**DHIRAJLAL GANDHI COLLEGE OF TECHNOLOGY**

*Department of electronic and communication and engineering*

IBM Naan Mudhalvan

**Internet Of Things**

phase 5 - Project documentation

Project title: smart water management

Team id: Proj\_223984\_Team

**Submitted on:**

01/11/2023

**Submitted by:**

Nithiyashree P

Swetha C

Shana J S

Sheeja S V

Mythili D

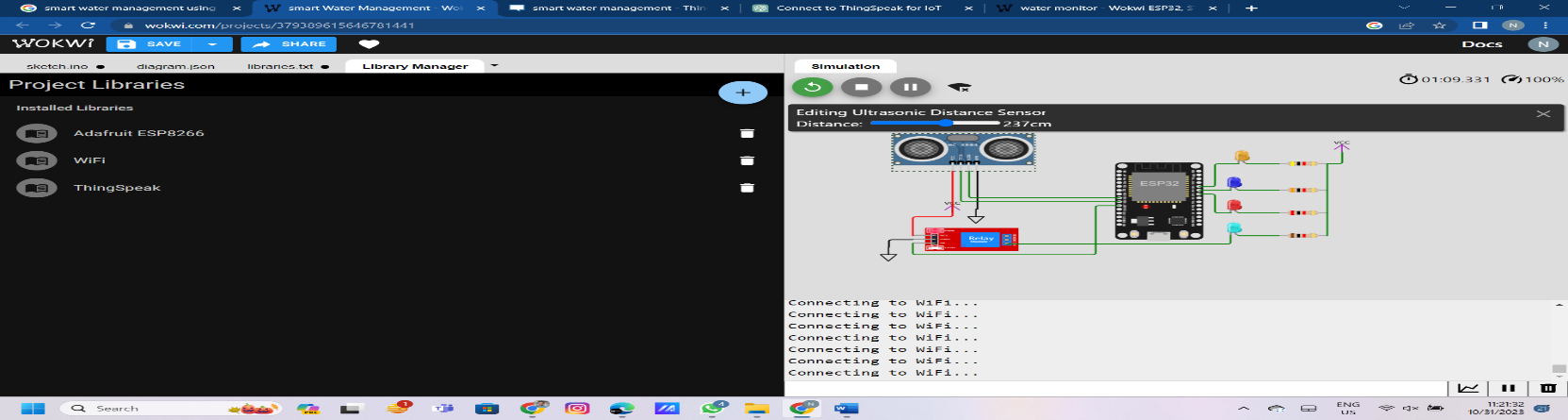
**INTRODUCTION:**

Smart water management is the iot based project which is used to monitor the flow of the water in public tanks and other public places to avoid to wastage of water. Where the circuit diagram is developed in wokwi platform to monitor the flow of water using ultrasonic sensor and the output was displayed in thinkspeak. Where the steps for the implementation of smart water management using wokwi and the web development are the following:

**CIRCUIT DESIGN USING WOKWI:**

**1. Including Necessary Libraries:**

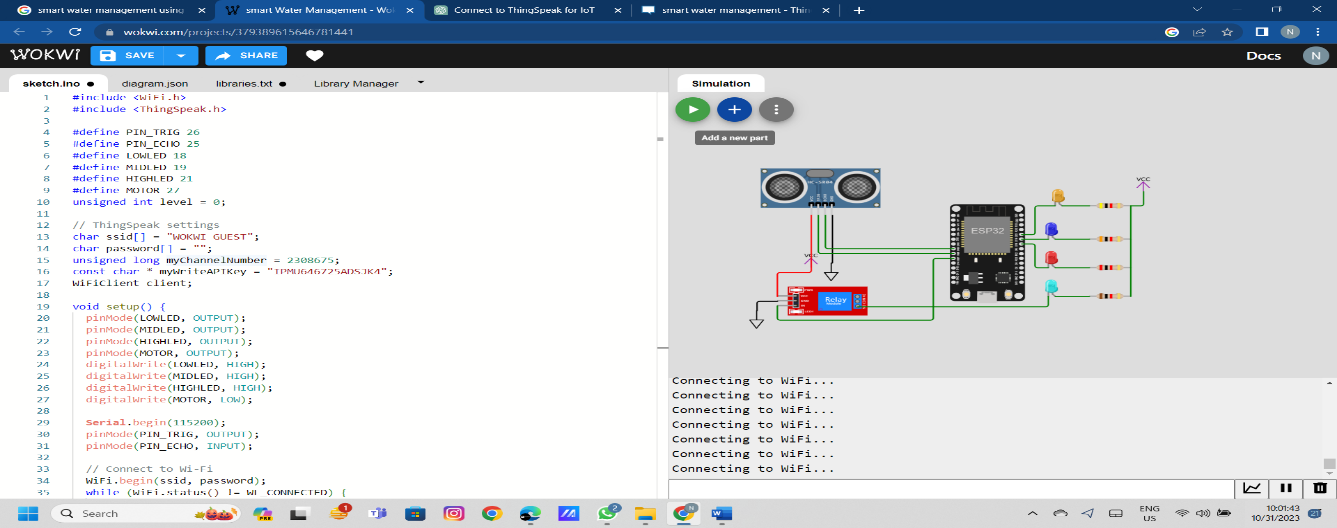
The code starts by including the necessary libraries: ESP8266WiFi.h, WiFiClient.h, and ThingSpeak.h. These libraries are required to connect to Wi-Fi and send data to Thing Speak using an ESP8266 board.



**2. Defining Pin and Variable Constants:**

The code defines pin numbers for the ultrasonic sensor, LEDs, and a motor, as well as a variable to store the distance measurement (level).

**Circuit diagram:**



**3. Configuring ThingSpeak Settings:**

You need to configure your Wi-Fi network credentials (ssid and password), ThingSpeak channel number (myChannelNumber), and Write API Key (myWriteAPIKey). You should replace the placeholders with your own values.

**4. Setup Function:**

In the setup function:

* Pins for LEDs, motor, and ultrasonic sensor are set as output or input.
* The LEDs are initially turned on, and the motor is turned off.
* Serial communication is initialized for debugging.
* Wi-Fi connection to your network is established, and you'll see status messages in the Serial Monitor.
* The ThingSpeak client is initialized.

**5. Loop Function:**

The loop function is where the main operation occurs, and it repeats continuously. It triggers the ultrasonic sensor to take a distance measurement by sending a short pulse on PIN\_TRIG.The code measures the time it takes for the echo signal to return and calculates the distance in centimeters and inches. The results are printed to the Serial Monitor. The distance measurement is stored in the level variable. Based on the distance, the code controls the LEDs and motor to indicate the proximity of an object. The distance measurements in centimeters and inches are sent to ThingSpeak using ThingSpeak.setField() and then sent to the ThingSpeak channel using ThingSpeak.writeFields(). The HTTP response code is checked, and the status of the data upload is printed to the Serial Monitor. There’s a delay of 1 second before the next measurement.

**6. Debugging and Testing:**

To debug and test the code, you can open the Arduino IDE's Serial Monitor. It will display the distance measurements and the status of data sent to ThingSpeak. This helps you ensure that the system is working as expected.

**7. ThingSpeak Output:**

After successfully uploading data to ThingSpeak, you can log in to your ThingSpeak account and access the data in your specified channel. The channel will display the distance measurements in real-time.

By following this code, you can create a system that measures the distance using an ultrasonic sensor, controls LEDs and a motor based on proximity, and logs the data to ThingSpeak for monitoring and analysis. Make sure to replace the placeholder values with your own Wi-Fi credentials and ThingSpeak details to make it work with your specific setup.

**Code:**

#include <WiFi.h>

#include <ThingSpeak.h>

#define PIN\_TRIG 26

#define PIN\_ECHO 25

#define LOWLED 18

#define MIDLED 19

#define HIGHLED 21

#define MOTOR 27

unsigned int level = 0;

// ThingSpeak settings

char ssid[] = "WOKWI GUEST";

char password[] = "";

unsigned long myChannelNumber = 2325279;

const char \* myWriteAPIKey = "1IHS3OWB1QSUX6XK";

WiFiClient client;

void setup() {

  pinMode(LOWLED, OUTPUT);

  pinMode(MIDLED, OUTPUT);

  pinMode(HIGHLED, OUTPUT);

  pinMode(MOTOR, OUTPUT);

  digitalWrite(LOWLED, HIGH);

  digitalWrite(MIDLED, HIGH);

  digitalWrite(HIGHLED, HIGH);

  digitalWrite(MOTOR, LOW);

**Serial**.begin(115200);

  pinMode(PIN\_TRIG, OUTPUT);

  pinMode(PIN\_ECHO, INPUT);

  // Connect to Wi-Fi

  WiFi.begin(ssid, password);

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

    delay(1000);

**Serial**.println("Connecting to WiFi...");

  }

**Serial**.println("Connected to WiFi");

  // Initialize ThingSpeak

  ThingSpeak.begin(client);

}

void loop() {

  // Start a new measurement:

  digitalWrite(PIN\_TRIG, HIGH);

  delayMicroseconds(10);

  digitalWrite(PIN\_TRIG, LOW);

  // Read the result:

  int duration = pulseIn(PIN\_ECHO, HIGH);

**Serial**.print("Distance in CM: ");

**Serial**.println(duration / 58);

**Serial**.print("Distance in inches: ");

**Serial**.println(duration / 148);

  level = (duration / 10);

  if (level < 100) {

    digitalWrite(LOWLED, LOW);

    digitalWrite(MOTOR, HIGH);

    digitalWrite(HIGHLED, HIGH);

    digitalWrite(MIDLED, HIGH);

  } else if ((level > 200) && (level < 400)) {

    digitalWrite(LOWLED, HIGH);

    digitalWrite(HIGHLED, HIGH);

    digitalWrite(MIDLED, LOW);

  } else if (level >= 400) {

    digitalWrite(HIGHLED, LOW);

    digitalWrite(MIDLED, HIGH);

    digitalWrite(LOWLED, HIGH);

    digitalWrite(MOTOR, LOW);

  }

  // Send data to ThingSpeak

  ThingSpeak.setField(1, duration / 58); // Distance in CM

  ThingSpeak.setField(2, duration / 148); // Distance in inches

  int httpCode = ThingSpeak.writeFields(myChannelNumber, myWriteAPIKey);

  if (httpCode == 200) {

**Serial**.println("Data sent to ThingSpeak successfully");

  } else {

**Serial**.print("Failed to send data to ThingSpeak, HTTP error code: ");

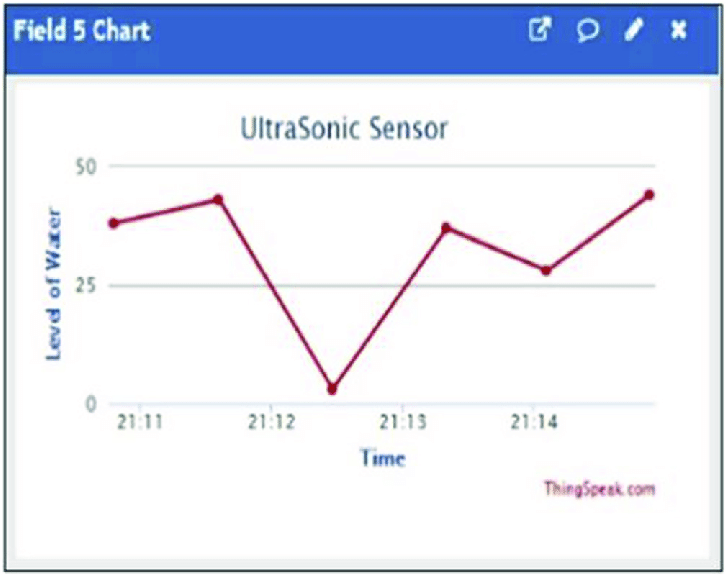
**Serial**.println(httpCode);

  }

  delay(1000); // Delay before the next measurement

}

**Output:**



**WEB DEVELOPMENT:**

**Html:**

This HTML code defines the structure of a webpage for displaying water level data. It includes a header with a title and a canvas element for rendering a dynamic line chart. The required CSS and JavaScript files are linked.HTML stands for Hyper Text Markup Language and is used to structure web content. In this HTML file, we define the basic structure of a web page.

* The <head> section includes meta information, such as character encoding and viewport settings.
* Inside the <body>, we have the main content of the webpage.
* The <header> element is used to create a header section with a title.
* The <canvas> element is employed to create a space for rendering the line chart. It has an id attribute set to "water Level Chart," which is used to target it in JavaScript.

**Code for html:**

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<link rel="stylesheet" href="styles.css">

<title>Random Water Level Data</title>

</head>

<body>

<header>

<h1>Random Water Level Data</h1>

</header>

<main>

<div class="chart-container">

<canvas id="waterLevelChart"></canvas>

</div>

</main>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script src="script.js"></script>

</body>

</html>

**CSS (styles.css):**

CSS stands for Cascading Style Sheets and is used for styling the appearance of web content.In this CSS file, we apply styling to elements of the HTML page. The body selector specifies styling for the entire webpage, including font family and text alignment. The header selector sets styling for the header section, including background color and text color. The h1 selector defines the appearance of the main title with a specific font size. The chart-container class styles the chart container, controlling its maximum width and margins.

**Code for css:**

body {

font-family: Arial, sans-serif;

text-align: center;

background-color: #f0f0f0;

}

header {

background-color: #333;

color: white;

padding: 10px;

}

h1 {

font-size: 24px;

}

.chart-container {

max-width: 80%;

margin: 0 auto;

}

**JavaScript (script.js):**

JavaScript is a versatile programming language used to add interactivity and functionality to web pages.In this JavaScript file, we utilize the Chart.js library to create a dynamic line chart.We define several variables to manage the chart and data. timestamps and water Levels arrays hold the data for the chart.The create Chart function initializes the Chart.js chart. It defines the type of chart (line chart), data labels, dataset properties, and chart options like time scaling for the X-axis.The generate Random Data function generates random data points for the last 10 seconds. It calculates timestamps and water level values for each data point, adds them to the arrays, and updates the chart accordingly.The generate Random Data function is called to initialize the chart with random data.While this code uses random data for demonstration, in a real application, you would replace the data generation logic with data fetched from sensors or a server.

**Code for java script:**

document.addEventListener("DOMContentLoaded", function () {

const ctx = document.getElementById("waterLevelChart").getContext("2d");

let chart;

let timestamps = [];

let waterLevels = [];

function createChart() {

chart = new Chart(ctx, {

type: "line",

data: {

labels: timestamps,

datasets: [{

label: "Water Level (cm)",

data: waterLevels,

borderColor: "blue",

fill: false,

}],

},

options: {

scales: {

x: {

type: "time",

time: {

unit: "second",

displayFormats: {

second: "HH:mm:ss",

},

},

},

},

},

});

}

function generateRandomData() {

const currentTime = new Date();

for (let i = 0; i < 10; i++) {

const newTime = new Date(currentTime);

newTime.setSeconds(currentTime.getSeconds() - (10 - i)); // Generate data for the last 10 seconds

const waterLevel = Math.random() \* 100;

timestamps.push(newTime);

waterLevels.push(waterLevel);

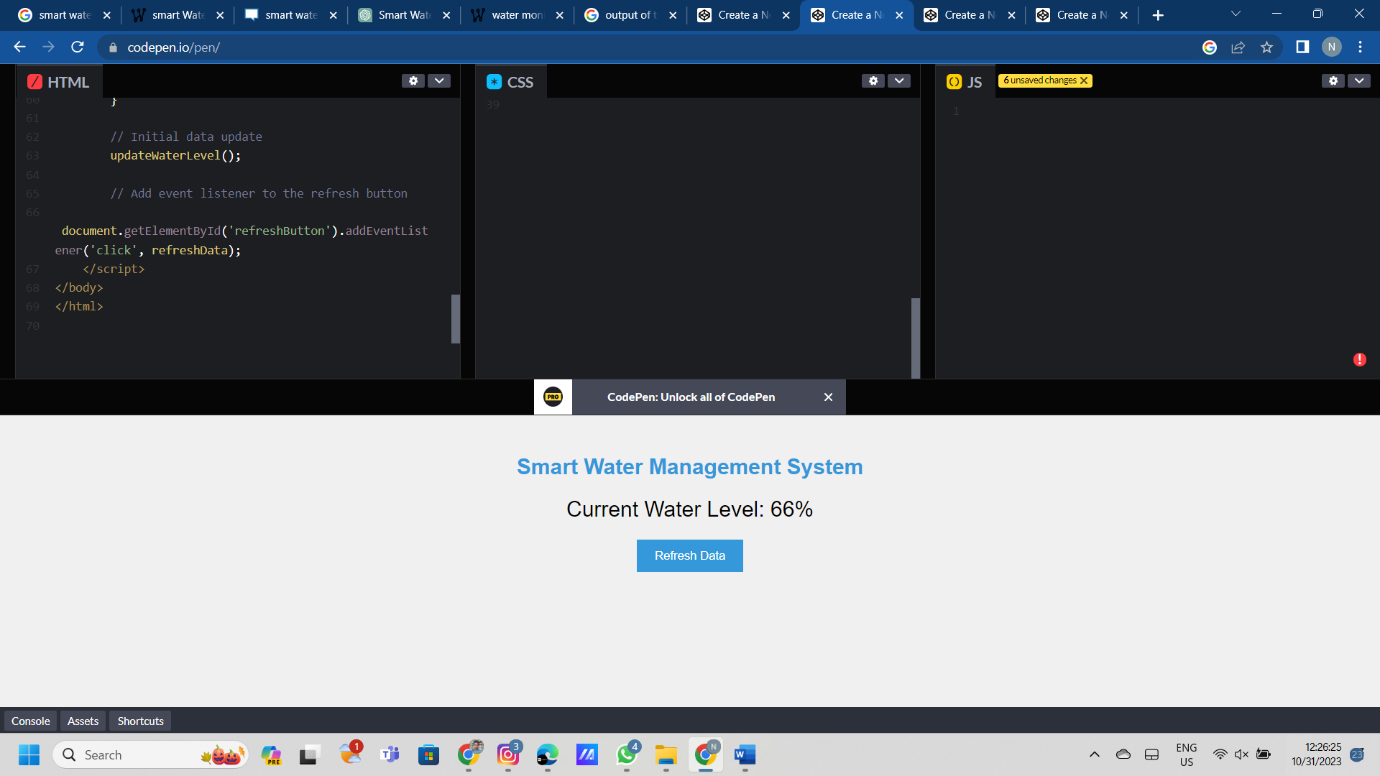
}

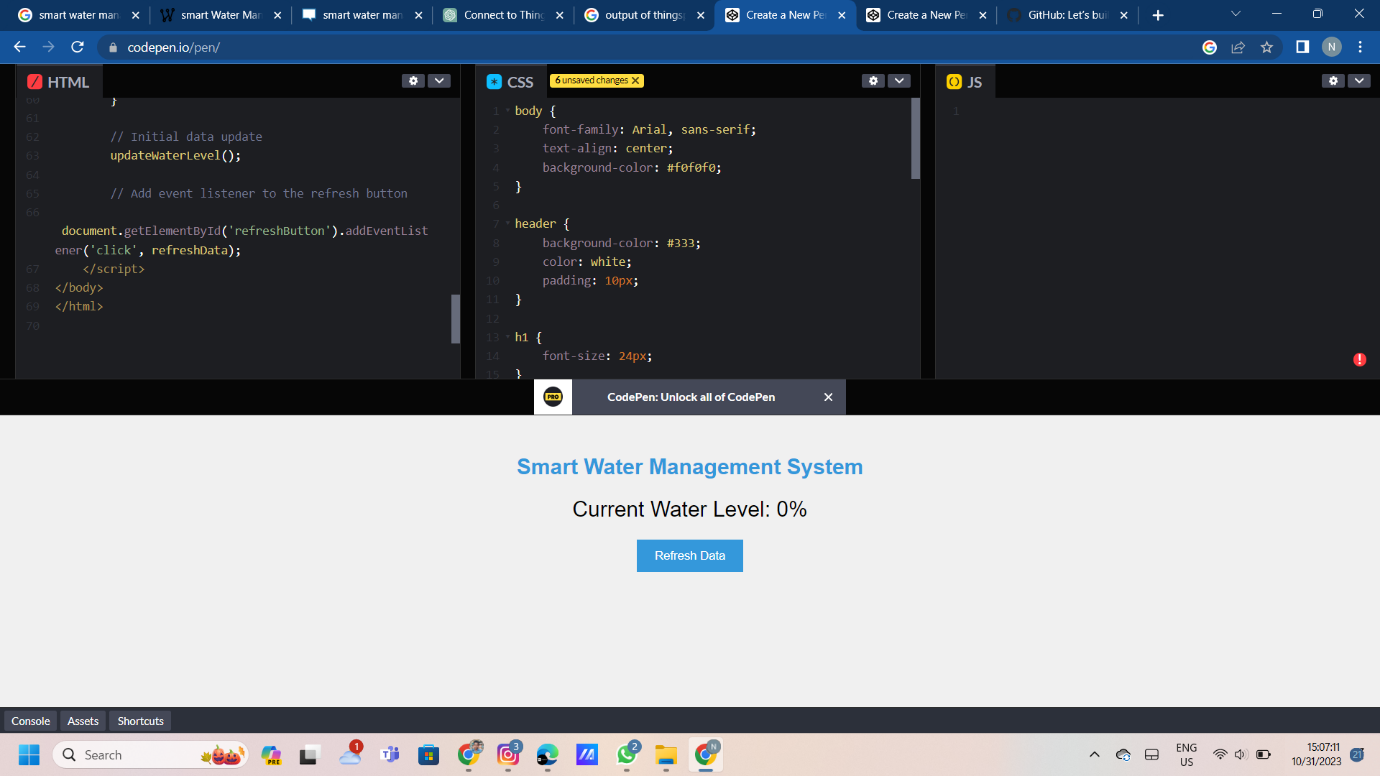
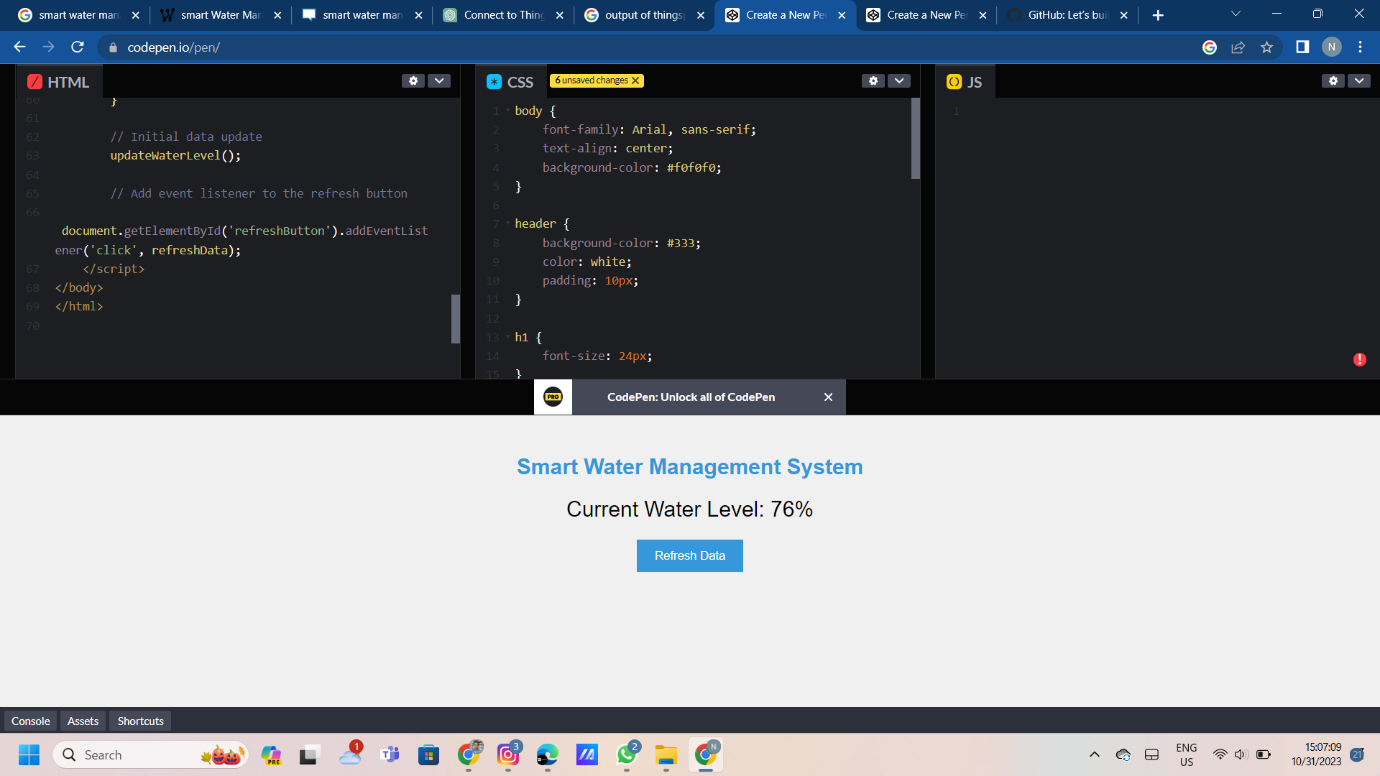
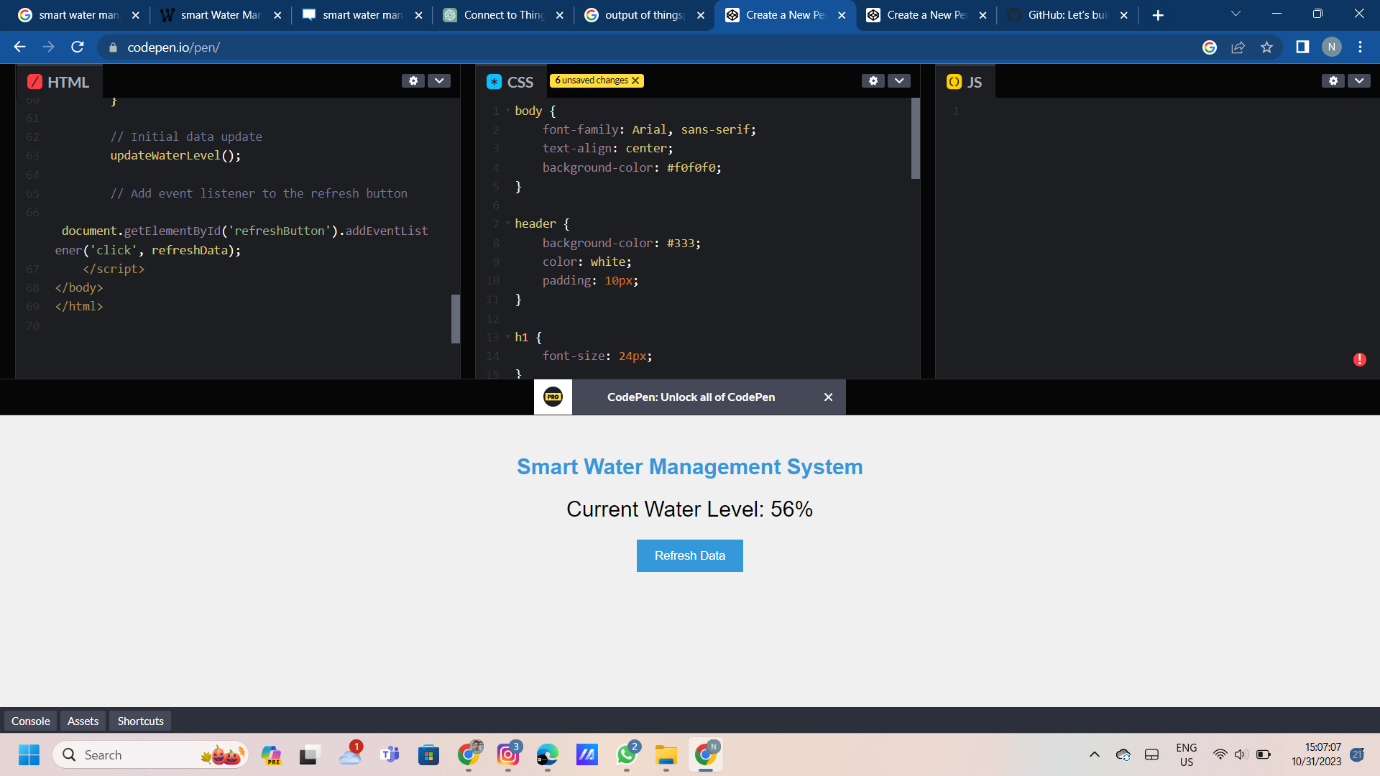
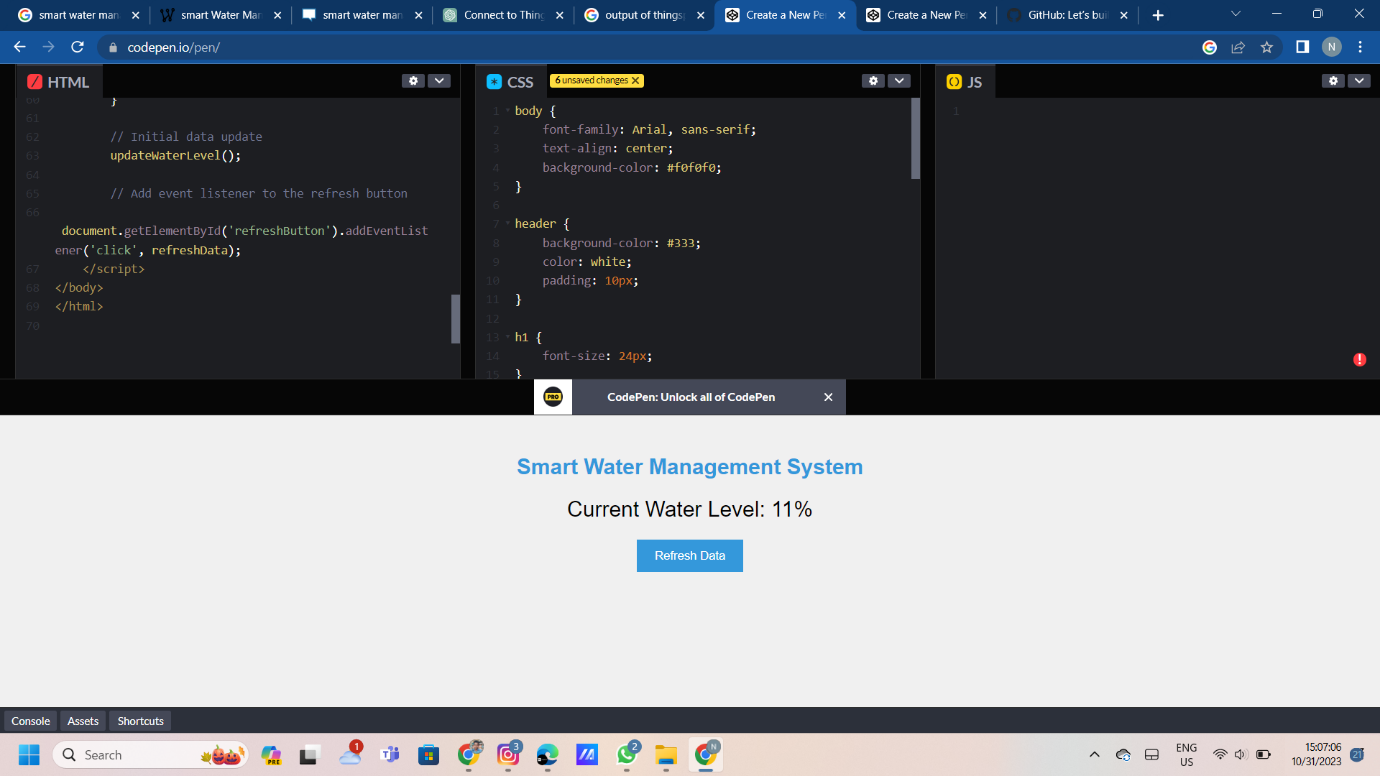
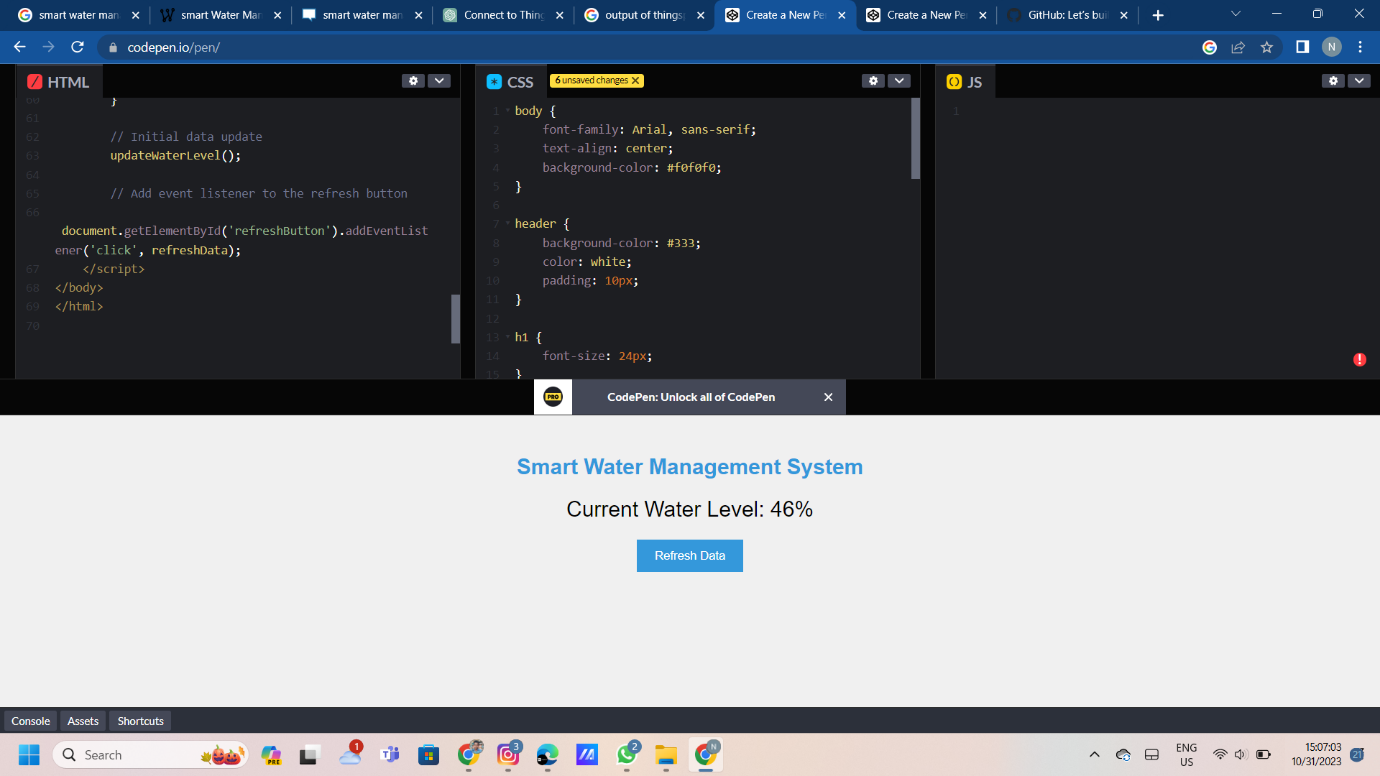
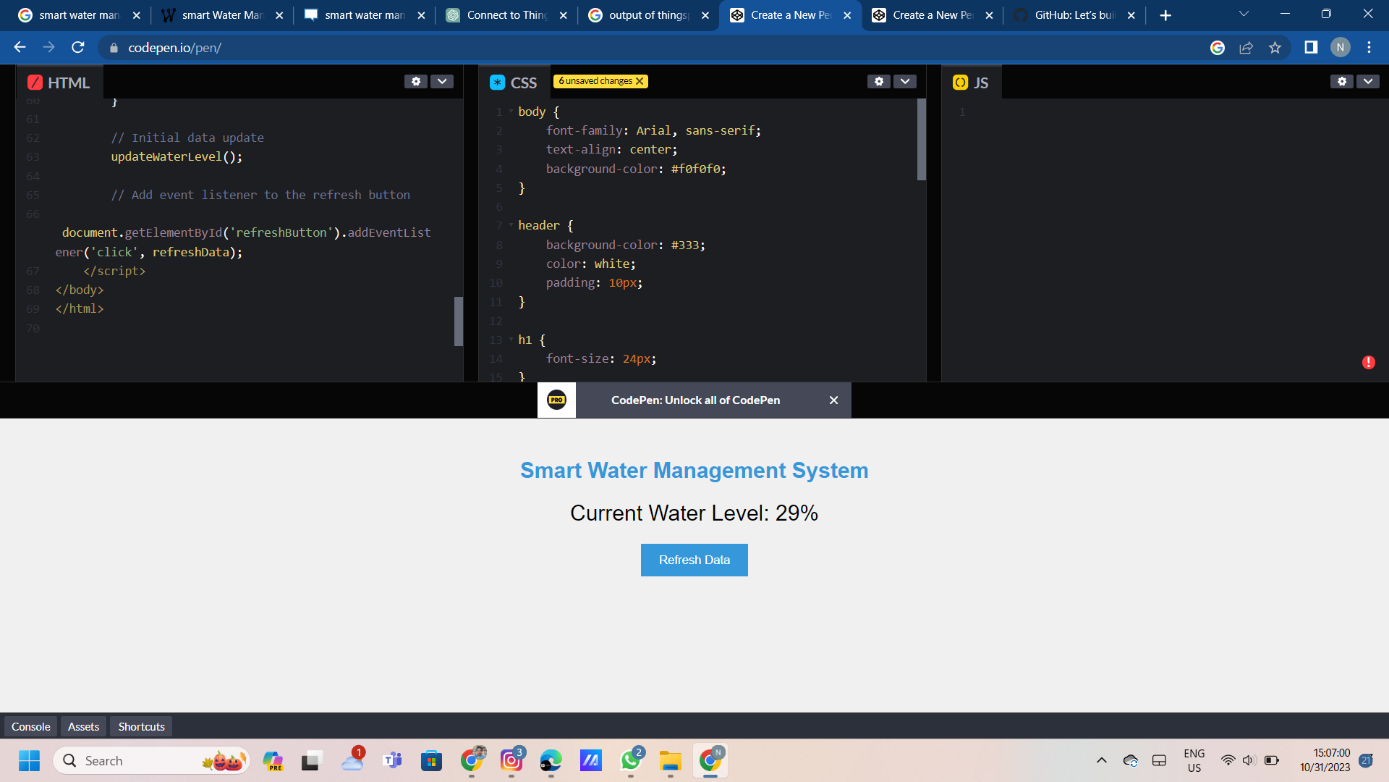
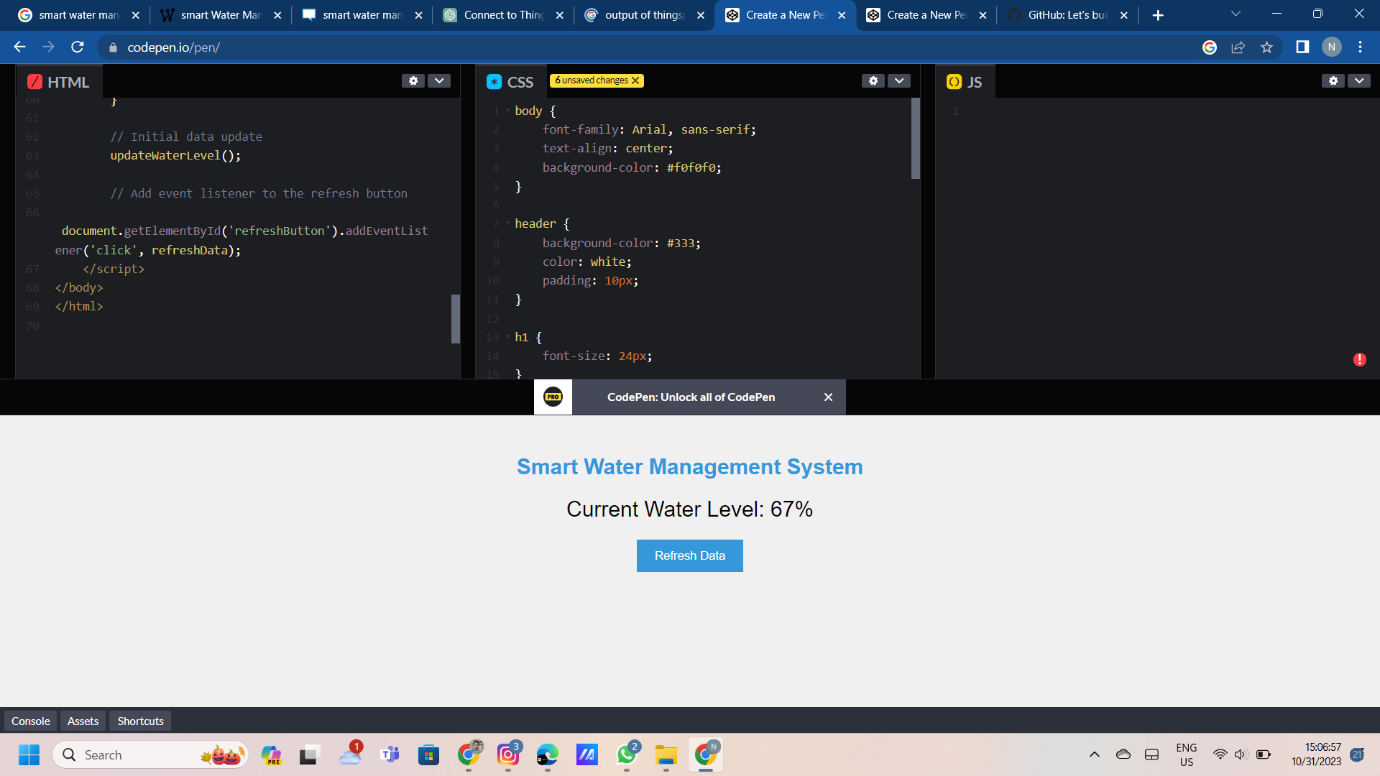
createChart();

}

generateRandomData();

**OUTPUT EXAMPLES:**





**CONCLUSION:**

The project described in the introduction focuses on smart water management using IoT technology and web development. It outlines the steps involved in designing a water level monitoring system that utilizes an ultrasonic sensor, an ESP8266 board, and ThingSpeak for data visualization. Additionally, it provides a basic web interface to display water level data in real-time using HTML, CSS, and JavaScript. In the circuit design phase, the code includes necessary libraries and defines constants and variables for pins and data storage. It configures Wi-Fi credentials and ThingSpeak settings, establishes Wi-Fi connectivity, and initializes ThingSpeak for data transmission. The loop function continuously measures water level, controls LEDs and a motor based on proximity, and sends data to ThingSpeak for monitoring. The web development aspect involves creating an HTML page with a title and a canvas element for rendering a dynamic line chart. CSS is applied to style elements, and JavaScript, with the help of the Chart.js library, is used to generate and display random water level data in the chart. It simulates real-time data for demonstration purposes, and in practice, actual sensor data would replace this simulation.