UNITED INTERNATIONAL UNIVERSITY (UIU)

Dept. of Computer Science & Engineering

Course No: CSE 4326

Course Title: Microprocessors and Microcontrollers Laboratory

Experiment 2: Wifi Communication and building IoT-based systems Using XAMPP and Arduino IoT Cloud

Objectives:

The objective of this lab experiment is to set up an ESP32 board to build IoT-based systems. We Shall divide this experiment in the following manners:

i) We will monitor temperature and humidity data using a DHT11 sensor that will be connected to ESP32. Then, the real time data shall be sent to a local server, called “XAMPP” where this data will be stored in a database. Additionally, we will display the data on a local web page.

Students shall perform similar gas data monitoring using this “XAMPP” server and interface led buttons to control using the IoT cloud.

ii) Next, students will monitor the gas sensor data and control an LED using the Arduino Cloud IoT platform. This comprehensive exercise encompasses multiple key steps, starting with the setup of the ESP32 device on the Arduino Cloud IoT platform, which involves device registration and the acquisition of essential credentials. Subsequently, students create a virtual "Thing" on the platform, linking it to the ESP32 and defining variables for data exchange, namely Gas_ppm and LED. Ensuring secure data transmission, students configure network credentials and incorporate the generated secret key.

The lab progresses to programming the ESP32, enabling it to process gas sensor data and manage the LED using digital control logic. Finally, students design a user-friendly dashboard within the XAMPP and Arduino Cloud IoT platform, integrating widgets for real-time sensor data visualization and remote LED control. Final outcomes of this experiment will be-

- Proficient ESP32 setup for gas sensor monitoring.
- Secure IoT data transmission using XAMPP and Arduino Cloud.
- Get a basic understanding of how databases work using PHP.
- How to prepare a simple webpage to show sensor data.
- Effective use of virtual "Thing" for data exchange in Arduino Cloud.
- Competent programming for sensor data processing and LED control.

- User-friendly IoT dashboard design for real-time data visualization.

<u>Components required:</u>

Hardware:

1) ESP-32 Development Board
2) LED
3) DHT11
4) MQ-2 Gas Sensor
5) Breadboard
6) Jumper wires
7) 10k ohm Resistor

Software:

XAMPP and Arduino Cloud IoT

<u>Theory:</u>

Fig. 1: Diagram of ESP32

The ESP32 is a widely-used System on a Chip (SoC) that plays a key role in developing Internet of Things (IoT) devices. It boasts a range of features that make it a preferred choice for various IoT applications.

- **Built-in Connectivity:** The ESP32 comes equipped with both WiFi and Bluetooth capabilities, making it suitable for IoT devices that need to transmit data over the internet to operate

- **Security Requirements:** The ESP32 offers robust hardware encryption support, which is crucial for ensuring secure communication.

- **Dual Cores:** With two processing cores, the ESP32 enables multitasking and fast parallel processing, making it ideal for handling real-time applications in embedded systems and IoT devices.

- **Storage:** With 1.5 MB of SPIFFS storage, the ESP32 can efficiently store and manage files on external flash memory. This is particularly useful for IoT applications where data storage is a requirement.

- **RAM:** The ESP32 boasts 520 KB of RAM, which is essential for handling data and operations efficiently, especially in IoT devices.

- **Sensor Compatibility:** The ESP32 interfaces seamlessly with a wide range of IoT sensors, supporting UART, I2C, and SPI communication. It features ADC(Analog to digital conversion) and pulse counting for data acquisition and is compatible with capacitive touch sensors, ensuring adaptability to diverse sensor types.

For more information about the ESP32 click here.

Pin Diagram of ESP32:

Fig. 2: ESP32 Pin Diagram

The ESP32 peripherals include:

- 18 Analog-to-Digital Converter (ADC) channels
- 3 SPI interfaces
- 3 UART interfaces
- 2 I2C interfaces
- 16 PWM output channels
- 2 Digital-to-Analog Converters (DAC)
- 2 I2S interfaces
- 10 Capacitive sensing GPIO's

Here, the ADC and DAC features are associated with specific static pins. However, UART, I2C, SPI, PWM, etc are assignable independently by code. A more detailed description of each pin is found here.

In this experiment, we're going to explore how to create an IoT device using the versatile ESP32 microcontroller, a DHT11 sensor, an MQ2 gas sensor, and a simple LED. Our goal is to first read temperature and humidity data using the DHT11 sensor and then send the data to be stored on a database (localhost). After that, we also want to show the variable temperature and sensor data on a local webpage. Then, we shall detect smoke using the MQ2 sensor and seamlessly transmit

this data to the local database and Arduino IoT Cloud platform. Moreover, we'll implement a convenient remote control feature via the XAMPP and Arduino IoT cloud dashboard to toggle the LED on and off. This experiment will provide you with valuable insights into the fundamental principles of IoT technology and hands-on experience in building IoT applications.

Installing ESP32 in Arduino IDE:

You can follow these links for installing and troubleshooting in ESP32:
Installing ESP32 in Arduino IDE (Windows, Mac OS X, Linux) | Random Nerd Tutorials
ESP32 Troubleshooting Guide | Random Nerd Tutorials

Now, let's connect the DHT11 sensor to your ESP32 as shown in Fig. 3.

Fig. 3: Connection diagram of DHT11 sensor with ESP32.

Now, we need to install a library for DHT11 in the Arduino IDE to read the temperature and humidity data successfully. Go to Sketch→Include Library→Manage Libraries and then search with the keyword "DHT11" like shown below:

Many libraries will show in the search results. However, install the one given by "Adafruit". The exact library is shown in the snippet below:

Now, let's write a simple Arduino code to read the temperature and humidity data from DHT11 and show it to the serial monitor of the Arduino IDE.

Code:

```
#include <DHT.h>
#define DHTPIN 19 //D19
#define DHTTYPE DHT11
DHT dht11(DHTPIN, DHTTYPE); // declare an object "dht11"

int temperature = 0;
int humidity = 0;

void setup() {
  Serial.begin(115200);

  dht11.begin(); // Initialize the sensor
}

void loop() {
```

```
  Load_DHT11_Data();
  Serial.println("Temperature: %d °C\n", temperature);
  Serial.println("Humidity: %d %%\n", humidity);
}


void Load_DHT11_Data() {
  //————————————————————————————————————————————————————————
  temperature = dht11.readTemperature(); //Celsius
  humidity = dht11.readHumidity();
  //————————————————————————————————————————————————————————
  // Check if any reads failed.
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    temperature = 0;
    humidity = 0;
  }
}
```

Now, save this code, compile and then upload the code to ESP32. It's important to note that sometimes the code uploading to the ESP32 may fail. While this can be due to old USB ports or cable issues, one of the main issues can be due to not rebooting the ESP32. For this, if you see the code is not uploaded to the ESP32, then **you should press and hold the boot button and press the EN button too on the ESP32, located at the bottom right and left corner shown in Fig. 4**, then you click the code upload button on the screen. Wait for the code to upload while holding the boot button.

Fig. 4: Boot Button and EN button on ESP32

After uploading to the ESP32, you should be able to see the temperature and humidity data on the **Serial Monitor**.

If you see any kind of problem with uploading issues, first read the error message. You can search the internet with the error messages. Additionally, you can follow the link for troubleshooting ESP32:

[ESP32 Troubleshooting Guide | Random Nerd Tutorials]

For more information of DHT11 with Arduino/ESP32, please visit: [Arduino - DHT11 | Arduino Tutorial (arduinogetstarted.com)]

IoT Cloud Using XAMPP:

Now, we will send the temperature and humidity data from the ESP32 to a local server!

XAMPP is a free and open-source tool that helps developers create and test websites on their own computers before making them live on the internet. It's like a mini-internet server that runs on your computer. The basic components of XAMPP can be seen in Fig. 5. XAMPP includes:

- **Apache**: The main software that serves web pages.
- **MariaDB**: A database that stores website data.
- **PHP and Perl**: Programming languages used to build web applications.
- **XAMPP** is an abbreviation where **X** stands for Cross-Platform, **A** stands for **Apache**, **M**

stands for **MYSQL**, and the **Ps** stand for **PHP** and **Perl**, respectively.

XAMPP is easy to install and use, making it perfect for beginners. It allows you to test websites and make sure everything works properly before going live.

Fig. 5: Basic Components of XAMPP

Apache

What it does: Apache is a web server software. It takes requests from users' web browsers and sends back the requested web pages.

Example: When you type "www.example.com" in your browser, Apache is the software that delivers the website's content to your screen.

MySQL

What it does: MySQL is a database management system. It stores and manages data for websites.

Example: For an online store, MySQL keeps track of product details, customer information, and order history.

PHP

What it does: PHP is a scripting language used for creating dynamic web pages. It runs on the server and generates HTML to send to the user's browser.

Example: When you log into a website, PHP checks your username and password against the data stored in MySQL, and then directs you to your account page.

Interaction in XAMPP among Apace, MySQL & PHP

Example of interaction: Imagine you have a blog. When a visitor comes to your site:

1. **Apache** handles the visitor's request and delivers the web page.
2. The blog post content is stored in **MySQL**.
3. **PHP** retrieves the blog post from MySQL, formats it, and generates the HTML to be displayed.

So, in XAMPP:

- **Apache** serves the web page.
- **MySQL** stores the data.
- **PHP** processes the data and creates the web page content dynamically.

Together, they create a fully functional and interactive website.

<u>Installing XAMPP:</u>

Step-1:
Download the xampp.exe from the website: **https://www.apachefriends.org/→** XAMPP Installers and Downloads for Apache Friends and install the file. The installation is very straightforward.

Step-2:
To activate the server, we need to go to the installation directory of the XAMPP (for example: "C:\xampp"), then double click the "xampp-control.exe" file.

xampp_shell.bat	6/24/2024 10:45 PM	Windows Batch File	2 KB
xampp_start.exe	3/30/2013 6:29 PM	Application	116 KB
xampp_stop.exe	3/30/2013 6:29 PM	Application	116 KB
xampp-control.exe	4/6/2021 5:38 PM	Application	3,290 KB

A control panel like the below image will pop up:

Click on the 'start' button in the Actions option to start the Apache and MySQL module. **This is a must to run the XAMPP services while interfacing with ARDUINO or ESP32.**

Step-3:

Let's see if the XAMPP server is working properly! First, type "localhost" in any of your web browser and press enter like the following:

The web server of XAMPP should be displayed like the following:

If you see such a page on your browser that means the "XAMPP" is properly installed.

Step-4:

Click on the 'phpMyAdmin' to enter the database. A image like the following will appear:

You can change username and password from the "User accounts" option and a following login screen will appear like shown below **(By default there will be no login screen like below, you have to change in the settings)**:

Username: "root"

Password: there is no default password

You can change the Username and Password in your personal PC/laptop but do not change for the LAB PCs!

After changing the username and password in your own laptop/pc, you should do the following in the XAMPP installed in your PC. Otherwise, you will not see the login screen!

Open xampp/phpMyAdmin/config.inc and change the line

```
$cfg['Servers'][$i]['auth_type'] = 'config';
```

to

```
$cfg['Servers'][$i]['auth_type'] = 'cookie';
```

<u>Creating a Database in XAMPP using "phpMyAdmin":</u>

Step-1:
Click on the 'new' option to create a new database:

Step-2:
Give an appropriate database name in the "Database name" option. Then, click on the "Create" button.

Step-3:

Then give an appropriate database table name as shown in the below image. A single database can have multiple tables. You can set the column value according to your need.

Here, we will get 2 values from the DHT11 sensor: temperature and humidity. Additionally, we want to maintain a serial number, “ID” and current timestamp, “Datetime” to track the time when we get the data from the sensors. The table should have 4 columns then:
“ID”, “temperature”, “humidity” and “Datetime”. Now, click on “Create” to create the table with 4 columns. An example of the created table is shown below:

Step-4:

Now, we want to see if our database can be updated properly using SQL command! Go to the SQL option and write the following command:

```
INSERT INTO your_table_name(temperature, humidity) VALUES (10, 20)
```

Then, click on the ‘go’ command and check if the data table has a new entry with (10, 20).

If the SQL command executes successfully, a message will be displayed like this:

You can check the inserted data by clicking the Browse button.

Step-5:

Next step is to write a php script to check whether we can insert fake data into our database through php script to make sure that everything is working correctly before we start sending the real data from the esp32.

To write a PHP script first you need to create a project folder where you will store all the project files. So navigate to the xampp installation directory from C drive here open the “htdocs” folder.

The path: **C:\xampp\htdocs**

This is where you will create your project folder. So create a new folder and give it a name such as dht11 underscore project (“dht11_project”). Next create a new PHP file in this folder and give your file a name for this project. Let's call it “testdata.php”.

Now open this file with any text editor(vscode/notepad/notepad++ etc.) Then write the following code:

```
<?php
echo "Hello World<br>";
?>
```

Here,"<?php ?>" is the opening and closing tags of PHP. And between the tags a message is printed("echo" is used to print any message in the webpage just like the printf function of C programming language).

Now, save the file by pressing **Ctrl S** and make sure you must save this file in the "dht11_project" folder with PHP extension.

To check whether it is working or not, open your web browser and enter the URL **"localhost\your_project_folder_name\file_name_with_extension".** For this case it is **"localhost/dht11_project/testdata.php"**. Now press enter to load the page. You'll be able to see the text displayed on the screen.

Now let's insert fake data in the database through the PHP script to see if the database is working properly. Update your php file by writing down the following code:

```
<?php

$hostname = "localhost";
$username = "root";
$password = "";
$database = "your database name"; //For this example, it is sensor_db

$conn = mysqli_connect($hostname, $username, $password, $database);

if (!$conn) {
        die("Connection failed: " . mysqli_connect_error());
```

```
}

echo "Database connection is OK<br>";
?>
```

Then save your code and refresh your browser. If everything goes well, the following page will be displayed.

Here, $hostname, $username, $password, and $database are some variables which are assigned appropriate values to establish the connection with the database. Then, hostname is assigned to localhost, username is root and no password (by default) and database is assigned to the name you have given for your database. For this example, it will be “sensor_db”.

Next to establish a connection to the database, a function is used. The function is **“mysqli_connect”.** This function requires four parameters which are already stored in the variables : hostname, username, password, and database name.
Next an if condition is used to handle the situation if the connection to the database fails. So if the connection fails then the program will stop running and an error message will be displayed to explain the reason for the error. Next if the program successfully connects to the database, then a message will be displayed “database connection is okay”.

Next write the following SQL query to save the data into the table.

```
$sql = "INSERT INTO dht11 (Temperature, Humidity) VALUES (20, 30)";
if (mysqli_query($conn, $sql)) {
        echo "\nNew record created successfully";
} else {
        echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
```

Again save your code and refresh your browser. If everything goes well, the following page will be displayed.

Here, the insertion query which was used before is being stored in the query variable and then executes the SQL query using the **"mysqli_ query"** function. Here, a if condition is added which will print out a message to indicate whether the query was successful or not.
To make sure that the data has been inserted into the database go back to phpmyadmin. Then click on sensor_db database. Then click on the dht11 table to select it. You will be able to see the temperature and humidity data that you have just inserted in the table.

←T→			ID	Temperature	Humidity	Date and Time
Edit	Copy	Delete	1	10	20	2024-07-01 09:48:16
Edit	Copy	Delete	2	20	30	2024-07-01 11:54:42

Next one more change in the code will be introduced. Currently the temperature and humidity data is hard-coded in the SQL query which is not a good practice. So instead of hard-coded data, we will use variables. So modify the previous code like this:

```
$t = 1;
$h = 2;

$sql = "INSERT INTO dht11 (Temperature, Humidity) VALUES (".$t.", ".$h.")";
if (mysqli_query($conn, $sql)) {
        echo "\nNew record created successfully";
} else {
        echo "Error: " . $sql . "<br>" . mysqli_error($conn);
}
```

Again save your code and refresh your browser. Then go back to the table to check. You will be able to see the temperature and humidity data that you have just inserted in the table through variables.

←T→			ID	Temperature	Humidity	Date and Time
Edit	Copy	Delete	1	10	20	2024-07-01 09:48:16
Edit	Copy	Delete	2	20	30	2024-07-01 11:54:42
Edit	Copy	Delete	3	1	2	2024-07-01 10:33:14

Here, two variables have been declared. The first one is $t it will be used to store the temperature and the second variable is $h and it will be used to store the humidity. Then proper changes are made in the sql query to insert the variables instead of hard coded data. This way if you want to change the temperature or humidity values, you can simply modify the variables and the query will automatically use the new values. Using variables is a good programming practice because it makes the code more flexible and easier to modify in the future.

Step-6:

Now, we will write our esp32 code to send data to the XAMPP server. Here first, you need to include libraries that are required for the project to function properly. The first one is the **Wi-Fi library** and this Library will be used for connecting to the Wi-Fi network. The next library is the **HTTP client library** and it will be used to send HTTP requests from ESP32 to our localhost server.

Code:

```
#include <WiFi.h>
#include <HTTPClient.h>

#include <DHT.h>
#define DHTPIN 19 //D19
#define DHTTYPE DHT11
DHT dht11(DHTPIN, DHTTYPE);

String URL = "http://computer_ip/project_folder/project_file.php";

const char* ssid = "ENTER_WIFI_SSID";
const char* password = "ENTER_WIFI_PASSWORD";

int temperature = 0;
int humidity = 0;

void setup() {
  Serial.begin(115200);
```

```
  dht11.begin();

  connectWiFi();
}

void loop() {
  if(WiFi.status() ≠ WL_CONNECTED) {
    connectWiFi();
  }

  Load_DHT11_Data();
   String postData = "temperature=" + String(temperature) + "&humidity=" +
String(humidity);

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

  int httpCode = http.POST(postData);
  String payload = http.getString();

  Serial.print("URL : "); Serial.println(URL);
  Serial.print("Data: "); Serial.println(postData);
  Serial.print("httpCode: "); Serial.println(httpCode);
  Serial.print("payload : "); Serial.println(payload);
  Serial.println("————————————————————————————————————————————");
  delay(5000);
}


void Load_DHT11_Data() {
  //————————————————————————————————————————————————————————
  temperature = dht11.readTemperature(); //Celsius
  humidity = dht11.readHumidity();
  //————————————————————————————————————————————————————————
  // Check if any reads failed.
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    temperature = 0;
    humidity = 0;
  }
```

```
  //————————————————————————————————————————————————————————————
  Serial.printf("Temperature: %d °C\n", temperature);
  Serial.printf("Humidity: %d %%\n", humidity);
}

void connectWiFi() {
  WiFi.mode(WIFI_OFF);
  delay(1000);
  //This line hides the viewing of ESP as wifi hotspot
  WiFi.mode(WIFI_STA);

  WiFi.begin(ssid, password);
  Serial.println("Connecting to WiFi");

  while (WiFi.status() ≠ WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }

  Serial.print("connected to : "); Serial.println(ssid);
  Serial.print("IP address: "); Serial.println(WiFi.localIP());
}
```

Next you need to define the address of the web server where you want to send the temperature and humidity data. Here first you need to enter the IP address of your computer which you can easily find by opening the command prompt. So open the **command prompt** (by writing cmd in the search option in windows) on your computer.

Here you can use the **"ipconfig"** command to display the IP address of your computer.

Now here you can see the IP address of the computer in the IPv4 Address part.
Next you just need to enter this IP address in the esp32 code. After that you need to enter the name of the folder where you have saved your project file and then the filename of the php script. For this example, the folder name is **dht11_project** and the filename is **testdata.php**.

Now the URL is ready. **If you are not sure about the URL, you can copy it and then paste it into your web browser. If you see a success message in your browser it means the URL is properly configured.** Just remember you must start the Apache and MySQL services before testing the URL. Next two variables are defined. One is ssid and another one is the password that you will use to connect to the Wi-Fi router or mobile hotspot. You should replace the values inside the quotes with the SSID and Password of your own network.

Next two more variables are defined. One for temperature and another one for humidity which will store the sensor data that will be sent to the Local Host. In this example both variables are initialized to 0.
In the setup function, **Serial.begin()** function is used to set up serial communication between the esp32 and your computer. Next you need to establish a connection to the Wi-Fi router by calling **connectWiFi()** function. This function will attempt to connect to the Wi-Fi network using the SSID and Password that you have defined earlier. This function is added below the loop function. This function contains a sample code that is used in many esp32 projects that require Wi-Fi connectivity. It is a standard code for connecting to any Wi-Fi network. You can also use this function in other projects you make with the esp32 board so you don't have to create a new function every time you want to connect to a Wi-Fi network.
Next let's move in the **loop function**. Here first a checking is done whether the esp32 is connected to the Wi-Fi network. If it is not then it calls the connect Wi-Fi function to attempt to reconnect to the Wi-Fi.
Next a string variable called **postData** is created which holds the temperature and the humidity data that you want to send to the local server. Here first the temperature data is added ("temperature=" + String(temperature)) which is 0. It is the same temperature value that you have defined above. Next the same way the humidity value which is 0 is added ("&humidity=" + String(humidity)). The complete string is look something like this:

temperature=0&humidity=0

Now you have defined the post data string with temperature and humidity data stored in it. Now you will send this data to the server using the HTTP post request. So you will create a new HTTP client object called **http**. It will be used to send an HTTP post request to the web server.
Next let's establish a connection to the xampp server. The **begin function (http.begin(URL))** of the HTTP client object is used to connect the esp32 to the web page which is defined in the URL variable. In this example it is **testdata.php** which you have created earlier. So the esp32 will connect to the testdata.php file and you will send temperature and humidity data to this file. To

send the temperature and humidity data to the server you will use the post method (**http.POST(postData)**). This method will send an HTTP post request to the server with the temperature and humidity data you have stored in the postData variable. An HTTP post method will send it to the testdata.php file. Now the temperature and humidity data is received at the server, so you need to store it to the database table.

Now, you need to modify your code of the testdata.php file like this:

```
if(isset($_POST["temperature"]) && isset($_POST["humidity"])) {

      $t = $_POST["temperature"];
      $h = $_POST["humidity"];

      $sql  =  "INSERT  INTO  dht11  (temperature,  humidity)  VALUES  (".$t.",
".$h.")";

      if (mysqli_query($conn, $sql)) {
            echo "\nNew record created successfully";
      } else {
            echo "Error: " . $sql . "<br>" . mysqli_error($conn);
      }
}
```

First a check is done, if there is any valid data received or not using the if condition and if you have some valid temperature and humidity data then you will store that data into these variables **($t, $h)**. So first you will store the temperature data into the $t variable that you have defined for temperature. The same way you will store the humidity data to $h variable. These changes will make the testdata.php file ready to store the temperature and humidity data into the database coming from the esp32.
Next, in the loop function of the arduino ide, a httpcode variable is used to store the response code from the server. If the data is successfully posted to the server then you will receive the response 200. Next another variable called payload is defined. It will store the response message from the server. The message looks something like this:

**Hello World
Database connection is OK
 New record created successfully**

These are the same messages that you have seen earlier while testing the PHP file. In case of error, the payload string contains the error message with the reason for the error. Finally for testing purposes the URL, postData, http code and payload variables are printed to the serial monitor. Here a special message is included called header. Header helps the server to understand the type of information sent to it. Now the code is ready to upload.

Let’s see an example of interfacing ESP32 with Arduino Cloud IoT to read Gas sensor data. Here is the circuit diagram below:

Circuit:

In this setup, you'll connect the gas sensor to the ESP32 as shown in Fig 4.

First, the VCC pin of the MQ2 sensor should be connected to the VIN pin of the ESP32. The A0 pin of the MQ2 will be linked to D4, and you'll connect the positive pin of the LED to D2 on the ESP. Finally, the negative pin of the LED and the GND pin of the MQ2 will be connected to the GND of the ESP32.

Fig. 4: Circuit Connection of ESP32 with Gas Sensor.

<u>Arduino Cloud IoT:</u>

The Arduino IoT Cloud is an online platform that makes creating, deploying, and monitoring IoT projects easy.

To start, we will need to head over to the Arduino Cloud IoT and then Sign in to the Arduino account.
To sign up, follow the steps below:

Step 01: Sign In or Create Account

Fig. 4: Arduino IoT Cloud Sign In Page

To create an account, click on Create one. Or if you already have an account, you can just simply click on the Sign In button as shown in *Fig. 4*. After Signing in, you will land to the page below shown in *Fig. 5*.

Fig. 5: Arduino IoT Cloud Home Page

Step 02: Setting up the device

1. In the Arduino Cloud IoT interface, click on the "Devices" tab as shown in Fig. 6.

Fig. 6: Device Tab in the arduino cloud website

2. Click the "Add device" button in this “device tab” as shown in Fig. 6.

Fig. 7: Device Setup Page

3. Select "Set up a 3rd party device" as shown in Fig. 7.

Fig. 8: Selecting Board

4. Then, Choose "ESP32" from the drop-down menu for the board selection. Then, Select the model “ DOIT ESP32 DEVKIT V1” as shown in Fig. 8.

Fig. 9: Setting Up Device Name

5. Now the board needs a name (Fig. 9). We chose *esp32*, but you may use any other creative name!

Fig. 10: Secret Key for the esp32 device.

6. You will now receive your Device ID and Secret key as shown in Fig. 10. Please note that the ***secret key cannot be recovered***, so make sure you note it down. You can also download a PDF with the information.

7. After saving it, tick the box at the bottom, and click on "Continue".

Fig. 11: Device Setup

By doing that, You have now configured your ESP32 board with the Arduino Cloud IoT. Your device will now appear in the list of devices.

Step 03: Creating a Thing

1. To create a Thing, you need to click on the "Things" to navigate to the “Things” tab as shown in Fig. 12

Fig. 12: Thing Tab

2. Then, click on the "Create Thing" button. We can then rename our Thing to something appropriate to what we are doing, in this case, we simply chose ***Gas Data***. In this ***Gas Data*** things (Fig. 13), you will find **setup** and **sketch** options. You can think of ***Gas Data*** as a project where you can declare variables, add codes and upload it just like in regular arduino!

Fig. 13: Giving your project a name in the Things tab

3. Now, we need to link our device with our Thing. This is done by navigating to the "Device" section. This will open a window, where your ESP32 should be available to select as shown in Fig. 14.

Fig. 14: Associate Device

4. Once the device is linked, we need to create two variables: ***Gas_ppm*** and ***LED***.
5. Click on the "Add variable" button. This will open a window where you need to fill in variable information.

   I. Let's create the ***Gas_ppm*** first (Fig. 15). The data type is *int*, permission is *read-only* and the updated policy is *Periodically every 3s*. (Here ESP32 will send gas sensor data

every 3s) Once done, click on the "Add variable" button.

Fig. 15: Setting Up Gas_ppm variable in the things.

II. Now, let's also add the ***LED*** variable. The data type for this variable is *boolean*, the permission is *read & write*, and the update policy is *on change*. Once done, click on the "Add variable" button.

Fig. 16: Setting Up “LED” Variable in the ***Gas Data*** Project.

Step 04: Adding credentials

Now, we need to enter the credentials for our network and the secret key generated during the device configuration.

1. Click on the button in the "Network Section" at right bottom as shown in Fig. 17 (marked by red box)

Fig. 17: Set Up Network

2. Then, enter the credentials (network name[Your WiFi you are connected to], network password[Your WiFi password], and secret key[Secret key you got while setting up things]) as shown in Fig. 18.. Click "Save" when finished.

Fig. 18: Entering Network Credentials

3. The next step is to program the board. To do so, we need to go to the "*Sketch*" tab (Fig. 19).

Fig. 19: Write in the Sketch (Code)

[Here all of the functions for the project will be automatically generated for the predefined variables.]

4. We have to add our code here for getting the gas sensor data and controlling the LED. Write the following code, then we will verify, then upload the sketch to your connected ESP32! We will see in the last section on how to connect your ESP32…

   Code:

```
#include "thingProperties.h"

// Pin connected to the MQ-2 sensor

const int sensorPin = 4;

int led = 2;

// Define the load resistance value (in ohms) used in the circuit

#define RL 10  //Load resistance

#define m -0.263 //Calculated Slope

#define cb 0.42 //Calculated intercept

#define Ro 20 // Resistance on fresh air

void setup() {

      // Initialize serial and wait for port to open:

      Serial.begin(9600);

      // This delay gives the chance to wait for a Serial Monitor without
blocking if none is found

      delay(1500);



       // Defined in thingProperties.h

      initProperties();
```

```
      // Connect to Arduino IoT Cloud
      ArduinoCloud.begin(ArduinoIoTPreferredConnection);


      /*
      The following function allows you to obtain more information
      related to the state of network and IoT Cloud connection and errors
      the higher number the more granular information you’ll get.
      The default is 0 (only errors).
      Maximum is 4
       */
      setDebugMessageLevel(4);
      ArduinoCloud.printDebugInfo();
      pinMode(sensorPin, INPUT);
      pinMode(led, OUTPUT);
}


void loop() {
      ArduinoCloud.update();
      float VRL; //Voltage drop across the MQ sensor
      float Rs; //Sensor resistance at gas concentration
      float ratio; //Define variable for ratio
      float sensorValue = analogRead(sensorPin);
```

```
      Serial.println(sensorValue);



      VRL = sensorValue * (5.0/1023.0); //Measure the voltage drop and convert
to 0-5V

      Rs = ((5.0*RL)/VRL)-RL; //Use formula to get Rs value

      ratio = Rs/Ro;  // find ratio Rs/Ro

      float ppm = pow(10, ((log10(ratio)-cb)/m)); //use formula to calculate
ppm



  Serial.print("PPM: ");

  Serial.println(ppm);

  gas_ppm = ppm;

}

/*

      Since LED is READ_WRITE variable, onLEDChange() is

      executed every time a new value is received from IoT Cloud.

*/

void onLEDChange()  {

      if (lED == 0) {

      digitalWrite(led, HIGH);

      }

  else

  {

    digitalWrite(led, LOW);
```

```
  }
}
```

Step 05: Creating a dashboard

Now that the sketch is written on the ESP32 board, we can move on to the final step, which is creating a dashboard.

1. Navigate to the "Dashboards" tab as shown below in Fig. 20

Fig. 20: Build Dashboard

2. Then, click on the "*Build dashboard*" button. A dashboard will appear as shown in Fig. 21. Here, you can observe your variables value, can control various parameters and can observe real time data!

Fig. 21: Dashboard

3. To edit the dashboard, click on the pencil icon at the top left icon, select "*widgets*" and search for your Thing.

Fig. 22 : Add Widgets

I. First of all, add a *switch.*

Fig. 23: Widget-Switch

a) Click the link variable and link the LED variable

Fig. 24: Link Led

II. Then add a "*Chart*" widget and link variable to link the Gas_ppm

Fig. 25: Chart-Widget

III. Lastly, add "*Value*" Widget and link it to Gas_ppm

Fig. 26: Value-Widget

[You can arrange the widgets as desired on the dashboard interface.]

4. Customize the appearance and settings of the widgets as needed. Finally, the dashboard should look like this:

Fig. 27: Widget Added Dashboard

Upload code to the Board:

Now, finally it's time to upload our code to the ESP32 board and make the IoT device operational, follow these steps:

1. Begin by installing the Arduino agent.

2. Next, connect the ESP32 to your computer. If you encounter an issue that says '*No associated device found*,' shown in Fig. 28 please click here to download and install the necessary driver. Once installed, you should be ready to proceed.

Fig. 28: No Associated Device Found

3. To upload the code to the board, click the right arrow button as shown in Fig. 29. It's important to note that you should ***press and hold the boot button and press the EN button too*** on the ESP32, located at the bottom right and left corner shown in *Fig. 30*, ***then*** you click the code upload button on the screen. Wait for the code to upload while holding the boot button. The ESP32 may ask to install 3 additional drivers this time, notice that and click yes if asked.

Fig. 29: Upload Button

Fig. 30: Boot Button and EN button on ESP32

Local Server:

1) ESP32 itself working as a WiFi
2) Your sensor connected to ESP32
3) Now, you can connect your other devices to your ESP32-WiFi
4) Then, login to your devices using the IP address provided by your ESP32 WiFi to see the data from the sensor.

Code:

```
#include "WiFi.h"
#include "ESPAsyncWebServer.h"

#define AO_PIN 35

const char* ssid = "ESP32-Access-Point";
const char* password = "123456789";

AsyncWebServer server(80);
```

```
String readGas() {
  int Gas = analogRead(AO_PIN);
  return String(Gas);
}

void setup() {
  Serial.begin(115200);

  WiFi.softAP(ssid, password);

  IPAddress IP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(IP);

  server.on("/events", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(200, "text/event-stream", "data: " + readGas() +
"\n\n");
  });

  server.on("/readGas", HTTP_GET, [](AsyncWebServerRequest *request) {
    request->send(200, "text/plain", readGas().c_str());
  });

  server.on("/", HTTP_GET, [](AsyncWebServerRequest *request) {
    String page = "<html><head>";
    page += "<style>body { font-family: Arial, sans-serif;
background-color: #f4f4f4; display: flex; align-items: center;
justify-content: center; height: 100vh; margin: 0; }</style>";
    page += "<link rel='stylesheet'
href='https://cdnjs.cloudflare.com/ajax/libs/font-awesome/5.15.4/css/a
ll.min.css'
integrity='sha512-Zh2e5lR6b1E7G5kmC5q2/Jz2QHiyyZB49UoMKApslEKnEz1o6eHA
h0FvsNpMOG/z6b1d0OE7QxTUQA9QHJ5oQ==' crossorigin='anonymous' />";
    page += "<script>var eventSource = new EventSource('/events'); "
            "eventSource.onmessage = function(event) {
document.getElementById('temp').innerHTML = '<i class=\"fas
fa-thermometer-half\"></i> Gas: ' + event.data; };</script>";
    page += "</head><body>";
```

```
    page += "<div id='temp' style='font-size: 100px; margin: 20px;
text-align: center;'></div>";
    page += "</body></html>";
    request→send(200, "text/html", page);
  });

  server.begin();
}

void loop() {
  // Your loop code goes here
}
```

Data Observation:

1. Access the dashboard from your Arduino Cloud IoT account.
2. Use the switch widget to control the LED on your ESP32 board.
3. Observe real-time gas sensor data displayed on the chart and value widgets.

Conclusion:

This IoT experiment demonstrated the entire lifecycle of an IoT project, from device configuration and variable setup to code implementation and user interface creation. The Arduino Cloud IoT platform served as a powerful tool for managing and controlling our ESP32-based project, highlighting the ease and effectiveness of using cloud-based services for IoT development. This experiment provided valuable hands-on experience in building and managing IoT solutions and can serve as a foundation for more advanced IoT projects and applications.

Lab Reports:

Normal Structure of the lab report (same for all the lab reports):

SL No.	Sections	What to Write
1.	Cover Page	Add a cover page and use the format uploaded in the drive (Lab Report Format (with cover page).docx)
2.	Objective	Write 2-3 lines using points to mention what was the main learning outcome of the report
3.	Components	Write what types of components you used.

4.	Theory	Write some theoretical background if needed for the problems you are going to solve in this report
5.	Problems mention in the tasks	Solve all the problems, attach circuit diagrams (take screenshots or using camera phone), attach codes in detail. Use comments in the code so that we can understand what did you write
6.	Discussion	Write 5-7 lines on what you learn and what problems you faced when solving these problems.

Provide the relevant circuit connections and their codes for all the problems mentioned below:

1) Explain briefly how ESP32 is communicating via Arduino IoT Cloud. Suppose, you are observing the moisture in an agricultural farm using a sensor. You have to send the sensor data to a cloud. Based on the sensor data, the cloud would send a signal to the microcontroller used in the farm to turn on the motor pump. Explain how you can build such a system using the concept we have seen in this experiment. Use a proper block diagram to draw your system.
2) Write a program to send “Hello!” and “CSE 4326!” using WiFi to the cloud and then read the data from the cloud. A green and red led light is connected to another ESP32. If “Hello!” is sent then a green led would turn on and if “CSE 4326!” is sent then the red led would turn on.
3) Create a local ESP32 server on which two Widgets are shown, one is a button switch which can be pressed to turn ON or OFF. The other widget is a LED image/symbol which shows the status of the LED i.e., when the light in a room is ON, the LED button turns ON else OFF. The web server should look like this:

Now, the LED is connected to the GPIO 2 Pin of the ESP32. When the button on the server is pressed, the LED should turn ON else OFF. The interface of the web server should look like the following:

Provide detailed codes and explanations!

4) A gas sensor is connected in your kitchen to measure the gas in ppm. The sensor is connected to ESP32 to send the data in the cloud. Now, write the necessary programs to save the gas data from the cloud in a google sheet or excel file. Then, read from the Excel file and turn on a buzzer if the gas value crosses 50 ppm. You can follow these links:
https://www.survivingwithandroid.com/integrate-arduino-google-sheets-iot-project/amp/
https://projecthub.arduino.cc/Arduino_Genuino/arduino-iot-cloud-google-sheets-integration-3c0df1