**Team Members:**

**Vinisha M**

**Punith S**

**Swathi S J**

**Ruthranayaki D**

**Environmental monitoring in parking system in IOT**

**Introduction:**

Environmental monitoring in a parking system typically involves using various sensors and technologies to collect data, which can then be processed and analyzed to make informed decisions or trigger actions. Implementing a complete system with code can be a complex task, but I can provide a simple example in Python to demonstrate the concept of environmental monitoring within a parking system using a temperature sensor as an illustration. In a real-world scenario, you would need the appropriate hardware and APIs for other sensors and systems.

Environmental monitoring in a parking system refers to the practice of using various sensors, technologies, and data analysis methods to continuously assess and manage environmental conditions within and around a parking facility. The data collected through environmental monitoring systems can be used for real-time decision-making, improved security, energy efficiency, and enhancing the overall user experience within the parking facility. Drivers can access this information through a mobile app or website, or through signs that are posted in the parking lot itself.

**Project description:**

The "Environmental Monitoring in Parking System" project aims to enhance the sustainability and user experience of parking facilities by implementing a comprehensive environmental monitoring system. This system will utilize a network of sensors and advanced technologies to collect, analyze, and manage various environmental parameters within and around parking areas.

Hardware components:

* Sensors
* Data Logger
* Central Control System
* User Interface
* Alarms and Notifications
* Power Supply
* Environmental Enclosures
* Cameras and Video Monitoring
* Display Panels
* Access Control Systems
* Weather Station (Optional)

Software components:

* User Interface (UI)
* Database
* Notification System
* Reporting and Analytics
* Security and Authentication
* GIS Integration
* Administration and Configuration
* APIs

Program:

int value\_sensor = 0;

void setup()

{

pinMode(A1, INPUT);

Serial.begin(9600);

pinMode(6, OUTPUT);

}

void loop()

{

// Gas senor with buzzer

value\_sensor = analogRead(A1);

Serial.println(value\_sensor);

if (value\_sensor > 200) {

tone(6, 523, 1000); // play tone 60 (C5 = 523 Hz)

}

delay(10); // Delay a little bit to improve simulation performance

**To continue building the project by developing the environmental monitoring platform, we need to:**

Identify the specific environmental parameters that we want to monitor. This could include air quality, water quality, soil quality, noise levels, vibration levels, and/or other factors.

Select the appropriate sensors and devices to collect data on these parameters. There are a wide variety of environmental sensors available, so it is important to choose the ones that are most relevant to the specific parameters that we want to monitor and the environment where we will be deploying them.

Design and implement a system for collecting, transmitting, and storing the sensor data. This system may include a variety of hardware and software components, such as data acquisition systems, edge computing devices, cloud computing services, and data management software.

Develop data visualization and analytics tools to analyze the sensor data and identify trends and patterns. This will allow us to gain insights into the environmental conditions and identify any potential problems.

Develop a user interface for the environmental monitoring platform. This interface should allow users to easily view the sensor data, historical data, and analytics results.

1. Identify the specific environmental parameters that we want to monitor:

2. Select the appropriate sensors and devices to collect data on these parameters:

**Some factors:**

Accuracy

Precision

Range

Sensitivity

Durability

Cost

3. Design and implement a system for collecting, transmitting, and storing the sensor data:

4. Develop data visualization and analytics tools to analyze the sensor data and identify trends and patterns:

5. Develop a user interface for the environmental monitoring platform.

This interface should allow users to easily view the sensor data, historical data, and analytics results:

**create web development:**

A web framework is a software framework that provides a basic structure for developing web applications. There are many different web frameworks available, such as Django, Flask, and Ruby on Rails.

Set up a web server. A web server is a software application that hosts web pages and makes them accessible to users over the internet. There are many different web servers available, such as Apache and Nginx.

Write the server-side code. The server-side code is the code that will be responsible for collecting and processing the environmental data. This code will typically be written in a programming language such as Python, Ruby, or Node.js.

Write the client-side code. The client-side code is the code that will be responsible for displaying the real-time environmental data to the user. This code will typically be written in HTML, CSS, and JS.

**Code :**

HTML

<!DOCTYPE html>

<html>

<head>

<title>Real-time Environmental Data</title>

</head>

<body>

<h1>Real-time Environmental Data</h1>

<div id="data"></div>

<script>

function updateData() {

// Make a request to the server to get the latest environmental data.

var xhr = new XMLHttpRequest();

xhr.open("GET", "/data");

xhr.onload = function() {

if (xhr.status === 200) {

// The request was successful.

var data = JSON.parse(xhr.responseText);

// Update the DOM with the latest environmental data.

document.getElementById("data").innerHTML = "Temperature: " + data.temperature + "°C<br>Humidity: " + data.humidity + "%";

} else {

// The request failed.

console.log("Error fetching environmental data: " + xhr.statusText);

}

};

xhr.send();

}

// Set an interval to update the data every second.

setInterval(updateData, 1000);

</script>

</body>

</html>

Use code with caution. Learn more

This code will make a request to the server every second to get the latest environmental data. If the request is successful, the code will update the DOM with the latest data.

**Json file to insert other nesseory files or library:**

{

"version": 1,

"author": "Ashok Iyengar",

"editor": "wokwi",

"parts": [

{ "type": "wokwi-breadboard-mini", "id": "bb1", "top": 56.2, "left": -136.8, "attrs": {} },

{ "type": "wokwi-esp32-devkit-v1", "id": "esp", "top": -43.3, "left": 110.2, "attrs": {} },

{ "type": "wokwi-dht22", "id": "dht1", "top": 29.1, "left": -101.4, "attrs": {} }

],

"connections": [

[ "esp:TX0", "$serialMonitor:RX", "", [] ],

[ "esp:RX0", "$serialMonitor:TX", "", [] ],

[ "dht1:VCC", "bb1:5b.g", "", [ "$bb" ] ],

[ "dht1:SDA", "bb1:6b.g", "", [ "$bb" ] ],

[ "dht1:NC", "bb1:7b.g", "", [ "$bb" ] ],

[ "dht1:GND", "bb1:8b.g", "", [ "$bb" ] ],

[ "dht1:VCC", "esp:3V3", "red", [ "v0" ] ],

[ "dht1:GND", "esp:GND.1", "black", [ "v-38.4", "h268.8" ] ],

[ "dht1:SDA", "esp:D4", "green", [ "v-67.2", "h288.1" ] ],

[ "dht1:NC", "esp:D4", "green", [ "v-67.2", "h278.5" ] ]

],

"dependencies": {}

}

Coding for workflow:

#include <WiFi.h>

#include <HTTPClient.h>

#include <DHT.h>

// WiFi credentials

const char\* ssid = "Wokwi-GUEST";

const char\* password = "";

// Beeceptor endpoint

const char\* serverUrl = "https://smartenviron.free.beeceptor.com/smartenviron/";

// DHT sensor configuration

#define DHTPIN 4 // Define the GPIO pin to which the DHT22 is connected

#define DHTTYPE DHT22 // Define the sensor type (DHT11 or DHT22)

DHT dht(DHTPIN, DHTTYPE);

void setup() {

Serial.begin(9600);

Serial.print("Connecting to WiFi");

WiFi.begin("Wokwi-GUEST", "", 6);

while (WiFi.status() != WL\_CONNECTED) {

delay(100);

Serial.print(".");

}

Serial.println(" Connected!");

// Initialize the DHT sensor

dht.begin();

}

void loop() {

// Read temperature and humidity

float temperature = dht.readTemperature();

float humidity = dht.readHumidity();

if (!isnan(temperature) && !isnan(humidity)) {

// Create an HTTP client

HTTPClient http;

// Send temperature and humidity data to Beeceptor as form parameters

String postData = "temperature=" + String(temperature) + "&humidity=" + String(humidity);

http.begin(serverUrl);

http.addHeader("Content-Type", "application/x-www-form-urlencoded");

int httpResponseCode = http.POST(postData);

if (httpResponseCode > 0) {

Serial.print("HTTP Response code: ");

Serial.println(httpResponseCode);

Serial.println("Data sent to Beeceptor.");

} else {

Serial.print("Error in HTTP request. HTTP Response code: ");

Serial.println(httpResponseCode);

}

http.end();

} else {

Serial.println("Failed to read from DHT sensor!");

}

delay(60000); // Send data every 1 minute (adjust as needed)

}

**Libraries included :**

DHT sensor library:

WiFi

DHT22

HttpClient

Servo

Fast LED

To design a platform to receive and display real-time temperature and humidity data from IoT devices:

Here is a sample architecture for a platform to receive and display real-time temperature and humidity data from IoT devices:

**IoT devices <-> MQTT broker <-> Platform**

The IoT devices will publish the temperature and humidity data to the MQTT broker. The platform will subscribe to the MQTT broker and receive the data in real time. The platform will then store the data in a database, process it to generate insights, and visualize it for users.

**Detailed overview:**

IoT devices:

The IoT devices can be any type of device that can collect temperature and humidity data and send it over the internet. Examples of IoT devices that can be used for this purpose include:

Microcontrollers (e.g., Arduino, Raspberry Pi)

Single-board computers (e.g., BeagleBone Black, NVIDIA Jetson Nano)

Dedicated environmental sensors (e.g., DHT11, DHT22, SHT31)

Weather stations

Communication protocol:

communication protocols:

MQTT: MQTT is a lightweight messaging protocol that is well-suited for IoT devices because it is efficient and scalable.

CoAP: CoAP is a constrained protocol that is designed for devices with limited resources, such as battery-powered sensors.

XMPP: XMPP is an extensible messaging protocol that can be used for a variety of applications, including IoT.

Data storage:

The platform will need to store the temperature and humidity data so that it can be displayed to users in real time. A database can be used to store the data. Some popular databases for IoT applications include:

InfluxDB: InfluxDB is a time series database that is designed to store and query large amounts of time-stamped data.

TimescaleDB: TimescaleDB is a time series database that is built on top of PostgreSQL.

MongoDB: MongoDB is a NoSQL database that is well-suited for storing and querying data from a variety of sources, including IoT devices.

Data processing:

The platform will need to process the temperature and humidity data to calculate averages, trends, and other insights

Data visualization:

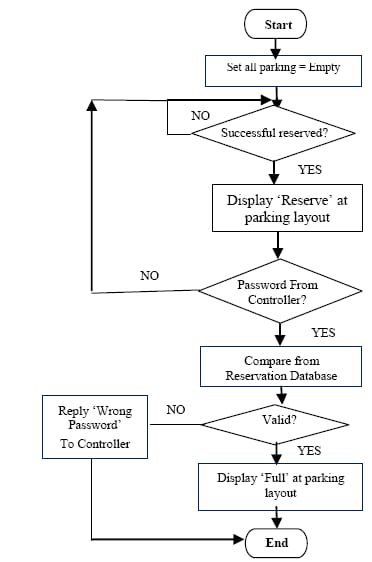
The platform will need to visualize the temperature and humidity data in a way that is easy for users to understand. This can be done using a variety of data visualization tools, such as:

Grafana: Grafana is a popular open-source data visualization platform.

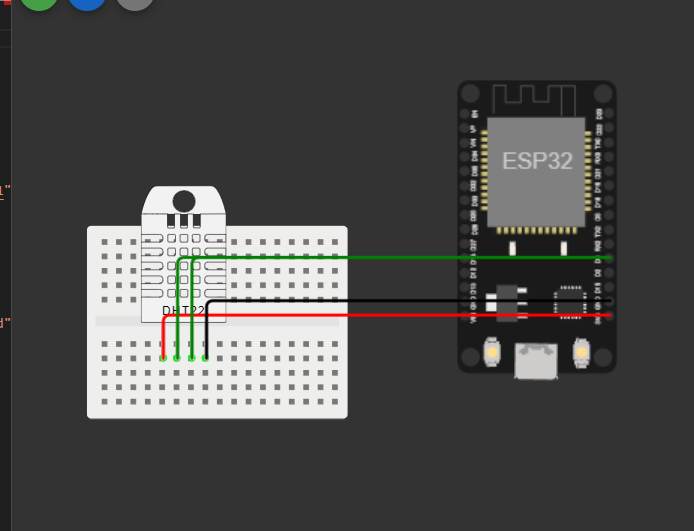
Kibana: Kibana is a data visualization tool that is part of the Elasticsearch stack.

Tableau: Tableau is a commercial data visualization platform.

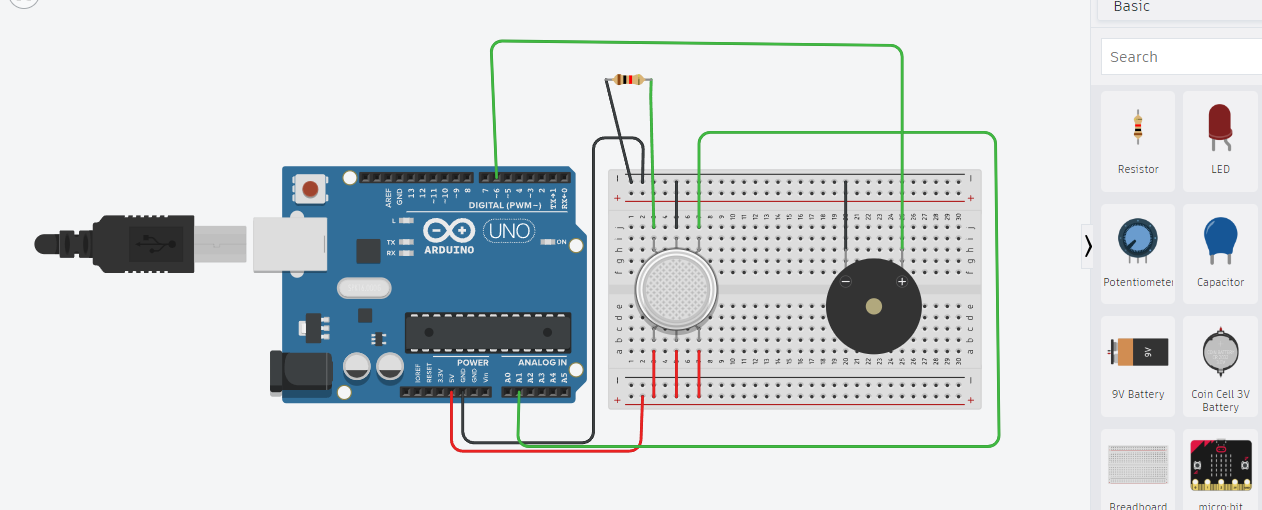
Flow Chart:



Simulaton Results:



Trinket Results:



Conclusion:

This code snippet simulates a temperature sensor and checks the temperature every 5 minutes. If the temperature goes above a certain threshold (25°C in this case), it prints a message indicating that the temperature is too high. In a real-world implementation, you would replace the temperature simulation code with actual sensor readings and replace the print statements with actions or alerts that are relevant to your parking system's needs.