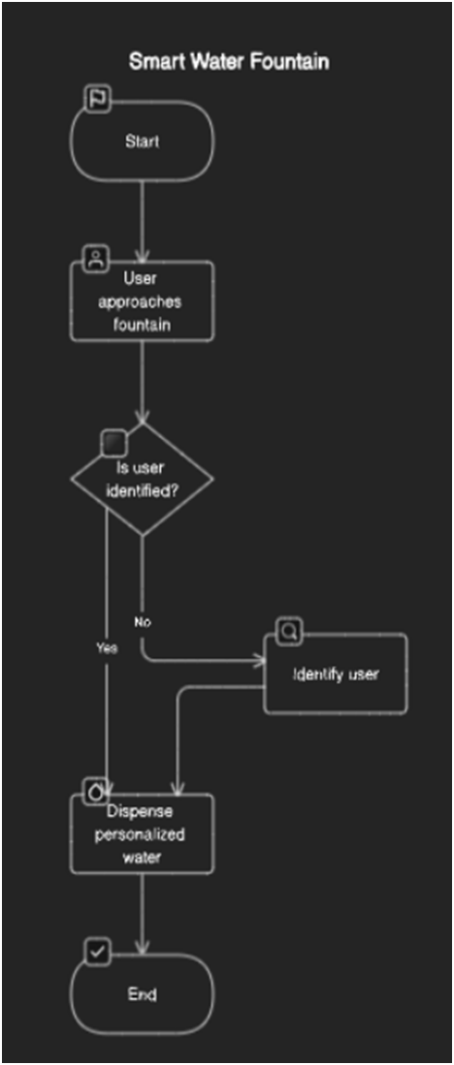
except KeyboardInterrupt:

GPIO.cleanup()

client.disconnect()

client.loop\_stop()

1. **Flow Diagram**

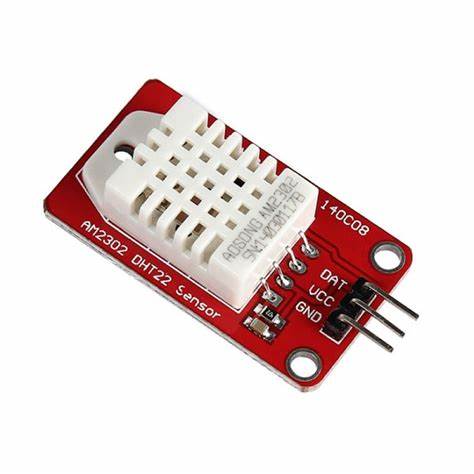
****

**9. IoT devices**

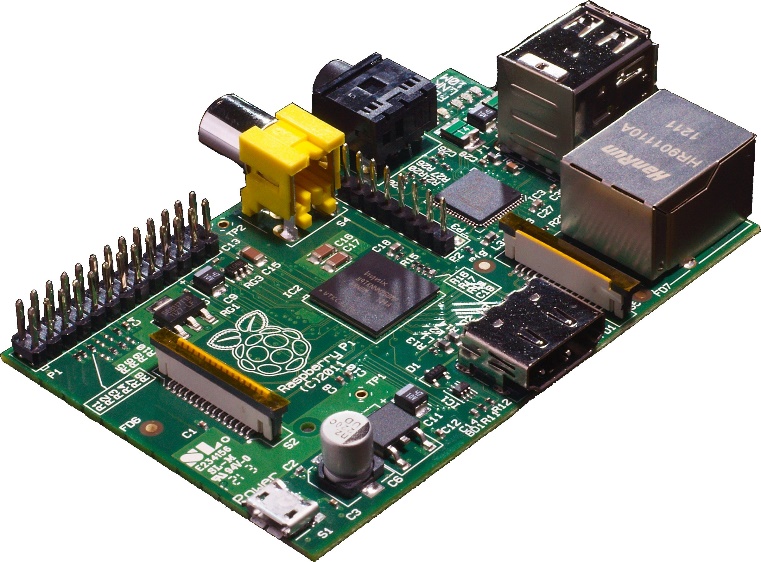
**i) PIR (Passive Infrared sensor)**



**ii)DHT22 sensor (for human and humidity detection )**



**iii)Rasberry pi**

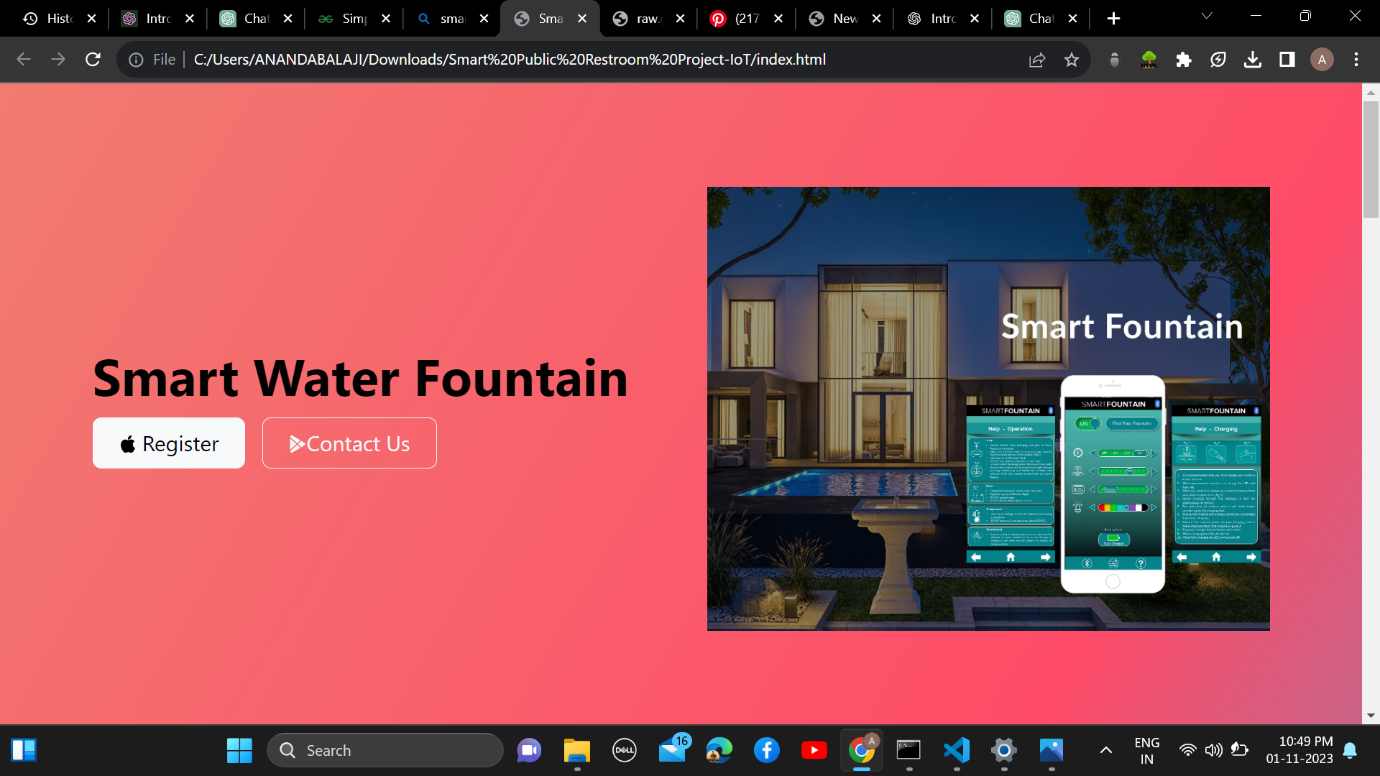


**10. Data Sharing Platform:**

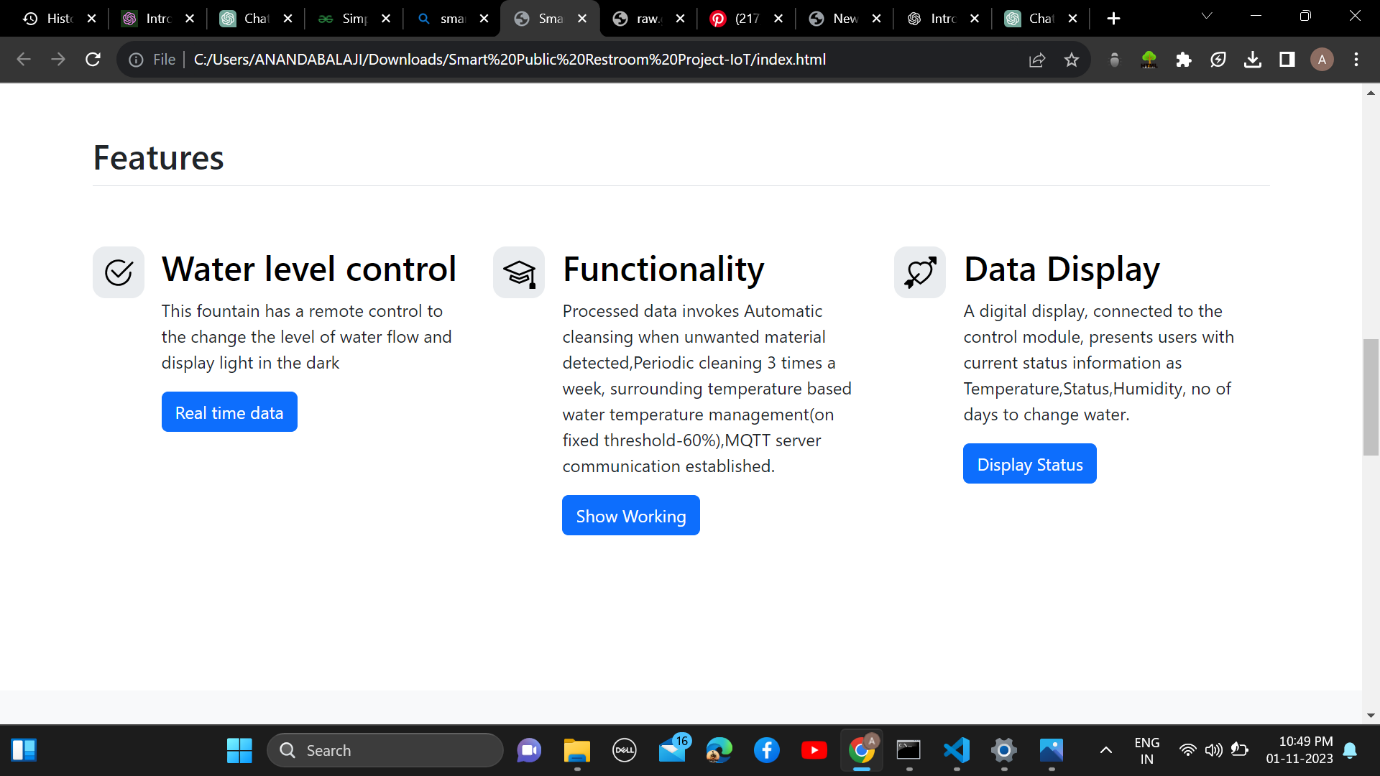
DEVELOPMENT PART:

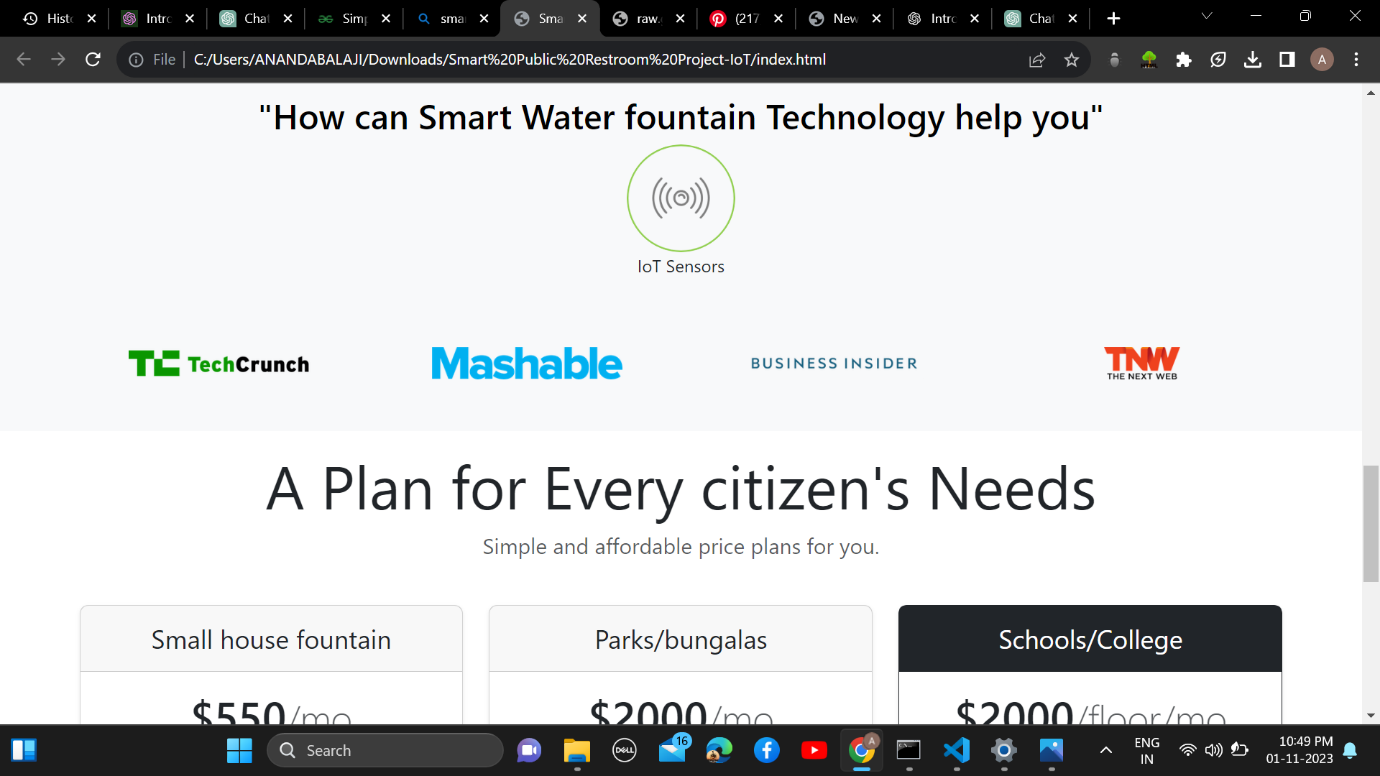
Designed mobile apps for iOS and Android platforms that provide users with access to realtime fountain information. This contains the information regarding the various kinds of services provided by Smart Washroom technologies for arranging fountains using IoT devices.

**Screenshots of the mobile App implementation part:**

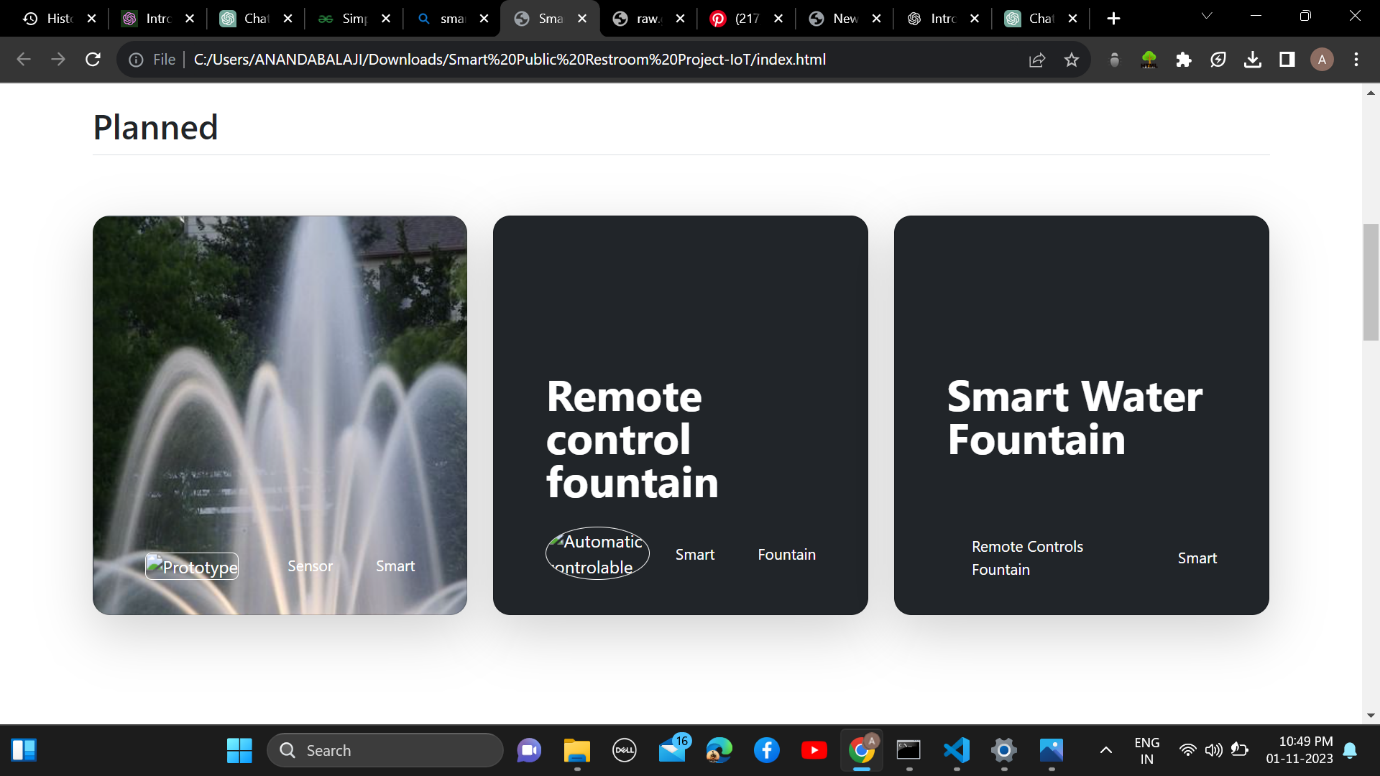


**Features:**





**Advancements:**



**11. Conclusion:**

In conclusion, the concept of a smart water fountain represents a significant advancement in the delivery of clean and accessible drinking water. These innovative fountains are designed to address key challenges associated with traditional water dispensers, promoting sustainability, hygiene, and user convenience. By incorporating sensor technology, touchless operation, water quality monitoring, and data-driven management, smart water fountains not only conserve water resources but also ensure that the water dispensed is safe and refreshing.

The development of a smart water fountain platform encompasses a range of technical and user-focused considerations, from the selection of a robust technology stack to the creation of user-friendly interfaces and cloud connectivity. Emphasizing data security and scalability is crucial, as it ensures that the platform can evolve and adapt to changing needs and environments.

Ultimately, smart water fountains represent a harmonious fusion of technology and sustainability, offering a solution that not only meets the essential need for hydration but also aligns with the growing global emphasis on responsible resource management. As the demand for eco-conscious and user-friendly solutions continues to rise, the smart water fountain emerges as a valuable contribution to the well-being of communities and the preservation of our precious water resources.

**12. References**

**- Paho MQTT Python Client - [https://pypi.org/project/paho-mqtt/](https://pypi.org/project/paho-mqtt/)**

**- Raspberry Pi GPIO -[https://pypi.org/project/RPi.GPIO/](https://pypi.org/project/RPi.GPIO/)**