University of Prince Mugrin

College of Computer and Cyber Sciences

Department of Computer Science

CS487 - Internet of Things

Course Project – Semester I (Fall 2023 - 2024)

Zamzam Dispenser Monitoring

Team Members:

Abeer Osman | 4010368

Salwa Shama | 4010405

Sana Shama | 4010404

Instructor:

Prof. Mohamed Zayed

December 10, 2023

Table of Contents

CHAPTER 1- INTRODUCTION

1.1 Project Description

The aim of this project is to establish a comprehensive monitoring system for water levels in containers situated at Al Haramain, Al-Masajid Haram, and Al-Masajid an Nabawi, with a specific focus on Zamzam water. Our solution involves the utilization of ultrasonic sensors and ESP8266 devices that are attached to each container. These sensors accurately measure the water level, and the gathered data is then transmitted to a gateway. Subsequently, the data is forwarded to a cloud-based database for storage by using ESP8266 devices.

In addition to the data collection and analysis functionalities, we have included a practical feature in the system. To enhance monitoring, we have attached an LED indicator to each container. When the water level reaches a pre-set threshold, the LED illuminates, serving as a visual alert. This feature ensures that prompt attention and maintenance can be provided to the water level as needed.

To facilitate easy access to the collected data, we have developed a web page dashboard. This intuitive dashboard provides real-time updates on the water levels, enabling users to monitor the status at any given time. Consequently, the system ensures an efficient and uninterrupted water supply for various purposes within the premises, as it effectively manages and maintains appropriate water levels.

1.2 Problem Statement

The problems that we aim to address are the manual checking and hourly tours required to monitor and replace Zamzam dispensers. This process consumes valuable manpower that could be utilized more efficiently. By implementing an automated monitoring system with real-time data updates and visual alerts, we can optimize resource allocation and streamline the management of water levels. This solution will eliminate the need for frequent manual checks and tours, allowing personnel to focus on other essential tasks and making the entire process of serving Al Haramain visitors more efficient.

1.3 Project Goals

The project aims to implement an automated monitoring system for water levels in containers at Al Haramain, Al-Masjid Haram, and Al-Masjid an-Nabawi. The goals include optimizing resource allocation, streamlining water level management, and enhancing the efficiency of serving visitors to Al Haramain. As mentioned in the following:

1. **Implement an automated monitoring system:** The primary goal of the project is to establish an automated monitoring system for water levels in Zamzam dispensers at Al Haramain, Al-Masajid Haram, and Al-Masajid Nabawi. This involves the installation of ultrasonic sensors and ESP8266 devices on each container to accurately measure the water level.

2. **Enable real-time data updates**: Develop a system that can transmit the gathered data from the sensors to a gateway and subsequently to a cloud-based database for storage. This ensures that water level information is available in real-time, allowing for prompt action and maintenance as needed.

3. **Provide visual alerts for water level thresholds**: Install LED indicators on each container that illuminate when the water level reaches a pre-set threshold. This visual alert serves as a signal for personnel to take necessary actions to maintain appropriate water levels promptly.

4. **Develop a web page dashboard:** Create an intuitive web page dashboard that provides real-time updates on the water levels in Zamzam dispensers. The dashboard should be accessible to supervisors, enabling them to monitor the status of water levels at any given time. This facilitates efficient management and maintenance of water levels within the premises.

1.4 Target Users

This project is for the Al Haramain administration, specifically targeting supervisors of Zamzam water dispensers, which hold significant importance for the Muslim community. The goal of this project is to enhance access to Zamzam water by strategically placing millions of containers throughout Al Haramain. A dedicated workforce is needed to monitor and refill the containers, which is a complex and time-consuming task. By implementing advanced technology, we offer a solution to streamline container management and ensure a seamless provision of Zamzam water to all visitors, especially during peak seasons.

1.5 Project Benefits

The implementation of the Zamzam Dispenser Monitoring system in the Al Haramain brings forth a myriad of advantages including improved efficiency, cost savings, and a better experience for the millions of visitors to Al Haramain.

1. **Optimize resource allocation:** By automating the monitoring process and providing real-time data updates, the project aims to optimize resource allocation. This includes reducing the need for frequent manual checks and hourly tours, freeing up manpower to focus on other essential tasks within Al Haramain.

2. **Streamline water level management:** The project aims to streamline the management of water levels by implementing an automated monitoring system. This ensures an efficient and uninterrupted water supply for various purposes within the premises, improving the overall experience of visitors to Al Haramain.

3. **Data-Driven Decision Making:** The data collected from IoT devices that are attached to the container provides valuable insights into usage patterns and peak times. This information allows the Foundation to make informed decisions to enhance overall operational efficiency.

4. **Cost Reduction:** By efficiently managing container refills based on real-time data, the Foundation can reduce operational costs associated with unnecessary workforce deployment.

5. **Enhanced Visitor Experience:** With this project, the Al Haramain Foundation can ensure a continuous and reliable supply of Zamzam water to visitors. This improves the overall experience for millions of pilgrims, contributing to the positive reputation of the Foundation.

6. **Scalability:** As the number of visitors fluctuates, this IoT system can easily scale to accommodate the increased demand for the containers, ensuring that the container management system remains effective during both regular and peak periods.

CHAPTER 2- REQUIREMENTS AND DESIGN

2.1 Project Plan

To execute the water level monitoring system effectively, adherence to specific prerequisites is imperative. The fundamental element involves employing water level sensors with precision in measuring water levels within containers, selected based on container characteristics and the intended application. A microcontroller or microprocessor unit becomes essential for processing sensor data and transmitting it to a central server or user interface. Connectivity options, such as Wi-Fi or cellular modules, are integral for seamless real-time data transmission.

2.2 Project Requirements

2.2.1 Software Requirements

In the execution of our project, we harnessed a diverse array of cutting-edge software technologies that played pivotal roles in bringing our innovative concepts to life. These sophisticated tools not only facilitated the seamless integration of circuits designed for the project but also enabled us to efficiently capture, process, and transmit data between the ESP8266 sensors and the cloud infrastructure.

Among the key software technologies employed was Arduino IDE, serving as a versatile and user-friendly platform for developing and programming the embedded systems within our project. The IDE's intuitive interface and extensive library support streamlined the coding process, providing a conducive environment for the implementation of intricate functionalities on the ESP8266 sensors and state-of-the-art programming languages, such as C++, to develop robust and scalable code for our embedded systems.

Additionally, we leveraged cloud storage platforms, including google firebase, to establish a secure and reliable connection for real-time data exchange. In essence, the amalgamation of these advanced software technologies not only empowered the functionality of our IoT devices but also underscored the sophistication of our project implementation.

Functional Requirements

The user shall be able to view the information that is related to water level	0.1
The ultrasonic sensor shall be able to read the level of water inside the container	0.2
The LED shall be able to turn-on when the level of water in distance is less than the half of the container	0.3
The LED shall be able to turn-off when the level of water in distance is greater than the half of the container	0.4
The ESP8266 sender shall be able to gather all updated data from ultrasonic sensor	0.5
The ESP8266 sender shall be able to send data to ESP8266 receiver (Hub)	0.6
The ESP8266 receiver (Hub) shall be able to receive data from other ESP8266 senders	0.7
The ESP8266 receiver (Hub) shall be able to send data to cloud	0.8

Table 2.1 – Functional Requirements

Non-Functional Requirements

The system should provide a fast and responsive user interface	1.1
The system shall be operational and available for 24/7 over 12-mounths	1.2
The system shall be up to date 24/7 over 12 months	1.3
The system should perform well as the number of containers increases, ensuring scalability without compromising performance.	1.4
The system should be able to adapt to different container sizes and types without requiring significant modifications.	1.5

Table 2.2 – Non-Functional Requirements
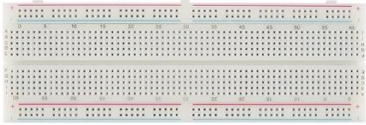
2.2.2 Hardware Requirements

Name of technology	Its illustration
Breadboard	
Ultrasonic sensor	
ESP8266	
LEDs	
Resistors	
Wires	

Table 2.3 Hardware Requirements

2.2.3 Future Plans

We assume that expanding the functionality of this project broadens its scope, targeting a diverse range of users across residential and industrial sectors. In residential settings, homeowners and property managers can utilize the system to monitor water levels in containers like rainwater harvesting tanks and basement sump pumps. This application ensures timely awareness of potential overflow or depletion, mitigating water damage and promoting efficient resource management. On an industrial scale, businesses engaged in water storage, treatment, or distribution stand to benefit by integrating this technology into their operations. The project caters to users valuing real-time monitoring, empowering them to make informed decisions based on accurate water level data.

The implementation of the water level monitoring system yields a multitude of advantages. Firstly, it enhances safety measures by preventing overflow or depletion incidents, thereby reducing the risk of property damage and associated repair costs. Secondly, it promotes water conservation, enabling users to utilize water more efficiently and align with principles of sustainability and environmental responsibility. Additionally, the system's real-time data facilitates proactive maintenance, ensuring prolonged water storage through informed decision-making. Ultimately, the project equips users with valuable information to manage resources effectively, reduce the environmental impact of water activities, and save time, money, and effort.

2.3 Project Architecture

The project architecture solution is decomposed into four main parts as shown in Figure 2.1. The first part consists of a group of water containers associated with ultrasonic sensors to measure the water level in the containers and LEDs as actuators to perform actions in the environment under certain conditions. If the water level reaches below a certain threshold, the LED is turned on. The ultrasonic sensor sends the readings to the hub (ESP8266), which is the second part, using the ESP NOW protocol. This is considered good practice to save sensor power compared to using the HTTP protocol, which consumes a significant amount of power. The ESP8266 is chosen here for its lower power consumption compared to Raspberry Pi and suitability for real-time applications, as it has no operating system.

Next, the readings are sent from the hub to the cloud to be stored in the database (Firebase) using the HTTP protocol over Wi-Fi. This is feasible in the development area where there is sufficient power supply to charge the hub when needed, as it consumes much power during the transmission to the cloud. This process constitutes the third part of the system, and using the HTTP protocol seems to be a suitable choice here.

The client, in this case, is a website that retrieves data from the cloud to present and update it whenever there is a change in the database. The following picture shows that:
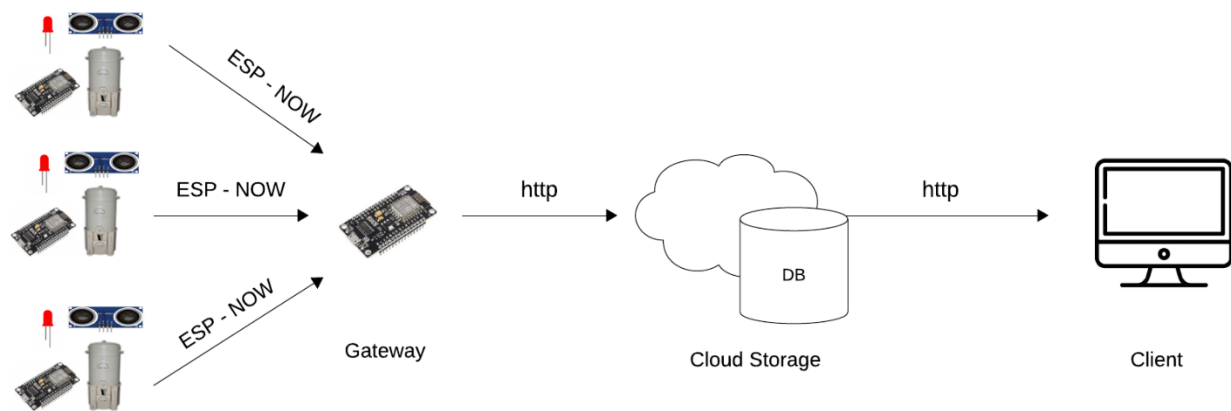
Figure 2.1 – System Architecture

CHAPTER 3- PROJECT IMPLEMENTATION

3.1 Project Outcomes

The dynamic behavior of the system is as follows:

1-The ultrasonic sensor will continuously read the water level in the water container. If the level is below a certain threshold, the LED will turn on; otherwise, the LED remains off.

2-The dashboard on the client receives water level readings. If it detects a level lower than a certain threshold, an alarm will appear on the dashboard; otherwise, the dashboard will appear normally.
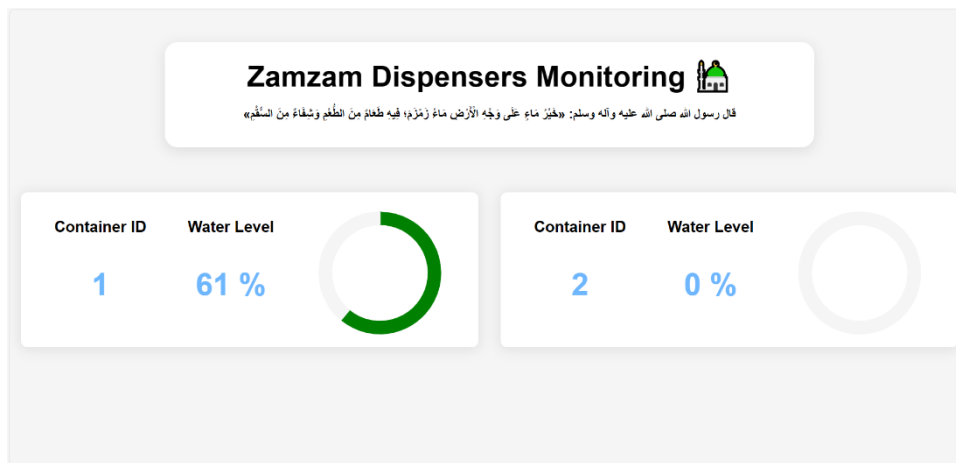

Zamzam Dispensers Monitoring 🕋

قال رسول الله صلى الله عليه وآله وسلم: «خَيْرُ ماءٍ على وَجْهِ الأَرْضِ ماءُ زَمْزَمَ؛ فيه طَعامٌ من الطُّعْمِ وشفاءٌ من السُّقْمِ»

Container ID	Water Level		Container ID	Water Level
1	61 %		2	0 %

Figure 3.1 - Client UI

3.2 Project Implementation

3.2.1 Sensor Node

```
LEDs_right_to_left.ino
 1    //These lines include the necessary libraries for the ESP8266 WiFi module
 2    //and the ESP-NOW communication protocol.
 3    #include <ESP8266WiFi.h>
 4    #include <espnow.h>
 5
 6    //RECEIVER MAC Address
 7    uint8_t broadcastAddress[] = {0xE8, 0xDB, 0x84, 0xE7, 0x98, 0xE3};
 8
 9    // my Board ID (Container2)
10    #define BOARD_ID 2
11
12    // This defines a structure (struct_message) that represents the data to be sent.
13    //It includes an ID and a distance value.
14    // Must match the receiver structure
15    typedef struct struct_message {
16        int id;
17        float distance;
18    } struct_message;
19
20    // Create a struct_message called myData to store variables to be sent
21    struct_message myData;
22
23    //These lines declare variables for storing pulse duration, distance, and percentage.
24    //It also defines pin assignments for the ultrasonic sensor trigger (TRIG_PIN), echo (ECHO_PIN), and the LED pin (LED_PIN).
25    double duration; // Variable to store the duration of the pulse
26    float distance;// Variable to store the calculated distance
27    #define TRIG_PIN D7  // Trig pin of ultrasonic sensor
28    #define ECHO_PIN D2  // Echo pin of ultrasonic sensor
29    #define LED_PIN D6  // Low water level LED pin
30
31    //These lines declare variables for tracking the last time the measurements were taken
32    //and set a delay of 10 seconds between measurements.
33    unsigned long lastTime = 0;
34    unsigned long timerDelay = 10000;
```

Figure 3.2 (a) – Sensor Node Code

```
35
36    // Callback function (OnDataSent) whenthat will be called when data is sent using ESP-NOW.
37    // It prints the delivery status (success or fail)
38    void OnDataSent(uint8_t *mac_addr, uint8_t sendStatus) {
39      Serial.print("\r\nLast Packet Send Status: ");
40      if (sendStatus == 0){
41        Serial.println("Delivery success");
42      }
43      else{
44        Serial.println("Delivery fail");
45      }
46    }
47    //This is the setup() function. It initializes the serial monitor,
48    // sets up the ESP-NOW communication, registers the callback function,
49    // adds the receiver device as a peer, and sets up the pin modes for the ultrasonic sensor and LED.
50    void setup() {
51      // initializes the serial communication with a baud rate of 9600.
52      Serial.begin(9600);
53
54      // Set device as a Wi-Fi Station
55      WiFi.mode(WIFI_STA);//sets the device as a Wi-Fi station, connect the ESP32 module to a WiFi access point
56      WiFi.disconnect();//disconnects from any existing Wi-Fi network
57
58      // Init ESP-NOW
59      if (esp_now_init() != 0) {
60        Serial.println("Error initializing ESP-NOW");
61        return;
62      }
63      // Set ESP-NOW role, sets the device role as a controller to initiating communication and sending data.
64      esp_now_set_self_role(ESP_NOW_ROLE_CONTROLLER);
65
66      // Once ESPNow is successfully init, we will register for Send callback(CB) to
67      // get the status of Transmitted packet
68      esp_now_register_send_cb(OnDataSent);
```

Figure 3.2 (b) – Sensor Node Code

```
69
70      // Register peer
71      esp_now_add_peer(broadcastAddress, ESP_NOW_ROLE_SLAVE, 1, NULL, 0);
72
73      pinMode(TRIG_PIN, OUTPUT); // Set pin D7 as an output for the trigger signal
74      pinMode(ECHO_PIN, INPUT); // Set pin D2 as an input for the echo signal
75      pinMode(LED_PIN, OUTPUT); // Set pin D6 as an output for the LED
76    }
77
78    void loop() {
79      if ((millis() - lastTime) > timerDelay) {
80        digitalWrite(TRIG_PIN, HIGH);
81        delayMicroseconds(10);
82        digitalWrite(TRIG_PIN, LOW);
83
84        duration = pulseIn(ECHO_PIN, HIGH);
85        distance = duration * 0.034 / 2;
86
87        // Set values to send
88        myData.id = BOARD_ID;
89        myData.distance = distance;
90
91        // Send message via ESP-NOW
92        esp_now_send(0, (uint8_t *) &myData, sizeof(myData));
93
94        // Control the LED based on the water level
95        if (distance > 5) {
96          digitalWrite(LED_PIN, HIGH); // Turn on the LED connected to pin D6
97        } else {
98          digitalWrite(LED_PIN, LOW); // Otherwise, turn off the LED
99        }
100
101       Serial.print("distance: "); // Print the label "percentage" to the serial monitor
102       Serial.println(distance); // Print the calculated percentage to the serial monitor
103       lastTime = millis();
104     }
```

Figure 3.2 (c) – Sensor Node Code

3.2.2 Gateway

```
1
2    // Include the ESP8266WiFi library
3    #include <ESP8266WiFi.h>
4    // Include the ESP-NOW library
5    #include <espnow.h>
6    // Include the Firebase ESP8266 library
7    #include <Firebase_ESP_Client.h>
8
9    // Define the Wi-Fi credentials
10   #define WIFI_SSID "AndroidAP40AA"
11   #define WIFI_PASSWORD "cukj2015"
12
13   // Define the Project Database URL and Firebase API Key
14   #define DATABASE_URL "https://iotproject-2b179-default-rtdb.asia-southeast1.firebasedatabase.app/"
15   #define API KEY "AIzaSyB75yW1LT22sxdfmZFl9Q39jrSIQi39-aE"
```

Figure 3.3 (a) – Gateway Code (Project Libraries)

```
18   typedef struct struct_message {
19       int id;
20       float distance;
21   } struct_message;
22
23   // Declare a variable to save received data
24   struct_message myData;
25   // Declare two elements that store messages received from two different boards
26   struct_message board1;
27   struct_message board2;
28   // Declare an array of boards with a size of 2
29   struct_message boardsStruct[2] = {board1, board2};
30
31   // Declare a FirebaseData object to handle Firebase operations, like: reading data, writing data, establish Firebase connection
32   FirebaseData fbdo;
33   // Declare a FirebaseAuth object for user authentication (email, and password)
34   // two email and password  are empty, going with an anonymous user
35   FirebaseAuth auth;
36   // Declare a FirebaseConfig object to store Firebase configuration (Project Database URL & Firebase API Key)
37   FirebaseConfig config;
38   // Declare a boolean variable to track the success of signup operation
39   bool signupOK = false;
40   // Declare a boolean variable to track if new data has been received
41   bool newDataReceived = false;
```

Figure 3.3 (b) – Gateway Code (Variable Declaration)

```
46   void OnDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {
47       // Declare a variable to store the MAC address as a string
48       char macStr[18];
49       Serial.print("Packet received from: ");
50       //Convert the MAC address to a string representation
51       //Snprintf is used to format the MAC address into the macStr array
52       //The "02" specifies that each byte should be represented by two characters (digits).
53       //The "x" indicates that the byte should be formatted as a hexadecimal number.
54       snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x",
55                mac_addr[0], mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);
56       Serial.println(macStr);
57       // Copy the received data into the `myData` structure
58       memcpy(&myData, incomingData, sizeof(myData));
59       Serial.printf("Board ID %u: %u bytes\n", myData.id, len);
60       // Update the corresponding board's ID and percentage in the `boardsStruct` array
61       boardsStruct[myData.id-1].id = myData.id;
62       boardsStruct[myData.id-1].distance = myData.distance;
63       Serial.printf("x value: %d \n", boardsStruct[myData.id-1].distance);
64       Serial.println();
65       // Set the flag to indicate that new data has been received
66       newDataReceived = true;
67   }
```

Figure 3.3 (c) – Gateway Code (Callback Function)

```
69  void setup() {
70    // Initialize the serial communication
71    Serial.begin(115200);
72    // Set the Wi-Fi mode to station mode
73    //In station mode, the device acts as a client and connects to an existing Wi-Fi network rather than creating its own network.
74    //This allows the device to connect to the internet.
75    WiFi.mode(WIFI_STA);
76    // Disconnect from any previous Wi-Fi connection.
77    WiFi.disconnect();
78    // Init ESP-NOW
79    // Initialize ESP-NOW and check for errors
80    if (esp_now_init() != 0) {
81      Serial.println("Error initializing ESP-NOW");
82      return;  }
83    // Connect to Wi-Fi using the provided credentials
84    WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
85    Serial.print("Connecting to Wi-Fi");
86    // Wait for the Wi-Fi connection to be established
87    while (WiFi.status() != WL_CONNECTED){
88      Serial.print(".");
89      // Delay for 300 milliseconds before checking the connection status again
90      delay(300);
91  }
```

Figure 3.3 (d) – Gateway Code (setup Function -1)

```
92    Serial.println();
93    // Print a message to indicate that the device is successfully connected to Wi-Fi
94    Serial.print("Connected with IP: ");
95    // Print the local IP address assigned to the device
96    Serial.println(WiFi.localIP());
97    Serial.println();
98
99    /* Assign the api key (required) */
100   config.api_key = API_KEY;
101   /* Assign the RTDB URL (required) */
102   config.database_url = DATABASE_URL;
103
104   /* Sign up */
105   // Sign up to Firebase using the provided credentials and with an anonymous user
106   if (Firebase.signUp(&config, &auth, "", "")){
107     // If the sign-up is successful
108     Serial.println("ok");
109     signupOK = true;
110   }
111   else{
112     // If the sign-up fails, print the error message
113     Serial.printf("%s\n", config.signer.signupError.message.c_str());
114   }
```

Figure 3.3 (e) – Gateway Code (setup Function -2)

```
117   Firebase.begin(&config, &auth);
118   // Enable automatic reconnection to Wi-Fi
119   Firebase.reconnectWiFi(true);
120
121   // Set the ESPNow role as a receiver (client)
122   esp_now_set_self_role(ESP_NOW_ROLE_SLAVE);
123   // Once ESPNow is successfully initialized, register the callback function to receive data
124   esp_now_register_recv_cb(OnDataRecv);
125 }
```

Figure 3.3 (f) – Gateway Code (setup Function -3)

```
127  void loop(){
128    // Check if Firebase is ready and sign up was successful
129    if (Firebase.ready() && signupOK) {
130      // Check if new data is received
131      if (newDataReceived) {
132        // Send data to Firebase Realtime Database
133        String DBpath = "Containers/Container" + String(boardsStruct[myData.id-1].id) + "/waterLevel";
134        if (Firebase.RTDB.setFloat(&fbdo, DBpath , boardsStruct[myData.id-1].distance )) {
135          Serial.println("Data send successful");
136          Serial.println("PATH: " + fbdo.dataPath());
137          Serial.println("TYPE: " + fbdo.dataType());
138        }
139        else {
140          Serial.println("Data send failed");
141          Serial.println("REASON: " + fbdo.errorReason());
142        }
143        // Reset the flag for "new data is received"
144        newDataReceived = false;
145      }
146    }
147  }
148
```

Figure 3.3 (g) – Gateway Code (Loop Part)

3.2.3 Client

```
<!DOCTYPE html>
<html lang="en">
    <head>
        <!-- Meta tags for character set and viewport -->
        <meta charset="utf-8">
        <meta name="viewport" content="width=device-width, initial-scale=1">

        <!-- Title of the page -->
        <title>Ultrasonic Readings</title>

        <!-- Link to the external stylesheet for styling -->
        <link rel="stylesheet" href="style.css">

        <!-- Scripts for Chart.js and Firebase -->
        <script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
        <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-app.js"></script>
        <script src="https://www.gstatic.com/firebasejs/8.6.8/firebase-database.js"></script>
    </head>
```

15

```html
<body>
    <!-- Heading for the data section -->
    <div class="data-head">
        Zamzam Dispensers Monitoring 🏠
        <p>
            «قَالَ رَسُولُ اللهِ صَلى اللهِ عَلَيهِ وَآلهِ وَسَلَم: «خَيْرُ مَاءٍ عَلَى وَجْهِ الأَرْضِ مَاءُ زَمْزَمَ؛ فِيهِ طَعَامٌ مِنَ الطُّعْمِ وَشِفَاءٌ مِنَ السُّقْمِ
        </p>
    </div>

    <!-- Container for the data reading-->
    <div id="data-container-wrapper">
        <!-- Data containers will be dynamically added here -->
    </div>

    <!-- Script for Firebase database connection -->
    <script src="DB_connection.js"></script>
</body>
</html>
```

Figure 3.4 (a) - Client Code

```css
/* Set the background color, font family, and reset margin and padding */
body {
    background-color: ▢#F5F5F5;
    font-family: Arial, sans-serif;
    margin: 0;
    padding: 0;
}

/* Styling for the data header */
.data-head {
    margin: auto;
    width: 65%;
    text-align: center;
    font-size: 45px;
    font-weight: bold;
    margin: 50px auto;
    padding: 20px;
    background-color: ▢#FFF;
    box-shadow: 0 5px 20px ▢rgba(0, 0, 0, 0.1);
    border-radius: 20px;
}
```

16

```css
.data-head p{
    font-size: 20px;
    direction: rtl; /* Set text direction to right-to-left */
}

/* This wrapper contains the data containers */
#data-container-wrapper {
    display: flex;
    flex-wrap: wrap;
}

/* This class represents a data container */
.data-container {
    display: flex;
    flex-wrap: wrap;
    justify-content: space-around;
    width: 45%;
    margin: 20px auto;
    padding: 20px;
    background-color: #FFF;
    box-shadow: 0 2px 20px rgba(0, 0, 0, 0.1);
    border-radius: 10px;
}

/* Hover effect for the data container */
.data-container:hover {
    box-shadow: 0 8px 30px rgba(0, 0, 0, 0.2);
}


/* Hover effect for the data container */
.data-container:hover {
    box-shadow: 0 8px 30px rgba(0, 0, 0, 0.2);
}

/* Styling for the data items */
.data-item {
    text-align: center;
}

.data-item h2 {
    font-size: 24px;
    font-weight: bold;
    margin-bottom: 10px;
}

.data-item p {
    font-size: 48px;
    font-weight: bold;
    color: #6EB7FF;
}
```

Figure 3.5 (b) - Client Code

```javascript
// Firebase configuration
const firebaseConfig = {
    apiKey: "AIzaSyB75yW1LT22sxdfmZFl9Q39jrSIQi39-aE",
    authDomain: "iotproject-2b179.firebaseapp.com",
    databaseURL: "https://iotproject-2b179-default-rtdb.asia-southeast1.firebasedatabase.app",
    projectId: "iotproject-2b179",
    storageBucket: "iotproject-2b179.appspot.com",
    messagingSenderId: "640293022864",
    appId: "1:640293022864:web:1eeea0e864a6608349fc15",
    measurementId: "G-RTWVMMFBT2"
};
```

```javascript
    // Initialize Firebase and declare the database variable
    firebase.initializeApp(firebaseConfig);
    var database = firebase.database();

    // Fetch data from Firebase
    function fetchData() {
        // Get reference to the 'Containers' node in the database
        var dataRef = database.ref('Containers');
        // Listen for value changes in the 'Containers' node
        dataRef.on('value', function(snapshot) {
            document.getElementById('data-container-wrapper').innerHTML = "";
            // Iterate over each child snapshot
            snapshot.forEach(function(childSnapshot) {
                var containerData = childSnapshot.val();
                createDataContainer(containerData); // Create a data container for the containerData
                createCircularChart(containerData); // Create a circular chart for the containerData
            });
        }, function(error) {
            console.error('Error fetching data:', error);
        });
    }


// Create a new data container div
function createDataContainer(containerData) {
    // Create a new data container div
    var newDataContainer = document.createElement('div');
    newDataContainer.className = 'data-container';

    // Create data item for Container ID
    var idItem = document.createElement('div');
    idItem.className = 'data-item';
    idItem.innerHTML = '<h2>Container ID</h2><p class="value" id="ID">' + containerData.ID + '</p>';
    newDataContainer.appendChild(idItem);

    // Create data item for Water Level
    var waterLevelItem = document.createElement('div');
    waterLevelItem.className = 'data-item';
    waterLevelItem.innerHTML = '<h2>Water Level</h2><p class="value" id="water_value">' + parseInt((10.455 - (containerData.waterLevel))/1
    newDataContainer.appendChild(waterLevelItem);

    // Create chart container for water level chart
    var waterChartItem = document.createElement('div');
    waterChartItem.className = 'chart-container';
    waterChartItem.innerHTML = '<canvas class="chart" id="waterChart' + containerData.ID + '" width="200" height="200"></canvas>';
    newDataContainer.appendChild(waterChartItem);

    // Append the new data container to the main container
    document.getElementById('data-container-wrapper').appendChild(newDataContainer);
}
```

```
// Create circular chart for water level
function createCircularChart(containerData) {
    var gradientColors = (containerData.waterLevel >= 4.5) ? ['red', '#F5F5F5'] : ['green', '#F5F5F5'];

    var ctx = document.getElementById('waterChart' + containerData.ID).getContext('2d');
    var waterChart = new Chart(ctx, {
        type: 'doughnut',
        data: {
            datasets: [{
                data: [parseInt((10.455 - (containerData.waterLevel))/10.455*100), 100 - parseInt((10.455 - (containerData.waterLevel))/10.
                backgroundColor: gradientColors, // Set the background color using the gradientColors based on the condition.
                borderColor: gradientColors, // Set the border Color using the gradientColors based on the condition.
                borderWidth: 2 //The borderWidth is set to 2 pixels.
            }]
        },
        options: {
            cutout: '80%', // Set the size of the center hole in the doughnut chart
            responsive: true, // Make the chart responsive to resize
            maintainAspectRatio: false, // Do not maintain aspect ratio between the height and width
        }
    });
}

// fetch data
fetchData();
```

Figure 3.6 (c) - Client Code

CHAPTER 4- CONCLUSION

4.1 Summary

In conclusion, we applied our knowledge of the Internet of Things to our project. We began by searching and reading about different sensors, microcontrollers, and protocols that could help us in the project; ultimately, we chose to use an ultrasonic sensor, LEDs as actuators, ESP8266 as a microcontroller, and ESP NOW and HTTP as protocols. Many reasons for these choices are mentioned in the report.

Throughout the project, we applied the techniques and concepts learned in our lectures, enhancing our understanding through hands-on application. Towards the end of the project, we now feel confident in determining when to use which IoT component architecture and where. Overall, this project provided us with practical experience and solidified our understanding of the course material.

Reference

[1] "ESP-NOW: Receive Data from Multiple ESP8266 Boards (many-to-one) | Random Nerd Tutorials," Jun. 12, 2020. https://randomnerdtutorials.com/esp-now-many-to-one-esp8266-nodemcu/ (accessed Dec. 09, 2023).

[2] "ESP8266: Getting Started with Firebase (Realtime Database) | Random Nerd Tutorials," Sep. 18, 2021. https://randomnerdtutorials.com/esp8266-nodemcu-firebase-realtime-database/

[3] "How to get Realtime Data from Firebase to your Web Application," www.youtube.com. https://www.youtube.com/watch?si=rvQKDiSTAZys7pni&v=B10HWeXouIg&feature=youtu.be (accessed Dec. 09, 2023).