هوشمند سازی آزمایشگاه IoTCPP

اعضای گروه:

عسل اطاعتی -مبینا فخیمی- آیدا علیمحمدی- علیرضا نادری- مهدی قشمی- پویا سلیمانی فر

تقسیم وظایف:

عسل اطاعتی: داکیومنتیشن- پین اوت LCD - کمک در مگنتیک به علیرضا

مبینا فخیمی: ثینگزبورد

علیرضا نادری: مگنتیک و LDR و LCD- سنسور گاز

امیرحسین محمدی: RFID و LCD- تهیه عکسها

مهدی قشمی: دما و رطوبت و داکیومنتیشن بخش دما و رطوبت و همکاری در داکیومنتیشن ثینگزبورد

آیدا علیمحمدی: رله-کمک به علیرضا در LDR- داکیومنتیشن بخش رله

پویا سلیمانی فر :RCP - رله- پین اوت-LED

دروس: سیستم ها و کاربرد های هوشمند- پروتکل ها در اینترنت اشیا

اساتید: دکتر تقی خاکی و دکتر معصوم

تی ای ها: ابوالفضل مرتضی پور- روح الله محمدی

زمستان 1403

Academic honesty: ChatGPT 4 is used in various stages by students.

سرفصل ها:

مقدمه:

این پروژه برای هوشمند سازی آزمایشگاه IoT CPP است. این پروژه فازهای مختلفی داشته است. اندازه گیری دما و رطوبت آزمایشگاه، ثبت ورود و خروج اعضا، گزارش باز یا بسته بودن پنجره و گزارش تمام این موارد در داشبورد ثینگزبورد از جمله این موارد است. از طریق داشبورد ما میتوانیم دستگاه ها را خاموش یا روشن کنیم.

هدف از هر بخش:

ثبت ورود و خروج اعضا: نظم دهی بیشتر به آزمایشگاه و جلوگیری از حضور افراد غیر عضو

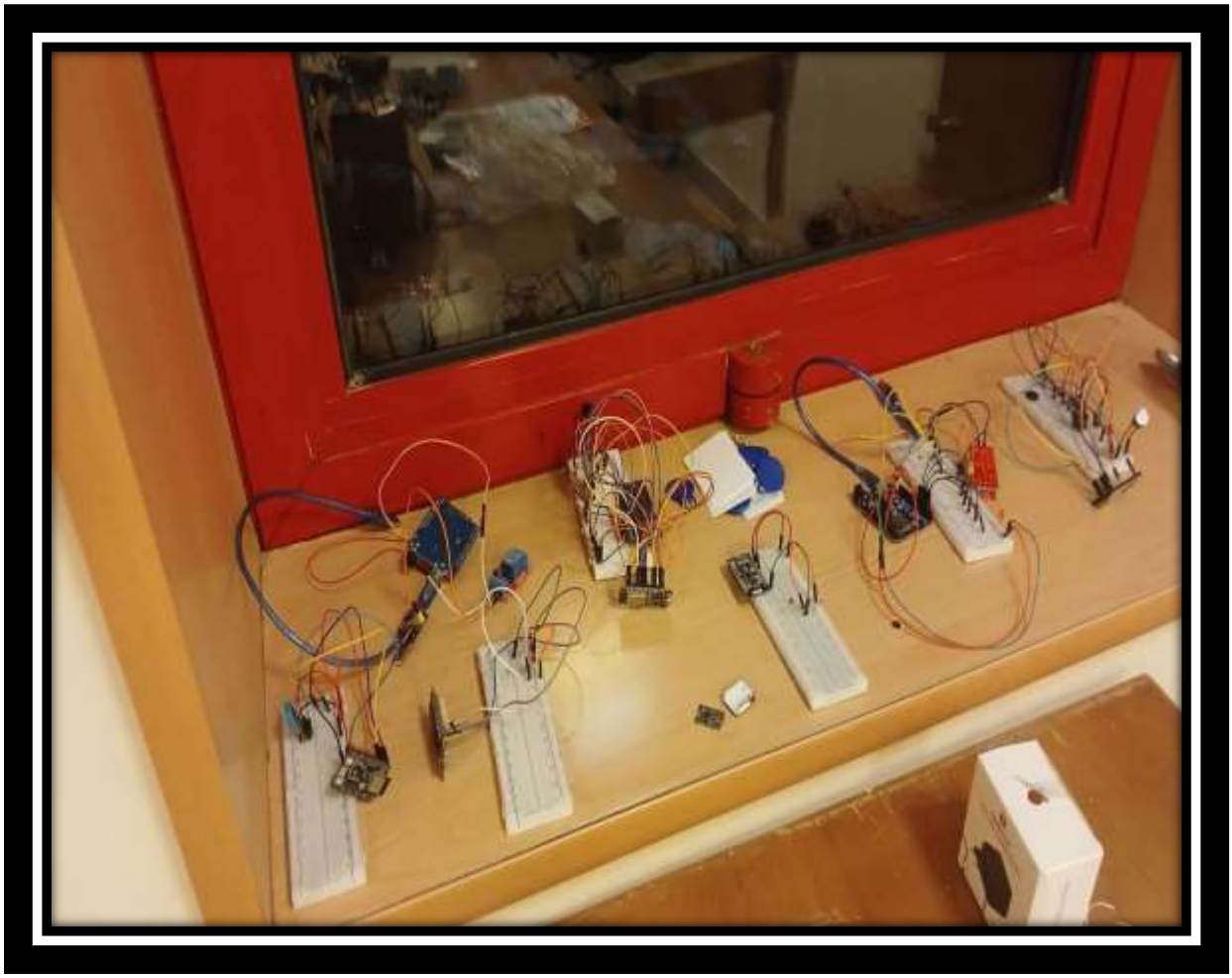
اندازه گیری دما و رطوبت: استفاده از سیستم های گرمایشی و سرمایشی در مواقع لزوم و جلوگیری از افزایش رطوبت ازمایشگاه

گزارش باز و بسته بودن در: بستن در توسط افراد جایگزین صورتی که اعضا فراموش کرده باشند آن را ببندند. که این کار باعث افزایش امنیت و عدم ورود گرد و غبار به داخل ازمایشگاه می شود.

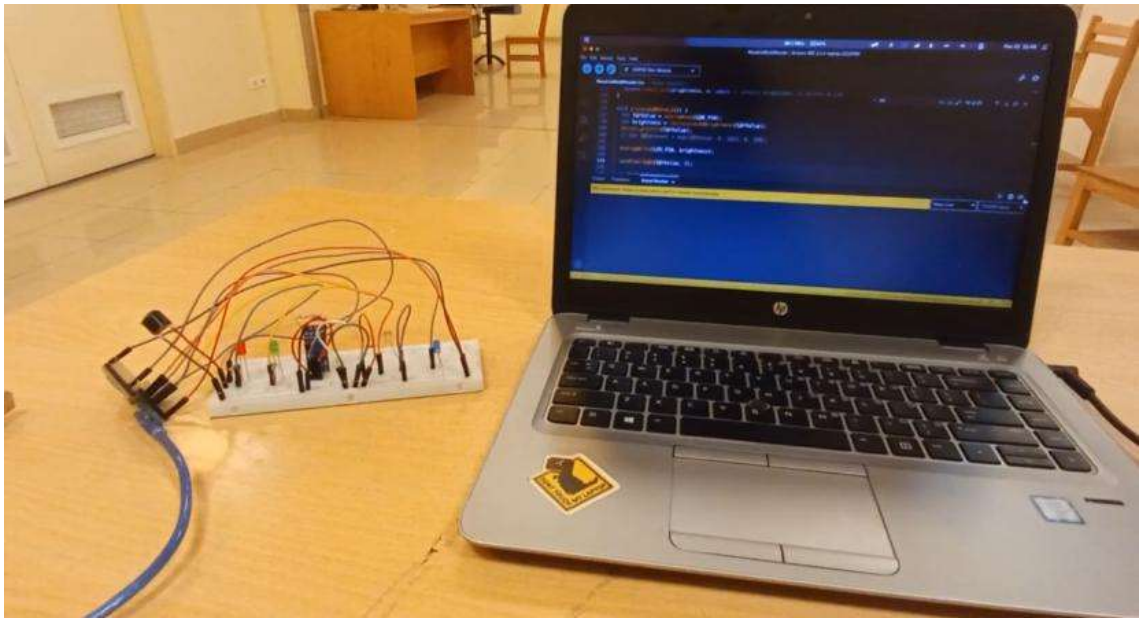
اندازه گیری گاز: جلوگیری از وقوع حوادث گازگرفتگی برای اعضای آزمایشگاه و دانشجوها.

RCP: کنترل از راه دور خاموش یا روشن بودن دیوایس ها

رله و برد اصلی: کنترل LED ها



فصل 1) پروژه مگنیتک



مقدمه: این برنامه یک سیستم نظارتی برای باز یا بسته بودن پنجره و روشنایی محیط با استفاده از میکروپروسسر ESP32، سنسورهای LDR و مغناطیسی، پروتکل‌های MQTT و HTTP است. همچنین داده‌های سنسورها به سرور ارسال شده و آلارم هایی با استفاده ازLED ها و Buzzer نمایش داده می شود.

اطلاعات شبکه مانند نام وای فای و رمز عبور وای فای مورد استفاده در متغیرهای ssid و password ذخیره شده اند. (در این کد از وای فای علیرضا استفاده شده است. اگر هنگام اجرا وای فای علیرضا مشکل پیدا کند کد تغییر می کند)

در صورتی که با موفقیت به وای فای متصل شویم، یک پیام در Serial Monitorنمایش داده می‌شود. که WiFi connected . از این طریق مطمئن می شویم اتصال ما به شبکه بدون مشکل بوده است.

از پروتکل MQTT برای ارسال داده ها به سرور استفاده شده است و اطلاعات مربوط به سرور شامل آدرس سرور، پورت، توکن ثینگزبورد در کد مشخص شده اند. از پورت 1883 استفاده شده است چون به صورت استاندارد برای پروتکل MQTT به کار می رود. همچنین از سرویس کلود ثینگز بورد اروپا استفاده شده است چون بهتر عمل می کند.

داده های وضعیت پنجره و سنسور از طریق MQTT به سرور Thingsboard ارسال می شوند.

از سنسور LDR برای اندازه گیری شدت نور محیط استفاده کردیم. مقدار نور خوانده شده و به درصد تبدیل شده و اگر نور کم باشد LED آبی روشن می شود. از ماژول مغناطیسی HW-017 برای تشخیص

باز یا بسته بودن پنجره استفاده شده است. اگر پنجره باز باشد LED سبز روشن می شود و Buzzer بوق می زند. اگر پنجره بسته باشد LED قرمز فعال می شود. صدای Buzzer قطع می شود.

مقادیر LDR و وضعیت پنجره به دو روش ارسال می شود:

1- از طریق پروتکل MQTT به سرور
2- از طریق درخواست HTTP به سرور ثینگزبورد

همچنین برنامه میتواند پیام هایی را از سرور دریافت کند مثلا اگر پنجره باز باشد پیام Open و اگر پنجره بسته باشد پیام closed را دریافت می کند.

برای جلوگیری از ارسال مکرر داده ها یک delay دو ثانیه تعریف می کنیم.

توضیح خط به خط کد:

کتابخانه ها

```
#include <WiFi.h>              // For connecting to Wi-Fi

#include <PubSubClient.h>      // For MQTT communication

#include <HTTPClient.h>        // For sending HTTP requests

#include <string>              // For using strings
```

اتصال به وای فای علیرضا و نعیین پسوورد و اسم

```
// Wi-Fi credentials

const char* ssid = "Alireza";  // Wi-Fi network name

const char* password = "123456789";  // Wi-Fi password
```

تنظیمات MQTT. انتخاب پورت 1883 به دلیل استاندارد بودن این پورت برای این نوع پروتکل است. توکن ثینگزبورد را جایگذاری می کنیم. و موضوع ارسال داده را مشخص می کنیم.

```
// MQTT server information

const char* mqttServer = "mqtt.eu.thingsboard.cloud";  // Server address

const int mqttPort = 1883;                              // MQTT server port

const char* mqttUser = "VJc3BJnrAzKLwKn7KTtR";         // Device token in ThingsBoard

const char* mqttTopic = "v1/devices/me/telemetry";     // Topic for sending data
```

پین ها را مشخص می کنیم

```
// Hardware pins

const int ldrPin = 34;       // LDR sensor pin (Light Dependent Resistor)

const int hw017Pin = 15;     // Magnetic module (HW-017) pin

const int greenLedPin = 26;  // Green LED pin

const int redLedPin = 27;    // Red LED pin

const int buzzerPin = 14;    // Buzzer pin

const int blueLedPin = 33;   // Blue LED pin
```

وای فای و کلاینت را تعریف می کنیم.

```
// Define Wi-Fi and MQTT clients

WiFiClient espClient;

PubSubClient mqttClient(espClient);
```

ولیو LDR را به ثینگربورد میفرستیم. از طریق توکن ثینگزبورد این ارسال انجام می شود.

```
// Function to send LDR sensor value to ThingsBoard

int sendToAllLdr(int value, int zoneId) {

  HTTPClient http;

  const String token = "IGa09xW2NYVHxcsaRvo8"; // API token

  const String tbServer = "http://eu.thingsboard.cloud/api/v1/" + token + "/attributes"; //
ThingsBoard server URL
```

پلی لود درواقع داده را در فرمت جیسون ارسال می کند . زون آیدی درواقع محل قرار گیری هر دستگاه در آزمایشگاه است.

```
  // Create payload in JSON format

  String payload = String("{\"Zone") + zoneId + "\" : " + value + "}";

  Serial.println(payload);  // Print payload for debugging
```

حالا از طریق جیسون درخواست HTTP میدهیم و داده ها در فرمت استرینگ رد و بدل می شوند.

```
  // Configure HTTP request

  http.begin(tbServer.c_str());

  http.addHeader("Content-Type", "application/json");  // Set content type as JSON
```

در HTTP ما انواع متود ها مثل post, getو ... را داریم درواقع متود پست برای ارسال داده ها به سرور است.

```
  // Send POST request

  int httpResponseCode = http.POST(payload);  // Send data to server
```

برررسی میکنیم که آیا ریسپانس کد ارسال شده است اگر نه ارور میفرستند. توجه: ما انواع ارور را مشخص نکردیم فقط یک نوع ارور مشخص کردیم. برای بهتر متوجه شدن مشکل میشود انواع ارور را مشخص کرد.

```
  if (httpResponseCode > 0) {

    String response = http.getString();  // Get response from server

    Serial.printf("HTTP Response code: %d\n", httpResponseCode);

    Serial.println(response);  // Print server response

  } else {

    Serial.printf("Error on sending POST: %s\n", http.errorToString(httpResponseCode).c_str());
// Print error if request fails

  }
```

ریکوئست را در آخر میبندیم.

```
  http.end();  // End HTTP request

  return httpResponseCode;  // Return HTTP response code

}
```

```
// Setup function for initial configuration

void setup() {
```

سرعت بالاتری را برای دیپاگ انتخاب کردیم به جای 9000

```
  Serial.begin(115200);  // Begin serial communication for debugging
```

پین مود ها (اوت پوت- این پوت را مشخص می کنیم)

```
  // Configure pin modes
  pinMode(ldrPin, INPUT);  // LDR sensor pin as input
  pinMode(hw017Pin, INPUT); // Magnetic sensor pin as input
  pinMode(greenLedPin, OUTPUT);  // Green LED pin as output
  pinMode(redLedPin, OUTPUT);    // Red LED pin as output
  pinMode(buzzerPin, OUTPUT);    // Buzzer pin as output
  pinMode(blueLedPin, OUTPUT);   // Blue LED pin as output
```

به وای فای متصل می شویم و چک می کنیم اگر اتصال درست بوده است.

```
  // Connect to Wi-Fi
  Serial.print("Connecting to Wi-Fi");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);  // Wait for Wi-Fi connection
    Serial.print(".");
  }
  Serial.println("\nWi-Fi connected!");  // Print message once connected
```

MQTT را کانفیگر می کنیم. و پورتش و سرورش را مشخص می کنیم.

```
  // Configure MQTT
  mqttClient.setServer(mqttServer, mqttPort);  // Set MQTT server and port
  mqttClient.setCallback(mqttCallback);        // Set MQTT callback function
  connectToMQTT();  // Call function to connect to MQTT
}

// Main program loop
void loop() {
```

سعی میکنیم اگر به MQTTمتصل نشدیم دوباره وصل شیم.

```
  // Check if MQTT is connected, if not, reconnect
  if (!mqttClient.connected()) {
    connectToMQTT();  // Reconnect if disconnected
  }
  mqttClient.loop();  // Handle MQTT communications
```

مقدار LDR را خوانده و به درصد تبدیل می کنیم.

```
  // Read LDR sensor value
  int ldrValue = analogRead(ldrPin);  // Read raw LDR value
  int ldrPercent = map(ldrValue, 0, 4095, 0, 100); // Convert to percentage (0-100)
  Serial.println(ldrPercent);  // Print LDR percentage
  Serial.println(ldrValue);  // Print raw LDR value
```

مقدار LDR را به ثینگز بورد ارسال می کنیم و زون آیدی هم که قبلا اشاره کردیم را لحاظ می کنیم.

```
  // Send LDR value to ThingsBoard
  sendToAllLdr(ldrPercent, 1); // ZoneID = 1
```

LEDرا بر اساس میزان نور LDR کنترل می کنیم. مثلا اگر ولیو LDR بیشتر از 200 باشد چراغ آبی خاموش می شود در غیر این صورت روشن می شود.

```
  // Control blue LED based on light level
  if (ldrValue > 200) {
    digitalWrite(blueLedPin, LOW);  // Turn off blue LED if light is above threshold
  } else {
    digitalWrite(blueLedPin, HIGH); // Turn on blue LED if light is below threshold
  }
```

مقدار مگنیتیک سنسور را می خوانیم و بر و استتوس های اوپن و کلوز را تعیین می کنیم.

```
  // Read magnetic sensor status
  int hw017Status = digitalRead(hw017Pin);  // Read the status of magnetic sensor
String status = (hw017Status == HIGH) ? "open" : "closed";  // Determine the status (open or
closed)
```

بر اساس استتوس پنجره چراغ ها و بازر (وسیله ای که صدا میدهد) را کنترل می کنیم.

```
  // Control LEDs and buzzer based on window status
  if (hw017Status == HIGH) {  // If magnetic sensor detects "open"
    digitalWrite(greenLedPin, HIGH);  // Turn on green LED
    digitalWrite(redLedPin, LOW);    // Turn off red LED
    digitalWrite(buzzerPin, HIGH);   // Activate buzzer
  } else {  // If magnetic sensor detects "closed"
    digitalWrite(greenLedPin, LOW);  // Turn off green LED
    digitalWrite(redLedPin, HIGH);   // Turn on red LED
    digitalWrite(buzzerPin, LOW);    // Deactivate buzzer
  }
```

وضعیت(اوپن کلوز پنجره) را به سرور MQTTمیفرستیم.

```
  // Send status (open/closed) to MQTT server
  String payload = String("{\"status\":\"") + status + "\"}";
  mqttClient.publish(mqttTopic, payload.c_str());  // Publish status to MQTT

  delay(2000); // Wait for 2 seconds before repeating the loop to avoid rapid sending
}
```

در این فانکشن تلاش می کنیم که به MQTTمتصل شویم. اگر کانکت نشدیم تلاش می کنیم دوباره کانکت بشویم.

```
// Function to connect to MQTT
void connectToMQTT() {
  while (!mqttClient.connected()) {  // While not connected to MQTT
    Serial.print("Connecting to MQTT...");
    if (mqttClient.connect("ESP32Client", mqttUser, "")) {  // Attempt to connect with device
token
      Serial.println("Connected to MQTT");
      mqttClient.subscribe(mqttTopic);  // Subscribe to the MQTT topic
    } else {
      Serial.print("Failed, rc=");  // Print error if connection fails
      Serial.print(mqttClient.state());
      delay(2000);  // Wait before retrying
    }
  }
}

// Callback function to handle received MQTT messages
```

```
void mqttCallback(char* topic, byte* payload, unsigned int length) {
  String message = "";
  for (int i = 0; i < length; i++) {
    message += (char)payload[i];  // Convert received payload to string
  }

  Serial.print("Received message: ");
  Serial.println(message);  // Print received message

  // Handle specific messages
  if (message == "open") {
    Serial.println("Circuit is open");  // If message is "open", print status
  } else if (message == "closed") {
Serial.println("Circuit is closed");  // If message is "closed", print status
```

نتیجه گیری: با استفاده از این پروژه می توانیم باز و بسته بودن در آزمایشگاه را از راه دور کنترل کنیم تا فرد نامطمئن وارد آزمایشگاه نشود.

فصل 2) پروژه دما و رطوبت

- مقدمه

به طور کلی مدار شماره 2 دارای دو وظیفه عمومی و اختصاصی است: عملکرد عمومی آن مانند پنج بورد اصلی مورد استفاده در پروژه هوشمندسازی آزمایشگاه، سنجش نور محیط محل استقرار آن بوده و وظیفه اختصاصی آن سنجش دما و رطوبت آزمایشگاه و ارسال این داده ها به پلتفرم ابری تینگزبورد است.

مطابق نیازمندی های پروژه داده های مذکور باید تحت پروتکل HTTP به پلتفرم ابری ارسال شوند.

2 - طراحی مدار

مدار شماره 2 شامل یک بورد ESP32-C3-MINI-1 است که ماژول ها و قطعات زیر به آن متصل شده اند:

- یک ماژول DHT11 که شامل سنسورهای دما و رطوبت بوده که به پین شماره 8 بورد ESP32 متصل شده است.
- یک قطعه LDR (فتورزیستور) جهت سنجش نور محیط که به پین شماره 2 ESP32 متصل شده است.
- دو مقاومت که جهت کنترل جریان ورودی به قطعات فوق روی مدار نصب شده اند.

3 - پیاده سازی برنامه

برنامه بورد ESP32 به زبان C++ نوشته شده و با Arduino IDE ساخته و روی آن بارگذاری شده است. در این بخش توضیح اجمالی کدهای نوشته شده آورده شده است.

3 – 1 – فراخوانی کتابخانه های مورد استفاده

```
#include <string>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Arduino_MQTT_Client.h>
#include <ThingsBoard.h>
#include <DHT.h>
```

کتابخانه های فوق در قسمت های مختلف برنامه استفاده شده اند که در ابتدای برنامه وارد می شوند.

3 – 2 – تعریف پین های متصل به قطعات حسگر

در این قسمت نوع ماژول DHT (که DHT11 است) و شماره پین ESP32 که پین سیگنال این ماژول به آن وصل شده مشخص گردیده و یک شیء از کتابخانه DHT بر اساس این مقادیر ساخته شده است. همچنین پین شماره 2 ESP32 به عنوان پینی که حسگر LDR به آن وصل شده مشخص شده است:

```
#define DHTTYPE DHT11
#define DHT_PIN 8
DHT dht(DHT_PIN, DHTTYPE);
const int ldrPin = 2;
```

3 – 3 – تنظیمات اتصال به شبکه وای-فای و پلتفرم ابری تینگزبورد

در این قسمت SSID و رمز ورود شبکه بیسیمی که بورد به آن متصل خواهد شد به همراه توکن دسترسی، آدرس سرور ابری و پورتهای مربوط به پروتکل های MQTT و HTTP، بیشترین اندازه پیام و نرخ تبادل داده سریال (جهت اشکال زدایی برنامه) مشخص شده است. همچنین آدرس API مربوط به ارسال داده های تله متری از طریق پروتکل HTTP در متغیر thingsboardServer ذخیره شده است:

```
constexpr char WIFI_SSID[] =  "A24";
constexpr char WIFI_PASSWORD[] = "amir0000";
const std::string TOKEN = "beegFN7ero2BkFexoZGv";
constexpr char THINGSBOARD_SERVER[] = "eu.thingsboard.cloud";
constexpr uint16_t THINGSBOARD_PORT = 1883U;
constexpr uint16_t THINGSBOARD_HTTP_PORT = 80U;
constexpr uint32_t MAX_MESSAGE_SIZE = 1024U;
constexpr uint32_t SERIAL_DEBUG_BAUD = 115200U;
const std::string thingsboardServer = "http://eu.thingsboard.cloud/api/v1/" + TOKEN + "/telemetry";
```

3 – 4 – تعریف اشیاء مربوط به کاربری شبکه و پلتفرم

از هر کدام از کلاسهای WiFiClient ، Arduino_MQTT_Client و ThingsBoard یک آبجکت جهت استفاده در طول برنامه ساخته و مقداردهی اولیه می گردد. لازم به ذکر است علت اینکه یک آبجکت به نام mqttClient ساخته شده این است که در قسمتی از برنامه آدرس مک بورد ESP32 توسط آبجکت تینگزبورد tb ، تحت پروتکل MQTT به سرور ابری فرستاده می‌شود.

```
WiFiClient wifiClient;
Arduino_MQTT_Client mqttClient(wifiClient);
ThingsBoard tb(mqttClient, MAX_MESSAGE_SIZE);
```

3 – 5 – محتوای تابع setup

در تابع setup که به عنوان تابع راه اندازی اولیه بورد کار می کند ابتدا به منظور یکسان سازی داده های ورودی از LDR (با توجه به اختلاف انواع آن که 10 بیتی یا 12 بیتی هستند) تمام داده های آنالوگ با فراخوانی تابع analogReadResolution با دقت 10 بیتی دریافت می شود. سپس ارسال داده های

سریال شروع شده و دستور آغاز به کار به ماژول DHT داده و پینی که LDR از طریق آن به بورد ESP32 وصل شده به حالت ورودی تنظیم می شود.

در انتها پس از یک انتظار یک ثانیه ای (جهت اطمینان از آماده بودن کامل اجزای سخت افزاری)، با فراخوانی تابع InitWiFi که در بخش 3-6 توضیح داده خواهد شد عملیات اتصال به شبکه بیسیم انجام می شود.

```
void setup() {
  analogReadResolution(10);
  Serial.begin(SERIAL_DEBUG_BAUD);
  dht.begin();
  pinMode(ldrPin, INPUT);
  delay(1000);
  InitWiFi();
}
```

3 – 6 – عملیات اتصال به شبکه بیسیم

جهت اتصال اولیه به شبکه وای فای یک تابع به نام InitWiFi پیاده سازی شده که طی آن تابع begin از کلاس WiFi با SSID و رمز شبکه فراخوانی می شود. در خروجی سریال ابتدا با یک پیام شروع اتصال به یک Access Point اعلام می شود و تا زمانی که وضعیت اتصال برقرار نشود، داخل یک حلقه شده و هر 500 میلی ثانیه یک نقطه چاپ می شود. پس از اتصال به access point نیز از حلقه خارج شده یک پیام مناسب در خروجی سریال چاپ می گردد.

```
void InitWiFi() {
  Serial.println("Connecting to AP ...");
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);

  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to AP");
}
```

برای بررسی اتصال به شبکه وای فای و اتصال مجدد به آن تابعی به نام reconnect پیاده سازی شده که طی آن در صورتی که وضعیت اتصال به وای فای برابر با WL_CONNECTED نباشد تابع InitWiFi فراخوانی می گردد:

```
bool reconnect() {
  if (WiFi.status() == WL_CONNECTED) {
    return true;
  }
  InitWiFi();
```

```
  return true;
}
```

3 – 7 – محتوای تابع loop

در تابع loop که به طور متوالی و حلقه وار توسط میکروکنترلر اجرا می شود تمام داده های مربوط به حسگرها به سمت سرور ابری تینگزبورد ارسال می شود. در این بخش به طور خلاصه عملکردهای کلی که در هر بار اجرای تابع loop رخ می دهد بررسی می شود.

ابتدا در هر بار اجرای حلقه loop، یک تأخیر 10 میلی ثانیه ای ایجاد شده و با فراخوانی تابع reconnect که در بخش قبل توضیح داده شد از برقرار بودن اتصال به شبکه اینترنت اطمینان حاصل می شود. علت کم بودن میزان تأخیر این است که به دلیل سنگین بودن عملیات ارسال پیام با پروتکل http فاصله بین داده های دریافت شده در سمت پلتفرم ابری نسبتاً زیاد (حداقل 2 ثانیه) شده است.

```
delay(10);
 if                                    (!reconnect())                                    {
   return;
 }
```

همان طور که در بخش 3 – 3 اشاره شد توسط یک آبجکت تینگزبورد به نام tb تحت پروتکل mqtt با پلتفرم ابری ارتباط برقرار شده و حداقل یک داده مانند آدرس مک کارت شبکه بیسیم میکروکنترلر به آن ارسال می شود. توکن دسترسی مورد استفاده مربوط به یک دیوایس متناظر در تینگزبورد با نام TEMP_HUMIDITY است.

```
 if (!tb.connected()) {
   Serial.print("Connecting to: ");
   Serial.print(THINGSBOARD_SERVER);
   Serial.print(" with token ");
   Serial.println(TOKEN.c_str());
   if (!tb.connect(THINGSBOARD_SERVER, TOKEN.c_str(), THINGSBOARD_PORT)) {
     Serial.println("Failed to connect");
     return;
   }
   tb.sendAttributeData("macAddress", WiFi.macAddress().c_str());
 }
```

در ادامه و همزمان با تعریف یک آبجکت HTTPClient به نام http (جهت ارسال داده ها با پروتکل HTTP) با فراخوانی تابع analogRead مقدار داده LDR خوانده شده و در متغیر عدد صحیح ldrValue ذخیره می شود. به همین ترتیب با فراخوانی توابع readHumidity و readTemperature از آبجکت dht (که قبلا تعریف شد) مقادیر دریافتی مربوط به رطوبت و دمای محیط به ترتیب در متغیرهای عدد اعشاری به نام های h و t ذخیره می گردد:

```
HTTPClient http;
 int ldrValue = analogRead(ldrPin);
```

```
  float h = dht.readHumidity();
  float t = dht.readTemperature();
```

مقادیر فوق به صورت یک رشته در قالب json در آمده و در متغیر رشته ای payload ذخیره می شود تا تحت پروتکل HTTP سمت سرور ابری تینگزبورد ارسال گردد:

```
String payload = String("{\"temperature\": ") + t + ", \"humidity\": " + h + ", \"light_intensity\": " + ldrValue + "}";
```

سپس تابع begin از آبجکت http با مقدار آدرس API پروتکل (قبلا اشاره شد که در thingsboardServer ذخیره شده) فراخوانی و بعد از اضافه شدن سرآیند (header) با نوع محتوای json ، مقدار ذخیره شده در payload با متد POST ارسال می شود:

```
http.begin(thingsboardServer.c_str());
  http.addHeader("Content-Type", "application/json");
  int httpResponseCode = http.POST(payload);
```

در صورتی که عملیات ارسال بسته HTTP موفقیت آمیز باشد (یعنی کد پاسخ ارسالی مثبت باشد) کد پاسخ دریافتی از سرور در خروجی سریال چاپ می شود و در غیر این صورت ناموفق بودن آن نیز در سریال چاپ می شود. در انتها نیز با فراخوانی تابع end کار آبجکت http به پایان می رسد.

```
  if (httpResponseCode > 0) {
    String response = http.getString();
    Serial.printf("HTTP Response code: %d", httpResponseCode);
    Serial.println(response);
  } else {
    Serial.printf("Error on sending POST: %s", http.errorToString(httpResponseCode).c_str());
  }
  http.end();
```

لازم به ذکر است به منظور ایجاد وحدت رویه در میان تمامی بوردهای راه اندازی شده در پروژه هوشمندسازی آزمایشگاه IOCPP در زمینه ارسال داده های LDR به عنوان عملکرد مشترک بین آنها، و همچنین جمع آوری داده های LDR ها در یک دیوایس متمرکز در محیط تینگزبورد، مقدار ldrValue از طریق تابعی به نام sendToAllLdr نیز ارسال می گردد که در بخش بعدی توضیح داده خواهد شد. در این جا صرفا به این مطلب بسنده می شود که دومین آرگومان ورودی به این تابع در واقع شماره مدار یا بورد جاری است:

```
sendToAllLdr(ldrValue, 2);
```

3 – 8 – ارسال داده های مربوط به نور محیط

به منظور ارسال مقادیر سنجش شده توسط حسگرهای نور (LDR) در نقاط مختلف آزمایشگاه، جمع آوری متمرکز این داده ها و نمایش آن ها در قالب یک ویجت گرافیکی، یک دیوایس در تینگزبورد به نام ALL_LDRS تعریف شده و توکن دسترسی آن به همراه ماژول/تابع مخصوصی که برای ارسال داده به آن تهیه شده در بین تمام برنامه های آپلود شده در بوردهای مختلف توزیع و به اشتراک گذاشته

شده است. در واقع با فراخوانی این تابع که با نام sendToAllLdr پیاده سازی شده است، هر کدام از مدارهای موجود در این پروژه که دارای حسگر LDR هستند مقدار نور محیط به همراه شماره قراردادی بورد خود را به آن پاس داده و این تابع عملیات تبدیل داده مذکور به واحد درصد و ارسال آن به سرور را انجام می دهد.

```
void sendToAllLdr(int value, int zoneId) {

  const std::string token = "IGa09xW2NYVHxcsaRvo8";

  const std::string tbServerAttributes = "http://eu.thingsboard.cloud/api/v1/" + token + "/attributes";

  const std::string tbServerTelemetry = "http://eu.thingsboard.cloud/api/v1/" + token + "/telemetry";

  int percentage = map(value, 0, 1023, 0, 100);

  String payload = String("{\"Zone") + zoneId + "\" : " + percentage + "}" ;

  // Serial.println(payload);

  httpPostJson(tbServerAttributes, payload);

  httpPostJson(tbServerTelemetry, payload);

}
```

در پیاده سازی تابع sendToAllLdr که به صورت فوق است ابتدا توکن دسترسی دیوایس ALL_LDRS در یک متغیر رشته ای به نام token ذخیره شده و با استفاد از دو آدرس API برای ارسال داده به حالت های telemetry و attribute تعریف می شود. سپس با فراخوانی تابع map مقدار ورودی نور محیط از بازه 0 تا 1023 به بازه 0 تا 100 نگاشت می شود تا قابلیت ارائه به صورت درصدی داشته باشد.

سپس در قالب json در یک رشته به نام payload ذخیره می شود که شامل یک کلید داده با نامی شبیه به Zone1 است. در واقع با توجه به اینکه 6 سنسور LDR در این پروژه استفاده شده و هر کدام از آن ها میزان نور یک محیط را می سنجند، اطلاعات میزان نور 6 ناحیه با عناوین Zone1, Zone2, ..., Zone6 به سرور ابری تینگزبورد ارسال می شود که دومین مقدار ورودی به تابع sendToAllLdr شماره ناحیه یا همان ناحیه است که در کلید داده داخل رشته payload بعد از کلمه Zone وارد می شود.

در انتها با دو بار فراخوانی تابع httpPostJson که پیاده سازی آن بسیار مشابه چیزی است که در بخش 3 – 7 اشاره شد و در انتهای این بخش کدهای آن آمده است، مقدار درصدی نور محیط (Zone) به آدرس های api تینگزبورد (یک بار به عنوان telemetry و یک بار به عنوان attribute) ارسال می شود.

علت اینکه داده های ldr به دو صورت ارسال می شود این است که جهت نگهداری به صورت timeseries و استفاده از آن ها در ویجت Real-Time Light Meter، باید به صورت telemetry ارسال گردد؛ اما با توجه به اینکه در بورد شماره 4 برای دریافت مقادیر ldr از پروتکل http استفاده شده است و در پروتکل http امکانی برای دریافت داده های تله متری نیست و صرفا می تواند داده های attribute را با متد GET دریافت نمود، داده های LDR به سمت API حالت attribute نیز ارسال می گردد.

```
int httpPostJson(std::string url, String data) {

    HTTPClient http;
```

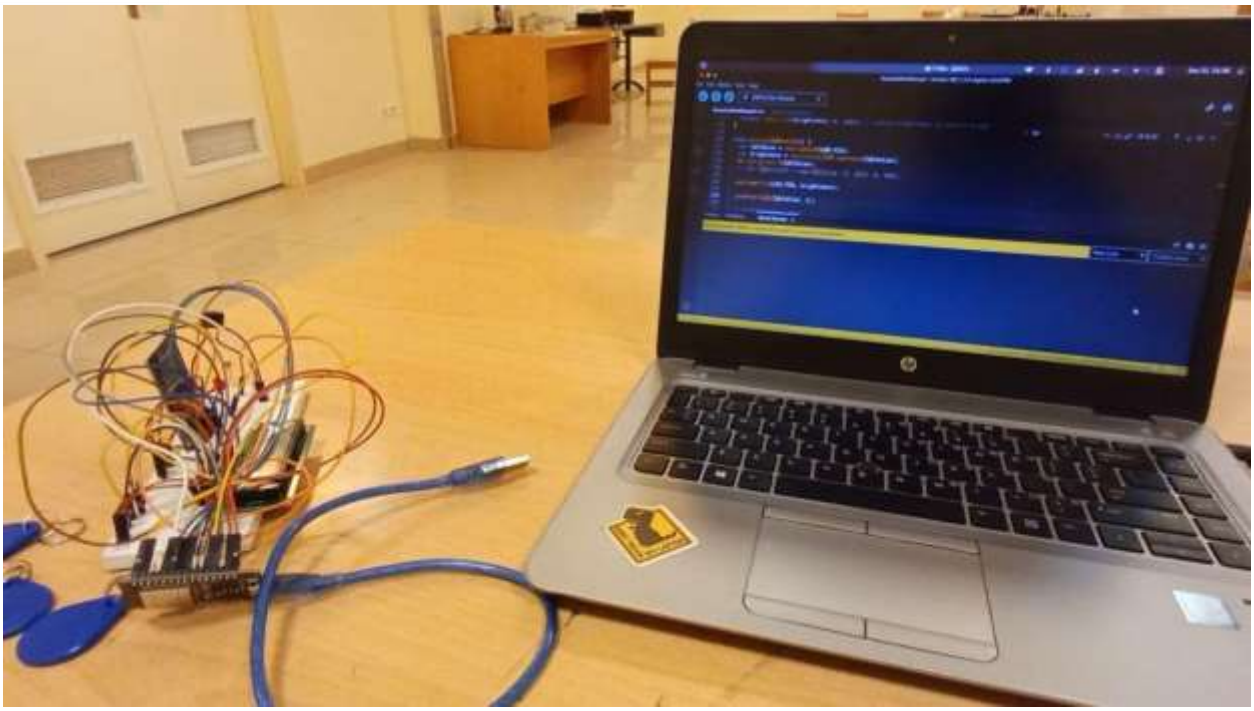
```
    http.begin(url.c_str());
    http.addHeader("Content-Type", "application/json");
    int httpResponseCode = http.POST(data);
    if (httpResponseCode > 0) {
      String response = http.getString();
      // Serial.printf("HTTP Response code: %d", httpResponseCode);
      // Serial.println(response);
    } else {
      Serial.printf("Error on sending POST: %s", http.errorToString(httpResponseCode).c_str());
    }
    http.end();
    return httpResponseCode;
}
```

4 – نتیجه گیری

علیرغم اینکه برنامه ریزی و راه اندازی این مدار با موفقیت انجام شده و سعی شده حتی المقدور برنامه نویسی به صورت بهینه انجام شود، لیکن در زمان پیاده سازی مدار مشکلاتی نظیر عدم یکسان بودن و کالیبره بودن قطعات LDR بروز کرد..

همچنین به نظر می رسد به خاطر الزامی که در نیازمندی های پروژه جهت ارسال داده های رطوبت، نور و دما با پروتکل HTTP وجود داشت، با توجه به اینکه این پروتکل نسبت به MQTT سنگین تر است، در بلندمدت از نظر مصرف انرژی بازدهی کمتری خواهد داشت.

فصل 3) پروژه RFID یا حضور غیاب اعضای آزمایشگاه



مقدمه: در این پروژه اسم اعضای آزمایشگاه (امیرحسین، پویا، قشمی، آیدا، عسل، علیرضا و مبینا) به صورت یک لیست در کنار uid مربوطه به آنها ذخیره شده است با استفاده از کتابخانه rfid و قسمت اسکنر آن می‌توانیم تگ‌های وارد شده را با این لیست را مقایسه کنیم اسم آن فرد را در قسمتی که کد وجود دارد ذخیره می کنیم که با گرفتن نام فرد، آن را بر روی ال سی دی نمایش می‌دهد حال در این پروژه یک بازر هم وجود دارد با ورود هر فرد دو بوق و با خروج هر فرد یک بوق می‌زند و فانکشنی که برای این کار استفاده می‌شود از ویژگی دیجیتال برد استفاده می‌کند و یک پالس ۱ به سمت پین بازر می‌فرستد و صدای بازر به صدا در می‌آید فانکشن‌هایی که در قسمت‌های مختلف این پروژه هم استفاده شده است قابلیت فرستادن تمامی داده‌های ال دی آر به سمت داشبورد را دارد .
ما در اینجا از ستاپ وای فای برای گرفتن رمز و نام کاربری وای فای استفاده می‌کنیم و با استفاده از این فانکشن ستاپ وای فای را راه اندازی می‌کنیم سپس با استفاده از فانکشن آر اس آی دی، ال سی دی را ستاپ می‌کنیم. ستاپ ال سی دی به این صورت عمل می کند که از طریف گرفتن پین های ریست و 16 نشان دادن آماده بودن شروع به کار می‌کند در تابع ستاپ آر اف آیدی با استفاده از کتابخانه SPI پروتکل پین SCK و SLK ریست 23 SS PIN را می‌گیرد.
توضیحات کد:

از کتابخانه مورد نظر استفاده می کنیم

```
 // Main source file
#include "header_libraries.h"
```

به وای فای وصل می شویم و رمز عبور و آدرس وای فای را قرار می دهیم و پورت 1883 را انتخاب می کنیم. توکن گرفته شده از ثینگز بورد را قرار می دهیم و ثینگز بورد نسخه کلود اروپایی را انتخاب می کنیم.

```
// WiFi credentials
const char* SSID = "A24";
const char* PASS = "amir0000";
const char* TOKEN = "38mLj5HzRKh6uHJr8baA";
const char* TB_SERVER = "eu.thingsboard.cloud";
constexpr uint16_t TB_PORT = 1883U;
```

کلاینت ثینگز برد را تنظیم می کنیم و از پروتکل MQTT استفاده می کنیم.

```
// ThingsBoard client setup
WiFiClient wifiClient;
Arduino_MQTT_Client mqttClient(wifiClient);
ThingsBoard tb(mqttClient);
```

پین ها مربوط به LCD را تعریف می کنیم.

```
// LCD pins
#define RS 21
#define EN 2
#define D4 32
#define D5 33
#define D6 25
#define D7 26
#define V0 13  //Analog Pin for Brightness of LCD
```

پین های مربوط به سنسور RFID را تعریف می کنیم.

```
// RFID reader pins
#define SS_PIN 5
#define RST_PIN 22
```

پین های مربوط به LDR را تعریف می کنیم.

```
// LDR and LED pins
#define LDR_PIN 34
#define LED_PIN 4
```

پین مربوط به بازر (بوق) را تعریف می کنیم.

```
#define BUZZER_PIN 27  // Buzzer connected to pin 27
```

آبجکت های LCD و RFIDرا تعریف می کنیم.

```
// LCD and RFID objects
LiquidCrystal lcd(RS, EN, D4, D5, D6, D7);
MFRC522DriverPinSimple ss_pin(SS_PIN);
MFRC522DriverSPI driver(ss_pin);
MFRC522 rfid(driver);
```

اسامی اعضای آزمایشگاه را تعیین می کنیم و دو نفر مهمان طبق قوانین آزمایشگاه تعریف می کنیم که با کارت مهمان بتوانند وارد آزمایشگاه شوند.

```
// Authorized cards
struct AuthorizedCard {
  const char* uid;
```

```
  const char* name;
  bool entered;
};



AuthorizedCard authorizedCards[] = {
  {"73:41:F3:0F", "Mobina", false},
  {"23:31:20:F8", "Asal", false},
  {"33:D6:0A:13", "Ayda", false},
  {"B3:92:D5:F7", "Alireza", false},
  {"43:F7:40:13", "Mahdi", false},
  {"C3:C1:16:F8", "Pouya", false},
  {"43:E2:30:F8", "Amir", false},
  {"53:BE:66:19", "Guest 1", false},
  {"73:B6:F1:0F", "Guest 2", false},

};
const int numAuthorizedCards = sizeof(authorizedCards) / sizeof(authorizedCards[0]);


// Multiplier for LED brightness
constexpr float multiplier = 0.1;
```

کانکشن وای فای را بررسی می کنیم. و سعی می کنیم از طریق SSID به وای فای وصل شویم.

```
void initWiFi() {
  Serial.println("Connecting to AP...");
  WiFi.begin(SSID, PASS);
  while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
  }
  Serial.println("Connected to WiFi");
  Serial.print("Local IP Address: ");
  Serial.println(WiFi.localIP());
}
```

مقدار LDRر ا به درصد تبدیل می کنیم و درصد را ریترن می کنیم

```
int convertLDRToPercent(int ldrValue) {
    // Map the LDR value from 0-1023 to 0-100
    int percentage = map(ldrValue, 0, 1023, 0, 100);
    return percentage;
}
```

در تابع ست آپ وای فای آن را ست می کنیم.

```
void setupWiFi() {
  initWiFi();
}
```

ال سی دی را ست می کنیم و قبلش در ال سی دی پیغامی را مبتنی بر آماده بودن RFID چاپ می کنیم و بعد 2 ثانیه پیغام را پاک می کنیم.

```
void setupLCD() {
  lcd.begin(16, 2);
  lcd.print("RFID Ready");
  delay(2000);
  lcd.clear();
}
```

RFID رت تعریف می کنیم و از آنالوگ رایت استفاده می کنیم و پین مود آن را اوت پوت می گذاریم.

```
void setupRFID() {
  pinMode(V0, OUTPUT);
  analogWrite(V0, 255);
  SPI.begin(18, 19, 23, SS_PIN);
  rfid.PCD_Init();
}
```

پین های LDR و LED و بازر و مودهای آن ها(اوت پوت، این پوت) را مشخث می کنیم.

```
void setupPINS() {
  pinMode(LDR_PIN, INPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUZZER_PIN, OUTPUT);
}
```

با استفاده از تابع بولین استتوس وای فای را بررسی می کنیم که اگر وای فای به خوبی کانکت شده باشد true و اگر کانکت نشده باشد false برگرداند.

```
bool reconnect() {
  if (WiFi.status() == WL_CONNECTED) {
    return true;
  }
  initWiFi();
  return true;
}
```

اگر که کارت معتبری دریافت کنیم باید trueبرگردانده شود در غیر این صورت False ریترن می شود.

```
bool authorizeCard(const String& uidStr, const char*& personName, bool*& enteredStatus) {
  for (int i = 0; i < numAuthorizedCards; i++) {
    if (uidStr == authorizedCards[i].uid) {
      personName = authorizedCards[i].name;
      enteredStatus = &authorizedCards[i].entered;
      return true;
    }
  }
  return false;
}
```

شدت روشنایی led یک عدد بین 0 تا 255 است. باید عدد محاسبه شود و عددی بین 0 و 255 برگردانده شود.

```
int calculateLEDBrightness(int ldrValue) {
  int brightness = 255 - (ldrValue * multiplier); // More light = Dimmer LED
  return constrain(brightness, 0, 255); // Ensure brightness is within 0-255
}
```

دداده ها را از LDR می خوانیم و بر اساس آن میزان روشنایی LED را تنظیم می کنیم.

```
void processLDRAndLED() {
  int ldrValue = analogRead(LDR_PIN);
  int brightness = calculateLEDBrightness(ldrValue);
  Serial.println(ldrValue);
  // int ldrprecent = map(ldrValue ,0, 1023, 0, 100);

  analogWrite(LED_PIN, brightness);

  sendToAllLdr(ldrValue, 3);
```

تلمنتری دیتا را ارسال می کنیم و از طریق پرینت سریال در سریال نمایش می دهیم.

```
  // Send telemetry data
  // tb.sendTelemetryData("ldr_value", ldrValue);
  // tb.sendTelemetryData("led_brightness", brightness);

  // Serial.print("LDR Value: ");
  // Serial.print(ldrValue);
  // Serial.print(" | LED Brightness: ");
  // Serial.println(brightness);
}
```

در فانکشن PROCCESSRFID درواقع اگر کارتی معتبر برای اولین بار چک شود ورود می زند و اگر برای دومین بار چک شود خروج می زند.

```
void processRFID() {
```

```
  if (!rfid.PICC_IsNewCardPresent()) return;
  if (!rfid.PICC_ReadCardSerial()) return;

  // Read UID and format it as a string
  String uidStr = "";
  for (byte i = 0; i < rfid.uid.size; i++) {
    uidStr += String(rfid.uid.uidByte[i] < 0x10 ? "0" : "");
    uidStr += String(rfid.uid.uidByte[i], HEX);
    if (i != rfid.uid.size - 1) uidStr += ":";
  }
  uidStr.toUpperCase();
```

معتبر بودن عضو از طریق کارتش چک می شود که اگر کاربر نامعتبر است متوجه شویم و روی LCD نمایش داده شود. اگر هم معتبر است نام کاربر نمایش داده می شود. در اخر هم LCD برای نمایش داده های جدید پاک می شود.

```
  // Authorization check
  bool isAuthorized = false;
  const char* personName = "";
  bool* enteredStatus = nullptr;

  isAuthorized = authorizeCard(uidStr, personName, enteredStatus);



  lcd.clear(); // Clear the LCD for a new message

  if (isAuthorized) {
    // Toggle entry/exit status
    *enteredStatus = !(*enteredStatus);
    lcd.setCursor(0, 0);
    lcd.print(personName);
    lcd.setCursor(0, 1);
    lcd.print(*enteredStatus ? "Entered" : "Exited");
```

تلمنتری دیتا برای نمایش به ثینگزبورد ارسال می شود. اطلاعاتی از جمله اسم فرد و ... از Uid استفاده می شود. بازر هم بسته به حالت ورود یا خروج 2 یا یکبار به صدا در میاید.

```
    // Send telemetry data to ThingsBoard
    // tb.sendTelemetryData("name", personName);
    // tb.sendTelemetryData("event", *enteredStatus ? "Entered" : "Exited");
    String data = String(personName) + " --- " + String(*enteredStatus ? "Entered" :
"Exited");  // String("{\"name\":\"")+personName + "\", \"event\":\""+ (*enteredStatus ?
"Entered" : "Exited") + "\"}";
    const char* cdata = data.c_str();
    tb.sendTelemetryData("data",cdata);
    Serial.println(data);
    // tb.sendTelemetryData("UID", uidStr.c_str());
    if(*enteredStatus){
      beepBuzzer(100);
      delay(100);
      beepBuzzer(100);
    }
    else{
      beepBuzzer(100);
    }
```

در سریال اطلاعات را نمایش می دهیم.

```
    Serial.print("Card UID: ");
    Serial.print(uidStr);
    Serial.print(" -- ");
    Serial.print(personName);
    Serial.print(" has ");
    Serial.println(*enteredStatus ? "Entered" : "Exited");
  } else {
```

در ال سی دی شرایط آثرایزد یا آن آثرایزد بودن کاربر و اطلاعات را نمایش می هیم.

```
    // Display not allowed message
    String  data = String("Not Allowed");
    Serial.println(data);
    const char* cdata = data.c_str();
    tb.sendTelemetryData("data",cdata);
    lcd.setCursor(0, 0);
    lcd.print("Not Allowed!");
    lcd.setCursor(0, 1);
    lcd.print(uidStr);
    Serial.print("Denied - Card UID: ");
    Serial.println(uidStr);
  }

  // Halt PICC and stop crypto
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
  delay(2000); // Delay to allow the message to be displayed on the LCD
}
```

تلمنتری را آپدیت می کنیم.

```
void updateTelemetry() {
  tb.loop(); // Process ThingsBoard client tasks
}
```

بوزر بوق می زند.

```
void beepBuzzer(int duration){
  digitalWrite(BUZZER_PIN, HIGH);
  delay(duration);
  digitalWrite(BUZZER_PIN, LOW);
}
```

داده ها را به فرمت جیسون از طریق پروتکل http سند می کنیم. هدر هم بهش اظافه می کنیم و ریسپانس کد را بررسی می کنیم که وجود داشته باشد.

```
int httpPostJson(std::string url, String data) {
    HTTPClient http;
    http.begin(wifiClient, url.c_str());
    http.addHeader("Content-Type", "application/json");
    int httpResponseCode = http.POST(data);
    if (httpResponseCode > 0) {
      String response = http.getString();
      // Serial.printf("HTTP Response code: %d", httpResponseCode);
      // Serial.println(response);
    } else {
      Serial.printf("Error on sending POST: %s", http.errorToString(httpResponseCode).c_str());
    }
    http.end();
    return httpResponseCode;
}
```

از طریق شرایط داده LDR را میزان نورش را در ثینگز بورد مشخص می کنیم. که در بخش ثینگز بورد مفصل تر به این قضیه پرداخته می شود.

```
void sendToAllLdr(int value, int zoneId) {
  const std::string token = "IGa09xW2NYVHxcsaRvo8";
  const std::string tbServerAttributes = "http://eu.thingsboard.cloud/api/v1/" + token +
"/attributes";
  const std::string tbServerTelemetry = "http://eu.thingsboard.cloud/api/v1/" + token +
"/telemetry";
  int percentage = map(value, 0, 1023, 0, 100);
  String payload = String("{\"Zone") + zoneId + "\" : " + percentage + "}"
;//String("{\"temperature\": ") + t + ", \"humidity\": " + h + ", \"light_intensity\": " +
ldrValue + "}";
  Serial.println(payload);
  httpPostJson(tbServerAttributes, payload);
  httpPostJson(tbServerTelemetry, payload);
}
```

```
void setup() {
  analogReadResolution(10);
  Serial.begin(115200);
  setupWiFi();
  setupRFID();
  setupLCD();
  setupPINS();
  Serial.println("Place your RFID card near the reader...");
}
```

وضعیت کانکشن به ثینگزبورد را بررسی میکنیم.

```
void loop() {
  if (!reconnect()) {
    return;
  }

  if (!tb.connected()) {
    if (!tb.connect(TB_SERVER, TOKEN, TB_PORT)) {
      Serial.println("Failed to connect to ThingsBoard.");
      return;
    }
    Serial.println("Connected to ThingsBoard server.");
  }

  processRFID();
  processLDRAndLED();
  updateTelemetry();
}
```

نتیجه گیری: این پروژه برای حضور غیاب و جلوگیری از ورود افراد نامعتبر به آزمایشگاه و نظم دهی بیشتر به اعضا است.

فصل 4) سنسور گاز

مقدمه: هدف از این پروژه اندازه گیری گاز کربن دی اکسید در محیط و ارسال هشدار در صورت افزایش میزان آن است که منجر به افزایش ایمنی اعضای آزمایشگاه می شود. از سنسور MQ2 برای اندازه گیری میزان گاز محیط استفاده شده است. اگر کربن دیکستید در فضا از مقداری بیشتر شود چراغ چشمک زن می شود. درواقع کربن دی اکسید محیط اندازه گیری و نشان داده می شود.

پین آنالوگ برای MQ2 استفاده می شود و از پروتکل MQTT استفاده می شود. از برد ESP32 DEAF استفاده شده است. در خط اول از کتابخانه وای فای و پابلیش سابسکرایب برای مدیریت شبکه و ارسال داده‌ها استفاده شده است سپس پایه‌ها را مشخص کردیم پایه شماره ۳۴ برای سنسور گاز پایه شماره ۲۵ برای ال ای دی و پایه شماره ۲۶ برای بازر که اگر مقدار سنسور بالاتر از ۷۰ برود ال ای دی قرمز روشن میشه و بازر صدا میده و این برنامه داده را که گاز محیط است را می سنجد و آن را در بازه صفر تا ۱۰۰ به ثینگزبورد ارسال می‌کند.
بعد از آن شخصات شبکه را وارد کردیم تا به آن شبکه وصل بشود در خط پایینی آدرس سرور و توکن را برای ارسال داده‌ها استفاده کردیم در تنظیمات پین‌هایی را که مشخص کرده بودیم را قرار دادیم تا وضعیت آنها مشخص شود که آیا ورودی یا خروجی است در قسمت لوپ برنامه سنسور ام کیو ۲ از محیط را می‌سنجد و در بازه 0 تا 4095 مقدار دهی می‌شود و با استفاده از مپ این بازه را بین صفر تا ۱۰۰ قرار دادیم که اگر مقدار بالاتر از ۷۰ برود ال ای دی چشمک زن شود و اگر پایین‌تر بود خاموش شود این مقدار ما توسط پروتکل ام کیو تی تی به ثینگزبورد ارسال می‌شود و مقدار در داشبورد نمایش داده می‌شود.

توضیحات کد:

کتابخانه ها برای وای فای و اتصال از طریق MQTT را مشخص می کنیم.

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

پین ها را مشخص می کنیم.

```
const int mq2Pin = 34;

const int redLedPin = 25;
```

مشخصات وای فای را تعریف می کنیم.

```
const char* ssid = "Alireza";
const char* password = "123456789";
```

مشخصات سرور که از ثینگزبورد استفاده کردیم و توکن ثینگز بورد را قرار می دهیم.

```
const char* mqtt_server = "mqtt.eu.thingsboard.cloud";
const char* mqtt_username = "frIRtOaKzBqt0rNVqCtK";
```

کلاینت وای فای را مشخص می کنیم.

```
WiFiClient espClient;
PubSubClient client(espClient);
```

در تابع ست آپ سرعت دیباگ را 115200 در نظر می گیریم و پین مورد ها را مشخص کرده و وای فای را ست آپ می کنیم و همچنین از پین 1883 برای MQTT استفاده می کنیم.

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


  pinMode(mq2Pin, INPUT);
  pinMode(redLedPin, OUTPUT);

  setup_wifi();

  client.setServer(mqtt_server, 1883);
}
```

اگر کلاینت کانکت نباشد سعی می کنیم کانکت شویم.

```
void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();
```

سنسور گاز را از طریق Analogread میخوانیم و از مپ استفاده می کنیم همانطور که در ابتدا توضیح دادیم و بر اساس آن led هارو کنترل می کنیم.

```
  int mq2Value = analogRead(mq2Pin);

  int scaledValue = map(mq2Value, 0, 4095, 0, 100);
  scaledValue = constrain(scaledValue, 0, 100);

  Serial.print("Raw MQ-2 Value: ");
  Serial.print(mq2Value);
  Serial.print(" -> Scaled Value: ");
  Serial.println(scaledValue);

  if (scaledValue > 70) {
    digitalWrite(redLedPin, HIGH);
    delay(500);
    digitalWrite(redLedPin, LOW);
    delay(500);
  } else {

    digitalWrite(redLedPin, LOW);
  }

  sendMQ2Value(scaledValue);

  delay(1000);
}
```

بررسی می کنیم که آیا وای فای به خوبی متصل شده است. در غیر این صورت تلاش می کنیم که دوباره متصل شویم.

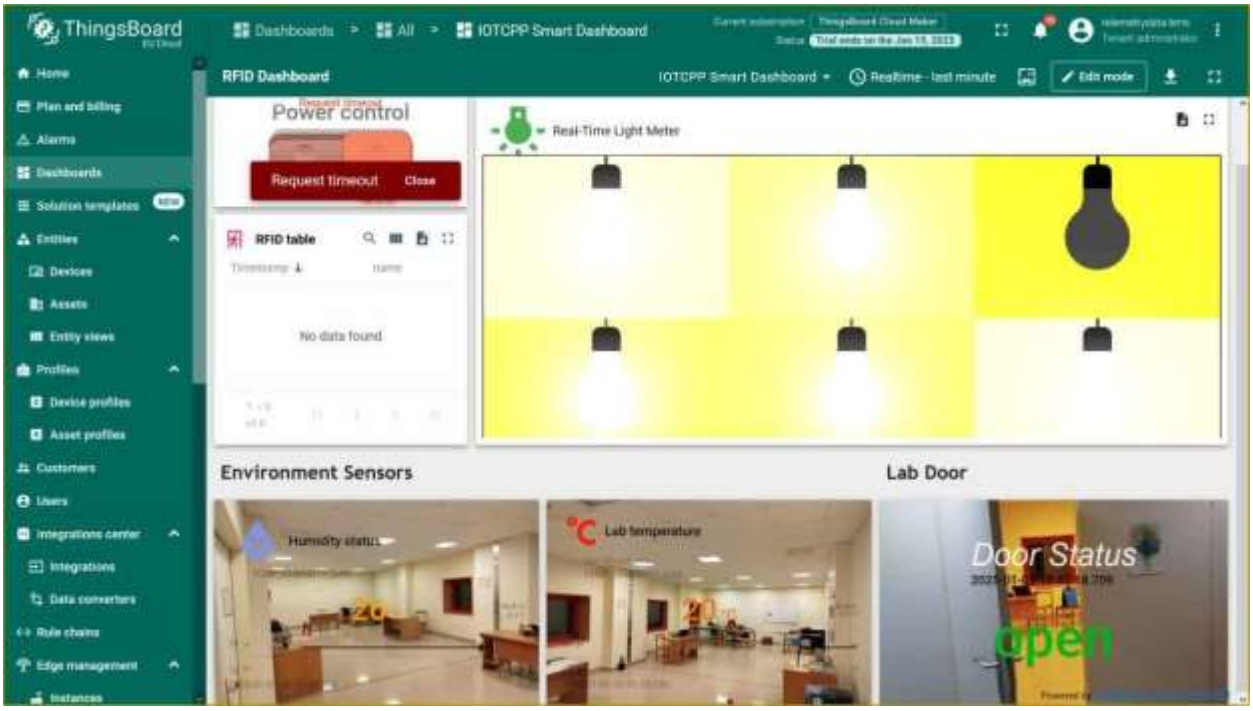
```
void setup_wifi() {
  delay(10);
  Serial.println();
  Serial.print("Connecting to WiFi...");
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to WiFi");
}
void reconnect() {
  while (!client.connected()) {
    Serial.print("Attempting MQTT connection...");
    if (client.connect("ESP32Client", mqtt_username, NULL)) {
```

```
      Serial.println("connected");
    } else {
      Serial.print("failed, rc=");
      Serial.print(client.state());
      Serial.println(" try again in 5 seconds");
      delay(5000);
    }
  }
}
```

مثدار سنسور 2MQ را سند می کنیم.

```
void sendMQ2Value(int value) {
  String payload = "{\"mq2\": " + String(value) + "}";
  client.publish("v1/devices/me/telemetry", payload.c_str());
}
```

فصل 5) ثینگز بورد



مقدمه: یک environment sensor داریم پایین سمت چپ و وسط صفحه که دما و رطوبت آزمایشگاه را اندازه گیری میکند با یک سنسور هم دما و هم رطوبت را اندازه گیری می کنیم که در کد همان DhT است. سنسور مگنتیک (پایین سمت راست صفحه) هم وضعیت باز و بسته شدن در را نشان می دهد اگر در باز باشد open و اگر در بسته باشد به صورت closed نشان می دهد. یک RFID TABLE برای ورود و خروج اعضای آزمایشگاه وجود دارد که اسامی افراد را بر اساس ورود و خروجشان نمایش می دهد. دو نفر مهمان تعریف شدند که تا سقف دو نفر آزمایشگاه مهمان می پذیرد(با توجه به قوانین آزمایشگاه) اما افراد دیگر به صورت افراد غیر مجاز تعریف شدند. در قسمت بالا سمت راست Real Time Light Meter قرار دارد که تمام LDR هایی که در نواحی مختلف آزمایشگاه قرار دادیم و میزان نورشان را نمایش میدهیم. یک power control داریم که قضیه این ویجت با بقیه ویجت ها فرق دارد. در ویجت های قبلی ما داده از سخت افزار به ثینگزبورد ارسال می کردیم اما از این طریق می توانیم سخت افزار های قرار داده شده در آزمایشگاه را خاموش یا روشن کنیم.

توضیحات ویجت LED :

1 - مقدمه

ویجت Real-Time Light Meter یکی از ویجت های داشبورد طراحی شده در پروژه هوشمندسازی آزمایشگاه است که به صورت اختصاصی طراحی و تولید شده است. هدف اصلی این ویجت نمایش شدت نور نقاط مختلف آزمایشگاه است که از جمله نتایج مورد انتظار در مستند تعریف پروژه پایانی درس سیستم ها و کاربردهای هوشمند بود. هدف دوم از طراحی آن نمایش وضعیت روشن یا خاموش بودن چراغ های نواحی مختلف آزمایشگاه است که توسط بورد شماره 4 انجام می شود.

2 – مراحل پیاده سازی ویجت

پیاده سازی این ویجت در سه مرحله انجام گرفت. در مرحله اول با طراحی یک گرید 2 در 3 در قالب HTML (مطابق تصویر زیر) تلاش شد تا از طریق API های تینگزبورد اطلاعات مربوط به داده های LDR دریافت و نمایش داده شود. با توجه به اینکه نوعاً داده های LDR به صورت telemetry به سرور ارسال و ذخیره شده بود این راهکار به نتیجه نرسید.

zone1 40%	zone2 70%	zone3 55%
zone4 20%	zone5 90%	zone6 60%

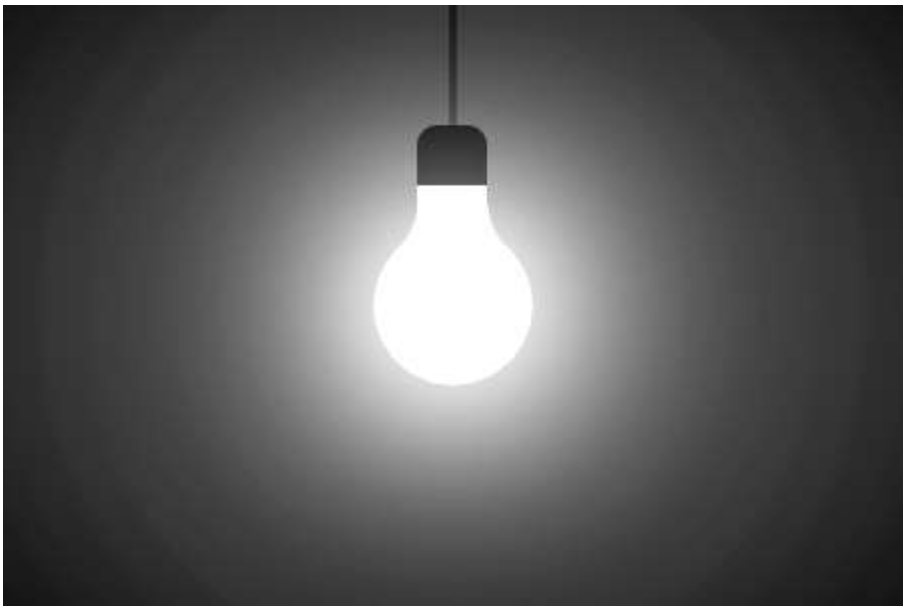
در مرحله دوم پیاده سازی پس از تحقیق درباره نحوه دریافت داده های پاس داده شده به ویجت ها در داشبورد تینگزبورد، از داده های آبجکت self.ctx که پس از انجام تنظیمات منابع داده در داشبورد به ویجت پاس داده می شود، استفاده شد. نتیجه این مرحله طراحی ویجت چیزی شبیه به تصویر زیر بود که در آن همزمان با نمایش عددی درصد نور یک ناحیه، روشنایی پس زمینه آن ناحیه نیز تغییر می کند.

Zone1: **44.72 %**	Zone2: %	Zone3: %
Zone4: %	Zone5: %	Zone6: %

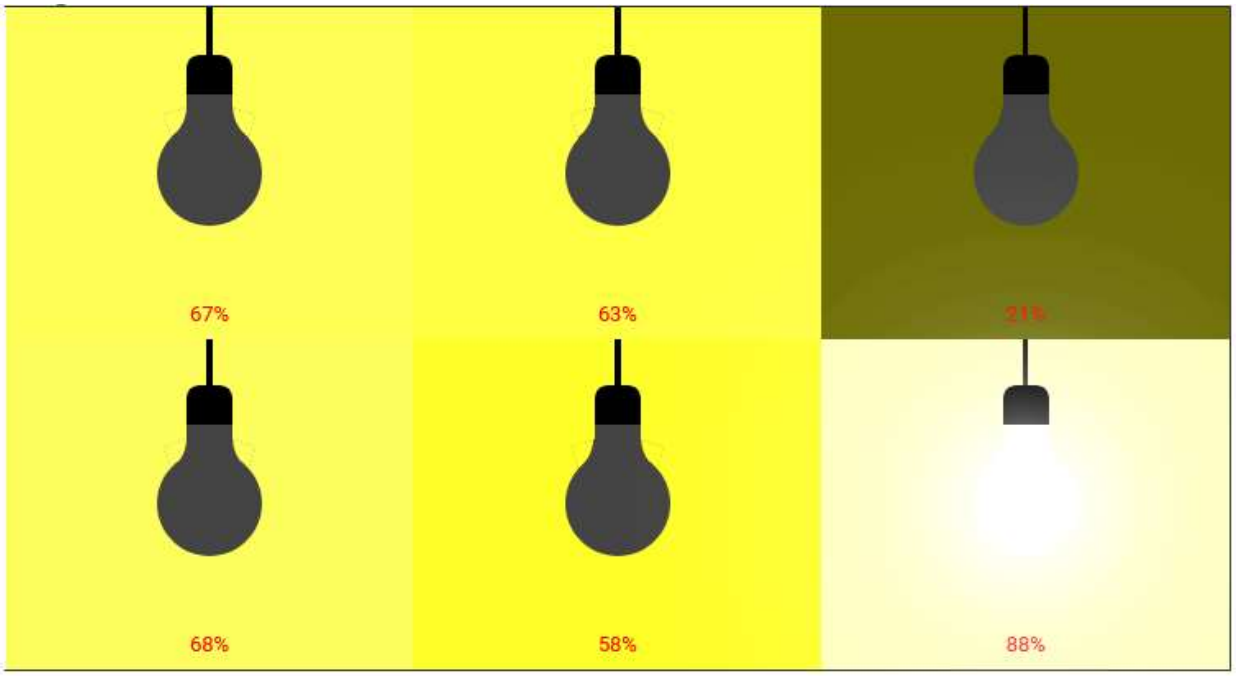
Last update: --

در مرحله سوم طراحی ویجت با بهره برداری و الگوگیری از یک طراحی چراغ روشنایی معرفی شده در یک مطلب سایت Mdeium (لینک زیر) نحوه نمایش ویجت دستخوش تغییرات اساسی شده و امکان جدیدی جهت نمایش وضعیت روشن یا خاموش بودن چراغ های نواحی مختلف آزمایشگاه به آن اضافه گشت.

https://enlear.academy/how-to-create-an-animated-light-bulb-with-javascript-css-4fa96eaf40bc



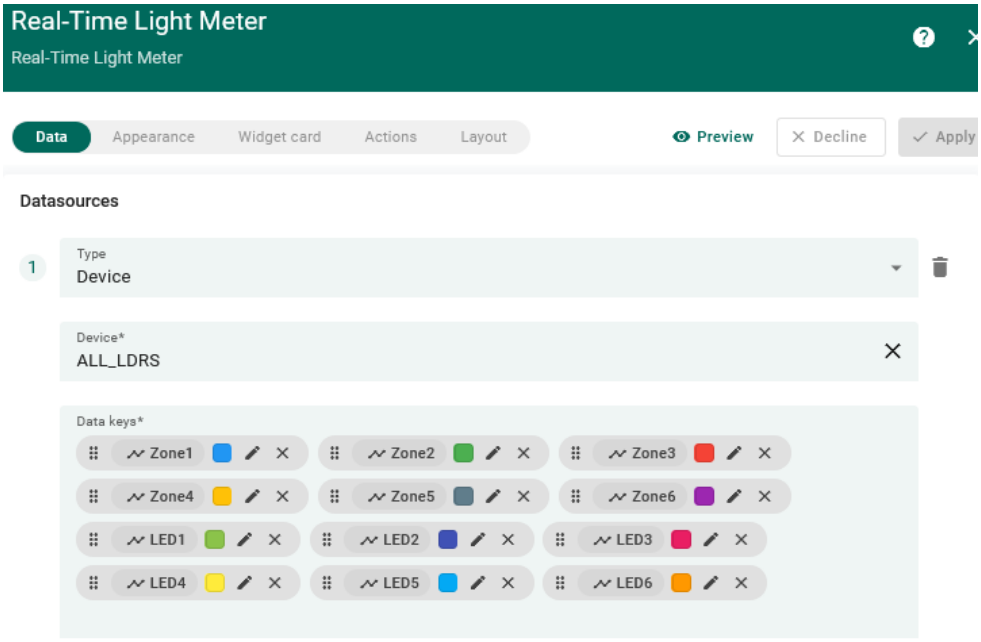
شکل اولیه طراحی به صورت فوق بود که پس از توسعه در تینگزبورد نتیجه قرارگیری و اجرای آن در تینگزبورد به صورت زیر درآمد:



3- منابع داده ویجت

منبع داده ویجت Real-Time Light Meter یک دیوایس به نام ALL_LDRS بوده که 12 کلید داده تله متری از آن با عناوین Zone1 ... Zone6 و LED1 ... LED6 به ویجت پاس داده می شود. شش

کلید داده اول مربوط به میزان روشنایی سنجیده شده توسط LDR ها توسط 6 بورد اصلی پروژه بوده و مقادیر آن از 0 تا 100 بر حسب درصد است و شش کلید داده دوم مربوط به وضعیت روشن یا خاموش بودن LED های متناظر با چراغ ها با مقادیر 1 یا 0 بوده که توسط بورد شماره 4 به دیوایس ارسال شده است.



4 – طراحی داخلی ویجت

ویجت Real-Time Light Meter آخرین داده های ارسالی (Latest values) هر کدام از کلید داده ها را نمایش می دهد. طراحی داخلی آن شامل سه بخش HTML ، CSS و JavaScript است.

4 – 1 – بخش HTML و CSS

همان گونه که در بخش 2 اشاره شد پایه طراحی این ویجت یک کد آماده از سایت Medium بوده که به تعداد نواحی شش گانه توسعه یافته است:

```
<div class="room-bulbs">
    <div class="zone" id="zone1" style="background:hsl(60,100%,{{value1}}%);">
        <div class="light">
            <div class="wire"></div>
            <div class="bulb">
                <span></span>
                <span></span>
            </div>
```

```
            </div>
            <span class="percent">{{value1}}%</span>
        </div>

        <div class="zone" id="zone2" style="background:hsl(60,100%,{{value2}}%);">
        ...
        </div>
        ...
</div>
```

در واقع 6 بخش با کلاس zone ساخته شده و علاوه بر استایل های موجود در بخش CSS برای رنگ پس زمینه هر zone از کد رنگ های hsl استفاده می گردد و مقدار درصد روشنایی آن رنگ با متغیرهایی مثل value1, value2, ... مشخص می شود که از طریق جاوااسکریپت مقداردهی خواهد شد.

در بخش CSS نیز علاوه بر استایل های موجود در الگوی اولیه ابعاد کلاس های zone و ... و تغییر یافته و محل کلاس percent نیز مشخص شده است:

```
.zone{display: flex;justify-content: center;align-items: center;min-height: 100vh;background: #222;} .zone.on{background: radial-gradient(#555,#111);}

.bulb{position: relative;width: 80px;height: 80px;background: #444;border-radius: 50%;z-index: 2;}.zone.on .bulb{background: #fff;box-shadow: 0 0 50px #fff,0 0 100px #fff,0 0 150px #fff,0 0 280px #fff,0 0 250px #fff,0 0 300px #fff,0 0 350px #fff;}.bulb::before{content: '';position: absolute;top: -50px;left: 22.5px;width: 35px;height: 80px;background: #444;border-top: 30px solid #000;border-radius: 10px;}.zone.on .bulb::before{background: #fff;}.zone.on .bulb::after{content: '';position: absolute;top: 50%;left: 50%;transform: translate(-50%,-50%);width: 120px;height: 120px;background: #fff;border-radius: 50%;filter: blur(40px);}.bulb span:nth-child(1){position: absolute;top: -16px;left: -4px;display: block;width: 30px;height: 30px;background: transparent;transform: rotate(342deg);border-bottom-right-radius: 40px;box-shadow: 20px 20px 0 10px #444;}.zone.on .bulb span:nth-child(1){box-shadow: 20px 20px 0 10px #fff;}.bulb span:nth-child(2){position: absolute;top: -16px;right: -4px;display: block;width: 30px;height: 30px;background: transparent;transform: rotate(17deg);border-bottom-left-radius: 40px;box-shadow: -20px 20px 0 10px #444;}.zone.on .bulb span:nth-child(2){box-shadow: -20px 20px 0 10px #fff;}
```

```
.wire{position: absolute;left: calc(50% - 2px);bottom: 50%;width: 4px;height: 60vh;background: #000;z-index: 1;}

.room-bulbs {width:100%; height: 100%; border:2px solid #333;}

.zone{width:33.33333%; float:left; min-height: 200px; height:50%; position:relative;}

.wire{height:50%;}

.percent{position: absolute; bottom:10px; left: calc(50% - 15px); color:red;}
```

4 – 2 – بخش JavaScript

بخش جاوااسکریپت ویجت های تینگزبورد شامل دو تابع اصلی onInit و onDataUpdated است که اولی در زمان بارگذاری اولیه ویجت اجرا می شود و دومی در زمان بروزرسانی داده ها. در این ویجت تابع onInit کارکرد خاصی ندارد لذا به توضیح نحوه پیاده سازی onDataUpdated پرداخته می شود.

به طور خلاصه در تابع onDataUpdated ابتدائاً داده های اول تا ششم موجود در آبجکت self.ctx که مربوط به داده های zone1 الی zone6 است، دریافت شده و به متغیرهای محلی value1,…, value6 انتساب داده می شود:

```
self.onDataUpdated = function() {
   try {
      self.ctx.$scope.value1 =  JSON.parse(self.ctx.data[0].data[0][1]);
...
      self.ctx.$scope.value6 =  JSON.parse(self.ctx.data[5].data[0][1]);
```

پس از اجرای کدهای فوق، در هر قسمت از کدهای html که متغیرهای محلی مذکور وجود دارد، این مقادیر قرار گرفته و شدت نور پس زمینه zoneها و درصد نمایش داده شده تغییر خواهد کرد.

پس از آن با انتخاب نواحی zone مختلف و با توجه به مقدار کلیدهای داده هفتم تا دوازدهم که مربوط به LED ها است کلاس on به zoneها اضافه شده یا حذف می گردد:

```
   let zone1 = document.querySelector('#zone1');
...
   let zone6 = document.querySelector('#zone6');

   if(getZoneBulb(6)) zone1.classList.add('on'); else zone1.classList.remove('on');
...
```

```
    if(getZoneBulb(11)) zone6.classList.add('on'); else zone6.classList.remove('on');
    } catch (error) {
        console.error('Error parsing value', error);
    } finally {
        self.ctx.detectChanges();
    }
};
```

در صورتی که کلاس on به یک ناحیه کلاس zone اضافه شود لامپ داخل آن پس زمینه سفید گرفته و روشن می شود. در تبدیل داده های LED از تابعی به نام getZoneBulb استفاده شده که به صورت زیر پیاده سازی شده است:

```
function getZoneBulb(i) {
    let x = JSON.parse(self.ctx.data[i].data[0][1]);
    if(x == 1) return true;
    else return false;
}
```

فصل 7: رله و برد اصلی

پایش شدت نور و کنترل LED با ESP8266 وThingsBoard :

هدف پروژه:

هدف این پروژه این بود که با استفاده از سنسور LDR، شدت نور محیط رو پایش کنیم و داده‌ها رو به ThingsBoard بفرستیم. در مقابل، از داده‌هایی که از ThingsBoard دریافت می‌کنیم برای کنترل LEDها (که نقش رله رو بازی می‌کنن) استفاده کردیم.

کد:

کتابخانه‌ها و ابزارها:

از ESP8266WiFi برای اتصال به وای‌فای استفاده کردم.

با ESP8266HTTPClient تونستم درخواست‌های HTTP رو ارسال و دریافت کنم.

برای تجزیه و تحلیل JSON از ArduinoJson استفاده شده .

بخش‌های مهم کد:

اتصال به Wi-Fi:

در قسمت connectToWiFi، کد برای اتصال به شبکه وای‌فای تنظیم شده. هر زمان که برد روشن میشه، به وای‌فای متصل میشه و آدرس IP خودش رو نشون میده.

```
void connectToWiFi() {
  Serial.print("Connecting to Wi-Fi");
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(50);
    Serial.print(".");
{
  Serial.println("\nWi-Fi connected!");
  Serial.print("IP Address: ");
  Serial.println(WiFi.localIP());
}
```

خواندن داده از LDR:

از پین A0 برای خوندن مقدار LDR استفاده کردم. داده‌های خام از سنسور رو به درصد تبدیل کردم که بشه راحت‌تر تحلیل کرد. این درصد رو به ThingsBoard می‌فرستم تا اطلاعات مربوط به zoneها رو اونجا داشته باشیم.

```
void processLDRAndLED() {
  int ldrValue = analogRead(PIN_LDR);
  int ldrprecent = map(ldrValue, 0, 1023, 0, 100);
  Serial.println(ldrValue);
  sendToAllLdr(ldrValue, 4);
}
```

دریافت داده از ThingsBoard:

از تابع getAttributesForZones استفاده کردم تا مقادیر نواحی رو از ThingsBoard بگیرم. این مقادیر به صورت JSON دریافت میشن. هر zone مقدار خودش رو داره. اگر مقدار یک zone از حد آستانه بیشتر باشه، LED مربوط به اون روشن میشه.

```
void getAttributesForZones() {
  HTTPClient http;
  http.begin(wifiClient, tbServer);
int httpResponseCode = http.GET();


  if (httpResponseCode > 0) {
    String response = http.getString();
    StaticJsonDocument<1024> jsonDoc;
    if (!deserializeJson(jsonDoc, response)) {
      for (int zoneId = 1; zoneId <= 6; zoneId++) {
        String zoneKey = "Zone" + String(zoneId);
        if (jsonDoc["client"].containsKey(zoneKey)) {
          int zoneValue = jsonDoc["client"][zoneKey];
          if (zoneValue >= zoneThreshold) {
            digitalWrite(ledPins[zoneId - 1], HIGH);
 {          else {
            digitalWrite(ledPins[zoneId - 1], LOW);
}
}
```

```
}
}
{
  http.end();
{
```

کنترل LEDها:

چون تعداد رله کم بود، به جاش از LED استفاده کردیم. هر LED به یک zone اختصاص داده شده. اگر مقدار zone از یه حد خاص بیشتر بشه، LED روشن میشه و اگر کمتر بشه، خاموش میشه.

ارسال وضعیت LEDها:

وضعیت روشن یا خاموش بودن LEDها رو هم به ThingsBoard ارسال کردم تا اطلاعات به‌روز بشه. این کار با تابع sendLedStatus انجام میشه.

تغییرات و چالش‌ها:

استفاده از LED به جای رله:

به دلیل کمبود رله، مجبور شدیم از LED برای شبیه‌سازی استفاده کنیم. این ایده اول ساده به نظر می‌اومد، ولی پیدا کردن پین‌های مناسب و کنترل همزمان همه LEDها یه کم زمان برد.

مشکل در دریافت داده از ThingsBoard:

گاهی اوقات درخواست‌های HTTP با خطا مواجه می‌شدن، برای همین یه تأخیر مناسب اضافه کردم و مطمئن شدم که اتصال Wi-Fi پایدار باشه.

اضافه‌شدن قسمت‌های جدید توسط تیم:

کد اصلی فقط برای کنترل رله نوشته شده بود، ولی بقیه بچه‌ها بخش‌های دیگه‌ای مثل ارسال داده‌های LED و پایش نواحی رو به کد اضافه کردن. همین باعث شد که کد یه کم پیچیده بشه و زمان بیشتری برای هماهنگ‌کردن لازم باشه.

تجزیه JSON:

با استفاده از ArduinoJson تونستم داده‌های JSON رو به راحتی بخونم، ولی تنظیم درست اندازه بافرها (StaticJsonDocument)یکی از چالش‌ها بود.

فصل 8) RPC

در این فصل ما می خواستیم تمام دیوایس ها را از طریق ثینگزبورد خاموش یا روشن کنیم. برعکس تمام تسک های قبلی قرار است داده از طریق ثینگز بورد به دیوایس محلی ارسال کنیم از ESP8466 ESP01 استفاده کردیم. یه رله را به ESP متصل کردیم که بتوانیم که بخش های مختلف را با ان کنترل کنیم. با توجه به دیوایسی که ساخته بودیم از ثینگز بورد یک دستور دریافت می کنیم برای دیوایس که بتواند رله را تغییر وضعیت دهد.

توضیحات کد: در قسمت کدزنی برای این دیوایس از کتابخانه های آردینو و جیسون استفاده کردیم و SUBPUBCLIENT استفاده شد برای اینکه ما میخواستیم عمل بر عکس را روی ثینگز بورد انجام دهیم نیاز داشتیم که یک تابع کال بک را فراخوانی کنیم. ما در تابع کال بک از RPC استفاده کردیم یعنی کنترل رویه از راه دور.

طبق کد هایی که ما زدیم تابع کال بک یک دستور را از طریق RPC از ثینگز بورد دریافت می کند که برایش ویجت سوییچ گذاشته بودیم. این داده ها به یک تابع بولینگ تبدیل می شود و از طریق تابع بولینگ و سینتکس پارامتر وضعیت رله را میتواند HIGH یا LOW کند. برای اتصال دیوایس به ثینگز بورد از طریق پروتکل MQTT اقدام کردیم و توکن از ثینگز بورد دریافت کردیم و دیوایس ثبت کردیم. دیوایس 22 داخل داشبورد به همین دیوایس اشاره دارد. از طرفی دیگر ما چالش های زیادی داشتیم یکی از آنها این بود که ما نمیتوانستیم که ویجت را مقدار دهی کنیم. یعنی نمیتوانستیم برای آن قسمت از ویجت که نیاز به پی لود دارد مقدار دهی کنیم و از ویجت اماده استفاده کردیم. بعدا متوجه شدیم که ویجت اماده ای که استفاده کردیم دقیقا چه قسمت هایی را تغییر می دهد که کارایی درستی داشته باشد. ما ویجت را علارغم اینکه چند ویجت را خودمان ساختیم از ویجت اماده از پلتفرم گیت هاب و تحت لایسنس اوپن سورس استفاده کردیم. یکی دیگر از چالش ها این بود که ما اول تلاش کردیم کار را از طریق پابلیش سابسکرایب بدون جیسون انجام دهیم یعنی سعی کردیم حتی از طریق دیوایس آی دی از طریق جی اس برای ویجت یک ارتباطی برقرار کنیم که به بن بست برخوردیم و از طریق جیسون اقدام کردیم.

شما میتوانید از طریق یک پاور کنترل که یک ویجت داخل ثینگزبورد است وضعیت رله را از راه دور تغییر دهید. اول میخواستیم با استفاده یک سری دیتا خود ثینگز بورد تصمیم بگیرد کی دیوایس ها را فعال و غیر فعال کند اما این تسک برای گروه ما سنگین بود و تلاش های زیادی انجام شد.

در این تسکی که انجام دادیم ثینگزبورد پابلیشر و دیوایس سابسکرایبر بود. نقطه خلاقیت ما در این دیوایس این بود که بتوانیم یک رویه جدید را تست و بررسی کنیم. بعد ها میخواهیم این خلاقیت ها را پیاده سازی کنیم.

تابع وای فای کانکت هم مانند سایر تابع ها برای بررسی و اتصال مجدد اتصال بود و کانکشن MQTT هم بررسی می کردیم.

فصل 9) نتیجه گیری و چالش ها:

به صورت کلی این مجموعه پروژه ها آزمایشگاه IoTCPP را هوشمند می کند. البته هنوز چالش هایی مثل امنیت وجود دارد که در آینده حل خواهیم کرد.

چالش اصلی ما عدم پیروی از یک الگوی خاص مثل عجایل و عدم وجود اسکرام مستر یا لیدر تیم بود که باعث بروز بی نظمی هایی شد که در آینده و پروژه های بعدی این موارد را رعایت خواهیم کرد.

پایان