**باسمه تعالی**


# امنیت در اینترنت اشیاء

## دکتر احمدی

## تمرین چهارم

## پیاده سازی HTTP

**پوریا دادخواه**

**401201381**

## 1. راه‌اندازی اولیه

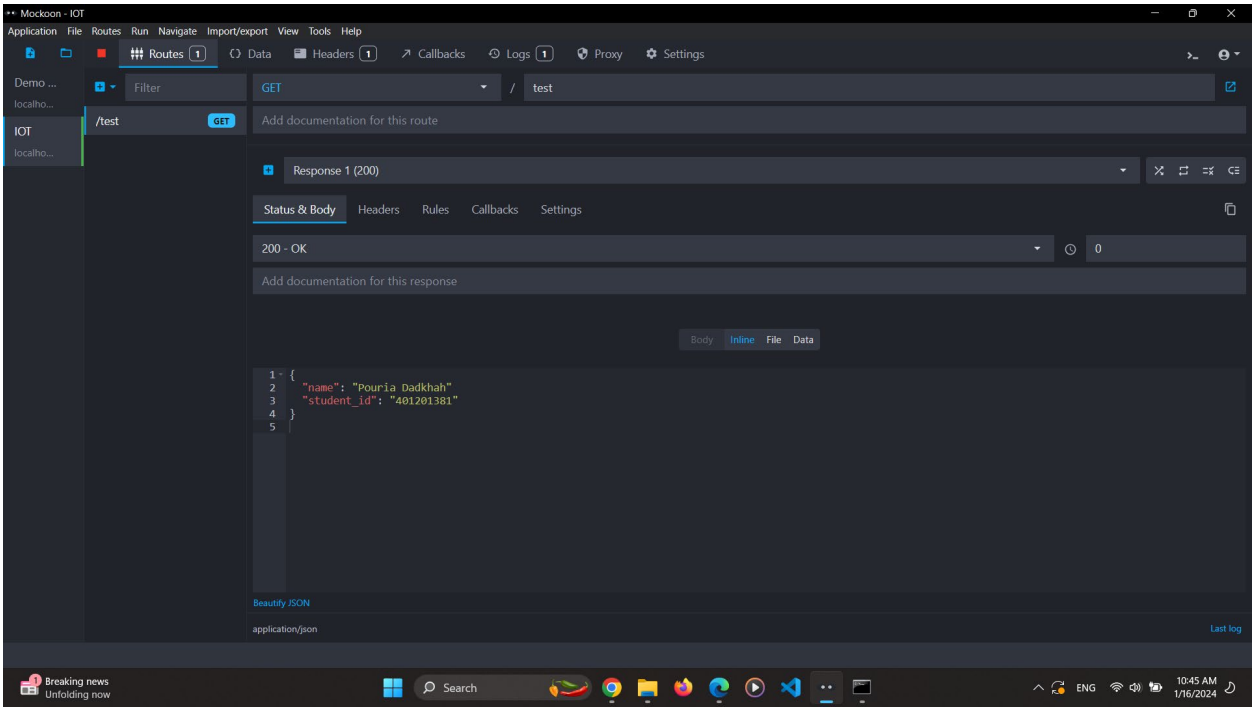
برای راه‌اندازی سرور در ویندوز از نرم‌افزار mockoon استفاده کردیم. کار با این نرم‌افزار پیچیدگی خاصی ندارد و صرفا کافی است پس از ساختن environment کار خود ( که ما با نام IOT ساختیم )، یک Http route جدید به آن اضافه کنیم و متد آن را مطابق خواسته سوال GET قرار داده و path گفته شده (test) را تعیین می‌کنیم و پاسخ json را مدنظر را در body پاسخ درخواست‌های ارسالی از کلاینت‌ها بنویسیم. در ضمن status code آن را نیز ok 200 تعریف می‌کنیم که در ادامه مفهوم این وضعیت پاسخ را شرح می‌دهیم.

برای تنظیم IP , Port سرور نیز از بخش تنظیمات، پورت را روی 80 تنظیم می‌کنیم.( نیازی به تغییر IP نیست و پیش‌فرض روی localhost ست شده‌است.

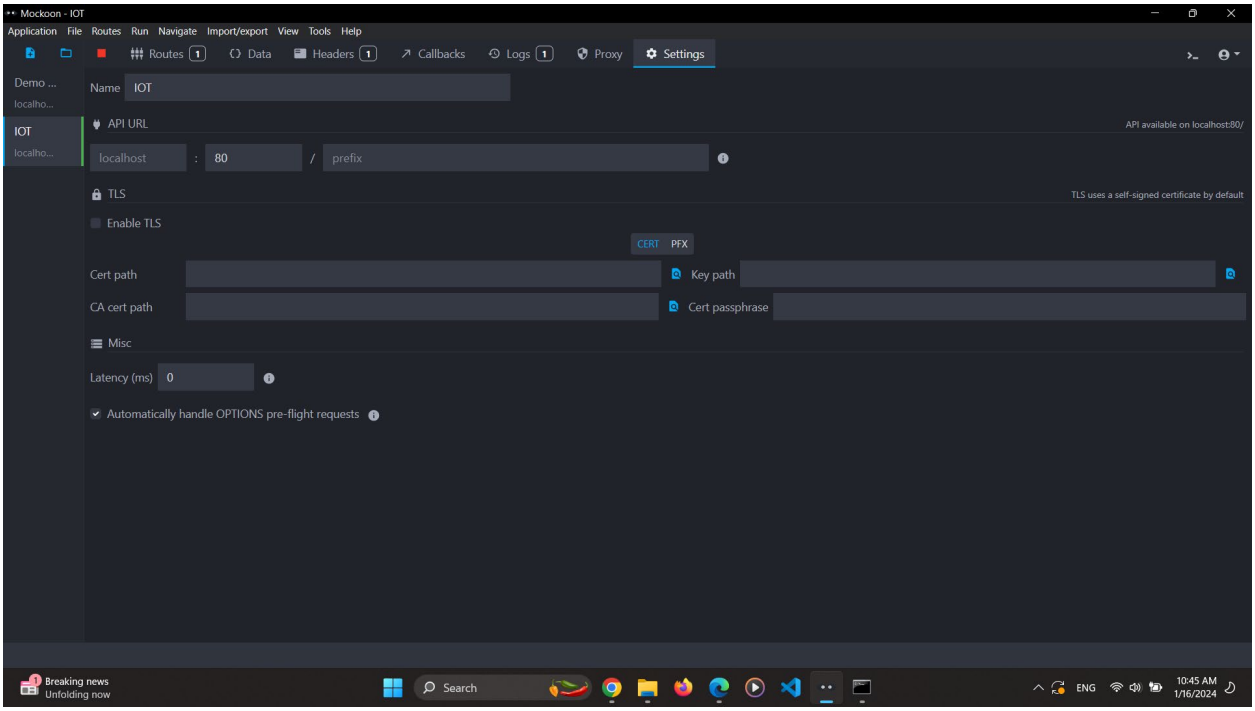
در نهایت پس از start این سرور، یک سرور خواهیم داشت که اگر به url زیر از طرف کلاینت‌های همین ماشین به آن درخواستی ارسال شود، پاسخ json نوشته شده را به عنوان response خواهد داد.

http://localhost:80/test

در ادامه ابتدا تنظیمات محیط mockoon و سپس پاسخ دریافتی توسط کلاینت cmd که از curl درخواست زده‌ایم را می‌بینیم:




تعریف route سرور




تنظیم ip:port سرور

```
C:\Users\User>curl http://localhost:80/test
{
  "name": "Pouria Dadkhah"
  "student_id": "401201381"
}
```

تست سرور توسط کلاینت محلی

- مفهوم status code 200 OK:

این وضعیت در زمینه پاسخهای HTTP نشان دهنده یک درخواست موفق است. این یکی از کدهای وضعیت استاندارد HTTP است و به طور خاص نشان میدهد که سرور با موفقیت درخواست را پردازش کرده و داده های مورد انتظار را برمی‌گرداند.

## 2. راه‌اندازی ESP32

### 2.1 ESP32 as Client

در این قسمت، ساختار اصلی کد نمونه esp_http_client_example از مثال‌های Espressif را انتخاب کردیم و در راستای هدف خود شخصی‌سازی کردیم.

در این قسمت به دو متد اصلی برای هندل کردن رویداد درخواست‌ها و ایجاد ارتباط http نیاز داریم.

اولین متد برای هندل رویدادها، _http_event_handler() است که این تابع را ساده کرده و فقط کافی است درصورت موفق بودن یک درخواست، محتوای پاسخ را نشان دهد. ( سایر رویدادهای اضافی و واکنش‌های غیردرخواستی را مورد تحلیل قرار نمی‌دهیم) ساختار نهایی این تابع در ادامه آورده‌شده است:

```
esp_err_t _http_event_handler(esp_http_client_event_t *evt)
{
    switch (evt->event_id)
    {
    case HTTP_EVENT_ON_DATA:
        printf("HTTP_EVENT_ON_DATA: %.*s\n", evt->data_len, (char *)evt->data);
        break;

    default:
        break;
    }
    return ESP_OK;
}
```

متد دومی که باید درخواست http را ایجاد کند، http_rest_with_url است. در این متد ابتدا مشخصات سرور را درکانفیگ http client وارد کرده ( ip, port، cert اگر نیازی به امن‌سازی باشد و نوع متد ارسالی) و سپس، نمونه‌ای از آن کلاینت می‌سازیم و اجرا می‌کنیم:

```
static void http_rest_with_url(void){
    esp_http_client_config_t config_get = {
        .url = "http://172.20.27.74:80/test",
        .method = HTTP_METHOD_GET,
        .cert_pem = NULL,
        .event_handler = _http_event_handler
    };
    // GET
    esp_http_client_handle_t client = esp_http_client_init(&config_get);
    esp_http_client_perform(client);
    esp_http_client_cleanup(client);
```

```
}
```

در آخر کافی‌اســـت در main پس از اجرای LOG های راه‌اندازی esp، آن را به wifi متصـــل کرده و متد http_rest_with_url() را فراخوانی می‌کنیم.

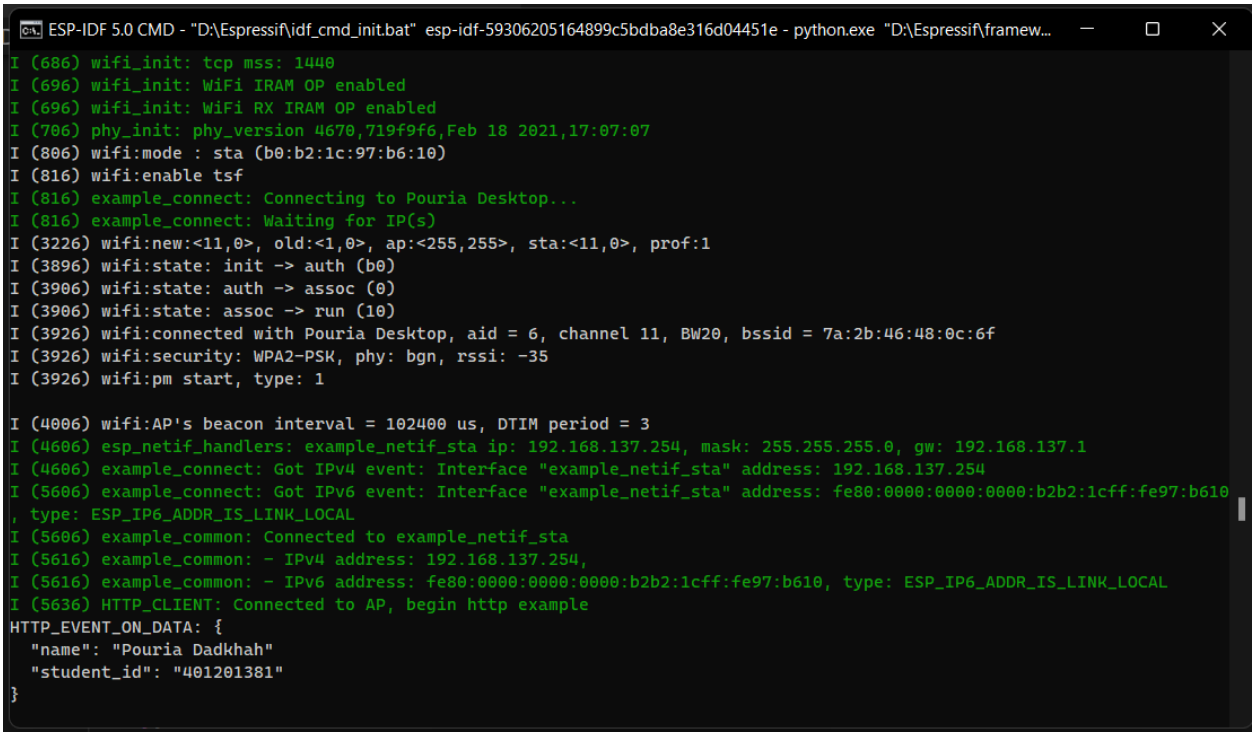
در این تمرین هم مشابه تمرین قبل به دو نکته باید توجه کنیم:

- برای اتصـــال به وای‌فای، از example_connect() خود esp-idf اســـتفاده می‌کنیم که از طریق اطلاعات وارد شده در menuconfig، ssid , password وای‌فای را می‌گیرد و متصل می‌شود.
- همانند تمرین قبل شرایط مناسب اتصال به wifi مشترک بین سیستم و esp را نداشتیم و نت دانشگاه قابل تنظیم روی esp نبوده و mobile hotspot گوشـــی هم کلاینت‌های خود را در zoneهای جداگانه قرار می‌دهد. بنابراین esp را به hotspot لپ‌تاپ متصل می‌کنیم و از سیستم خود به عنوان access point استفاده می‌کنیم.

متد miainنهایی را نیز می‌توانیم در ادامه ببینیم:

```
void app_main(void)
{
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND)
{
      ESP_ERROR_CHECK(nvs_flash_erase());
      ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);
    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());

    /* This helper function configures Wi-Fi or Ethernet, as selected in
menuconfig.
     * Read "Establishing Wi-Fi or Ethernet Connection" section in
     * examples/protocols/README.md for more information about this function.
     */
    ESP_ERROR_CHECK(example_connect());
    ESP_LOGI(TAG, "Connected to AP, begin http example");

    http_rest_with_url();
}
```

نتیجه تســت این کلاینت و درخواســت آن به mock server ای که در بخش قبل راه‌اندازی کردیم را در شکل بعدی آورده‌ایم. ( در ضمن جهت اطمینان، ip سرور را در mockoon برابر ip wifi سیستم که در تنظیمات esp وارد کردیم تنظیم کردیم.




درخواست از سمت کلاینت esp32 به سرور و دریافت پاسخ

## ESP as Server 2.2

در این قسمت esp را به عنوان یک access point راه‌اندازی کرده و یک وبسرور را روی آن اجرا کنیم که توانایی هندل کردن درخواست‌های ارسالی مشخصی را داشته باشد. برای این منظور از دو ساختار پایه نمونه espressif استفاده می‌کنیم؛ از نمونه softAp برای تنظیم esp به عنوان ap و در ادامه از http_esp_server برای راه‌اندازی وب‌سرور. در ادامه به توضیح جزییات هر دو بخش می‌پردازیم:

- ESP Access Point

برای این منظور به دو متد wifi_event_handler() و wifi_init_softap() نیاز داریم.

از wifi_event_handler() برای هندل کردن درخواست اتصال و قطع کلاینت‌ها به wifi مانند اختصاص ip به آن‌ها – که در ساده‌ترین روش از DHCP خود تابع کتابخانه‌ای استفاده می‌کنیم – استفاده می‌کنیم.

البته برای اینکار نیاز به پیاده‌سازی دستی موارد جزیی نداریم و تنها کافیست دو شی از event های آماده wifi_event_ap_staconnected_t و wifi_event_ap_stadisconnected_t بسازیم.

از متد wifi_init_softap هم برای ساخت نقطه دسترسی روی esp با کانفیگی که برای آن تعریف می‌کنیم و پاس دادن هندلر تعریف شده به آن استفاده می‌کنیم. در این قسمت ssid و wifi password خود را تنظیم می‌کنیم و سایر ملاحظات امنیتی را برابر مقادیر پیش‌فرض نمونه قرار می‌دهیم ( مثلا طول پسورد حداقل باید 8 حرف باشد وگرنه نقطه دسترسی ایجاد نمی‌شود)

در ادامه کد تکمیل شده راه‌اندازی ap را مشاهده می‌کنیم:

```
static void wifi_event_handler(void* arg, esp_event_base_t event_base, int32_t
event_id, void* event_data)
{
    if (event_id == WIFI_EVENT_AP_STACONNECTED) {
        wifi_event_ap_staconnected_t* event = (wifi_event_ap_staconnected_t*)
event_data;
        ESP_LOGI(TAG, "station "MACSTR" join, AID=%d",
                 MAC2STR(event->mac), event->aid);
    } else if (event_id == WIFI_EVENT_AP_STADISCONNECTED) {
        wifi_event_ap_stadisconnected_t* event =
(wifi_event_ap_stadisconnected_t*) event_data;
        ESP_LOGI(TAG, "station "MACSTR" leave, AID=%d",
                 MAC2STR(event->mac), event->aid);
    }
}

esp_err_t wifi_init_softap(void)
{
    esp_netif_create_default_wifi_ap();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT, ESP_EVENT_ANY_ID,
&wifi_event_handler, NULL));

    wifi_config_t wifi_config = {
        .ap = {
            .ssid = EXAMPLE_ESP_WIFI_SSID,
            .ssid_len = strlen(EXAMPLE_ESP_WIFI_SSID),
            .password = EXAMPLE_ESP_WIFI_PASS,
            .max_connection = EXAMPLE_MAX_STA_CONN,
```

```
            .authmode = WIFI_AUTH_WPA_WPA2_PSK
        },
    };
    if (strlen(EXAMPLE_ESP_WIFI_PASS) == 0) {
        wifi_config.ap.authmode = WIFI_AUTH_OPEN;
    }

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_AP));
    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_AP, &wifi_config));
    ESP_ERROR_CHECK(esp_wifi_start());

    ESP_LOGI(TAG, "wifi_init_softap finished. SSID:%s password:%s",
             EXAMPLE_ESP_WIFI_SSID, EXAMPLE_ESP_WIFI_PASS);
    return ESP_OK;
}
```

- Http server

برای این قسمت نیز به دو متد اصلی start_webserver و post_handler_ نیاز داریم. در تابع اول با ست کردن کانفیگ‌های دلخواه، سرور را راه اندازی کرده و متدهایی که می‌خواهیم در path های مختلف ساپورت کنیم را به کانفیگ ایجاد شده مربوطه ارجاع می‌دهیم.

در متد دوم هم که در کد ما در path echo ست شده است، وظیفه هندل کردن درخواست‌های post را نوشته و پارس کردن مناسب body درخواست دریافتی، درصورت معتبر بودن پیام ( valid بودن، LED را روشن یا خاموش می‌کنیم. ) برای پارس کردن json درخواس از کتابخانه cJson استفاده کرده و کد مربوط به کنترل led نیز مشابه تمرین با قبل با ست کردن تنظیمات GPIO NUM2 و تعیین level آن صورت می‌گیرد که از توضیح مجدد آن خودداری می‌کنیم.

توابع نهایی این دو متد به صورت زیر هستند که کامنت گزاری مناسب نیز برای هر بخش صورت گرفته است:

```
static esp_err_t echo_post_handler(httpd_req_t *req)
{
    char buf[100];
    int ret, remaining = req->content_len;

    while (remaining > 0) {
        /* Read the data for the request */
        if ((ret = httpd_req_recv(req, buf, MIN(remaining, sizeof(buf)))) <= 0) {
            if (ret == HTTPD_SOCK_ERR_TIMEOUT) {
                /* Retry receiving if timeout occurred */
```

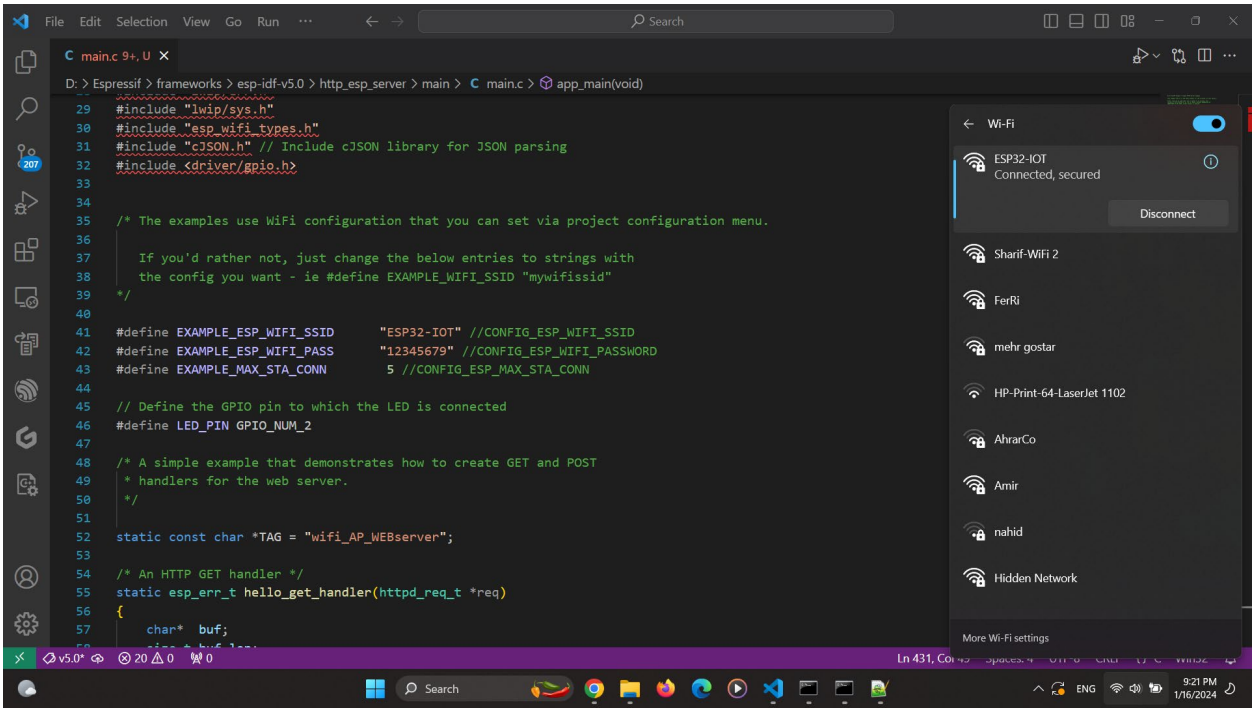
```
            continue;
        }
        return ESP_FAIL;
    }

    /* Log data received */
    ESP_LOGI(TAG, "=========== RECEIVED DATA ==========");
    ESP_LOGI(TAG, "%.*s", ret, buf);
    ESP_LOGI(TAG, "====================================");

    // Print the JSON data for further inspection
    printf("Received JSON data: %.*s\n", ret, buf);

    // Parse JSON data
    cJSON *root = cJSON_Parse(buf);
    if (root == NULL) {
        ESP_LOGE(TAG, "Error parsing JSON data");
        return ESP_FAIL;
    }

    // Get device and command from JSON
    cJSON *device = cJSON_GetObjectItem(root, "device");
    cJSON *command = cJSON_GetObjectItem(root, "command");

    if (device != NULL && command != NULL) {
        // Check if the device is LED and the command is either on or off
        if (strcmp(device->valuestring, "LED") == 0) {
            if (strcmp(command->valuestring, "on") == 0) {
                // Turn on the LED
                gpio_set_level(LED_PIN, 1);
                ESP_LOGI(TAG, "LED turned ON");
            } else if (strcmp(command->valuestring, "off") == 0) {
                // Turn off the LED
                gpio_set_level(LED_PIN, 0);
                ESP_LOGI(TAG, "LED turned OFF");
            } else {
                ESP_LOGE(TAG, "Invalid command: %s", command->valuestring);
            }
        } else {
            ESP_LOGE(TAG, "Invalid device: %s", device->valuestring);
        }
    } else {
        ESP_LOGE(TAG, "Missing device or command in JSON");
    }
```

```
        // Free cJSON objects
        cJSON_Delete(root);

        remaining -= ret;
    }

    // Send response
    const char *response = "{\"message\": \"LED operation completed\"}";
    httpd_resp_send(req, response, strlen(response));

    return ESP_OK;
}

static const httpd_uri_t echo = {
    .uri       = "/echo",
    .method    = HTTP_POST,
    .handler   = echo_post_handler,
    .user_ctx  = NULL
};
```

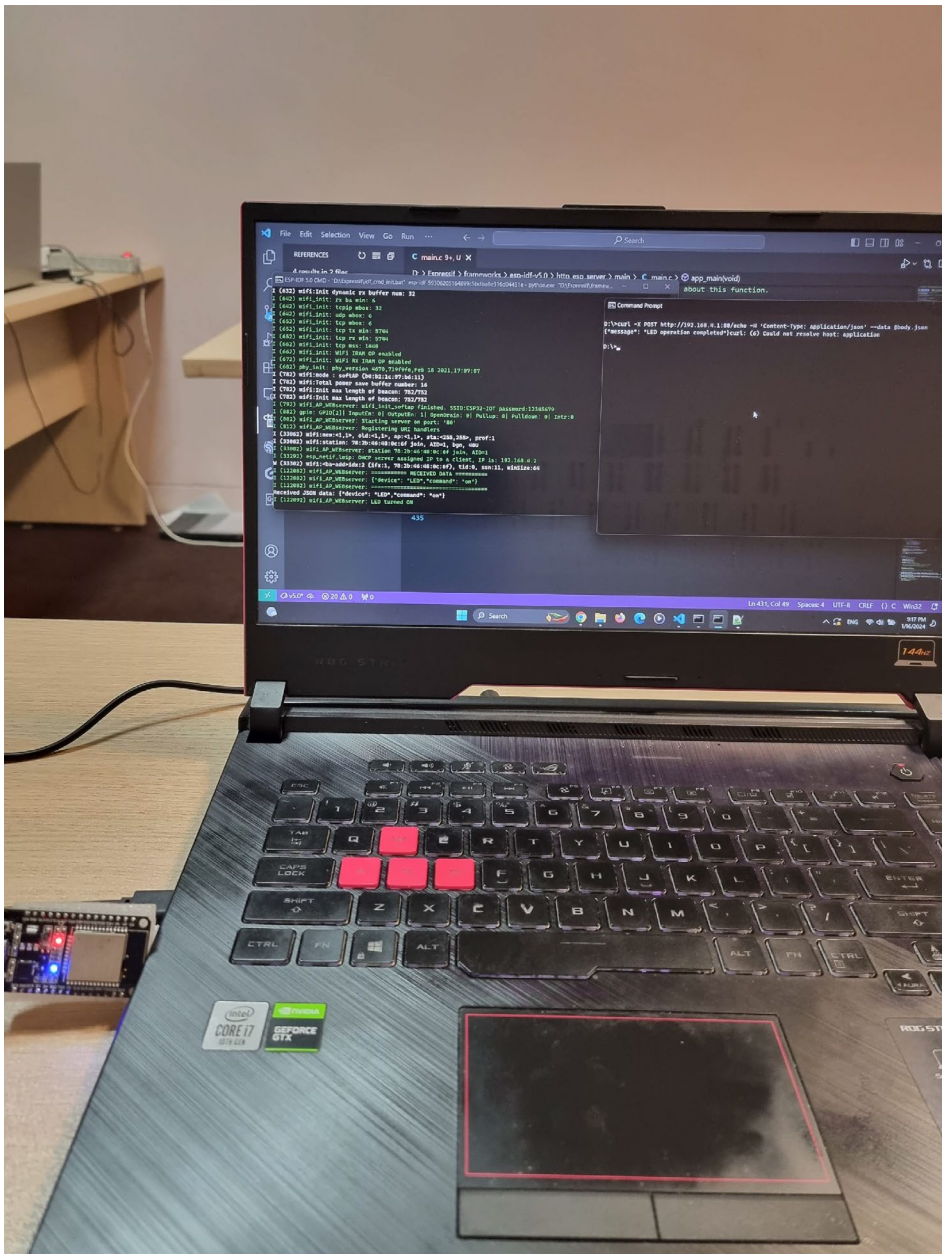
```
static httpd_handle_t start_webserver(void)
{
    httpd_handle_t server = NULL;
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.lru_purge_enable = true;

    // Start the httpd server
    ESP_LOGI(TAG, "Starting server on port: '%d'", config.server_port);
    if (httpd_start(&server, &config) == ESP_OK) {
        // Set URI handlers
        ESP_LOGI(TAG, "Registering URI handlers");
        httpd_register_uri_handler(server, &hello);
        httpd_register_uri_handler(server, &echo);
        httpd_register_uri_handler(server, &ctrl);
        #if CONFIG_EXAMPLE_BASIC_AUTH
        httpd_register_basic_auth(server);
        #endif
        return server;
    }

    ESP_LOGI(TAG, "Error starting server!");
    return NULL;
}
```

هم‌چنین در کد موجود، توابع اضافه برای هندل کردن درخواست‌های get و put نیز در pathهای دیگر نوشته شده‌است که صرفا آن‌ها را حذف نکردیم و استفاده‌ای نیز از ایشان نمی‌شود.

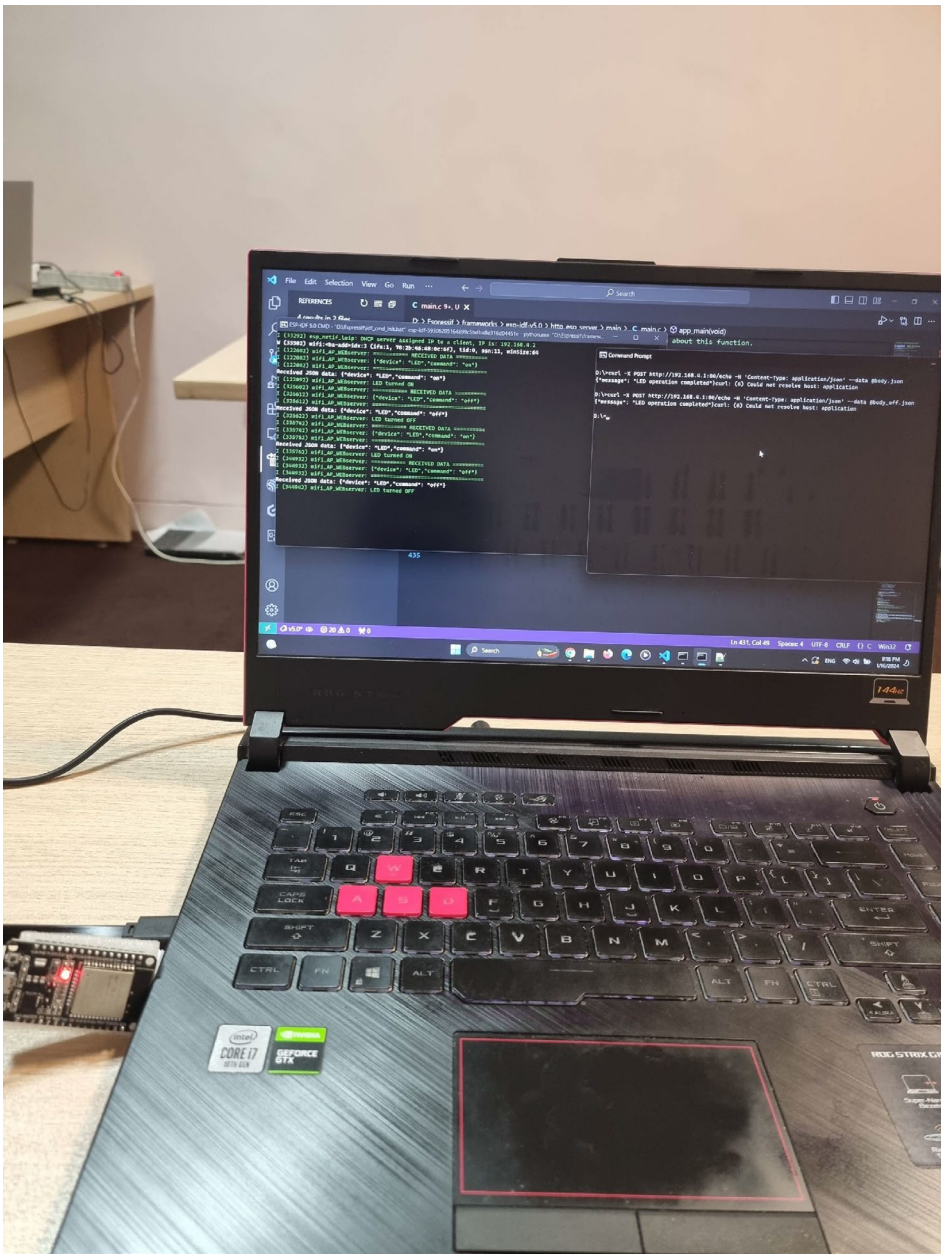
پس از اجرای کد فوق، ابتدا سیستم را به ESP32_IOT hotspot متصل کرده ( مطابق شکل اول ) و سپس درخواست post روشن و خاموش کردن را ارسال می‌کنیم که نتایج و اسکرین‌شات‌های مربوطه به ترتیب آورده‌شده‌اند:



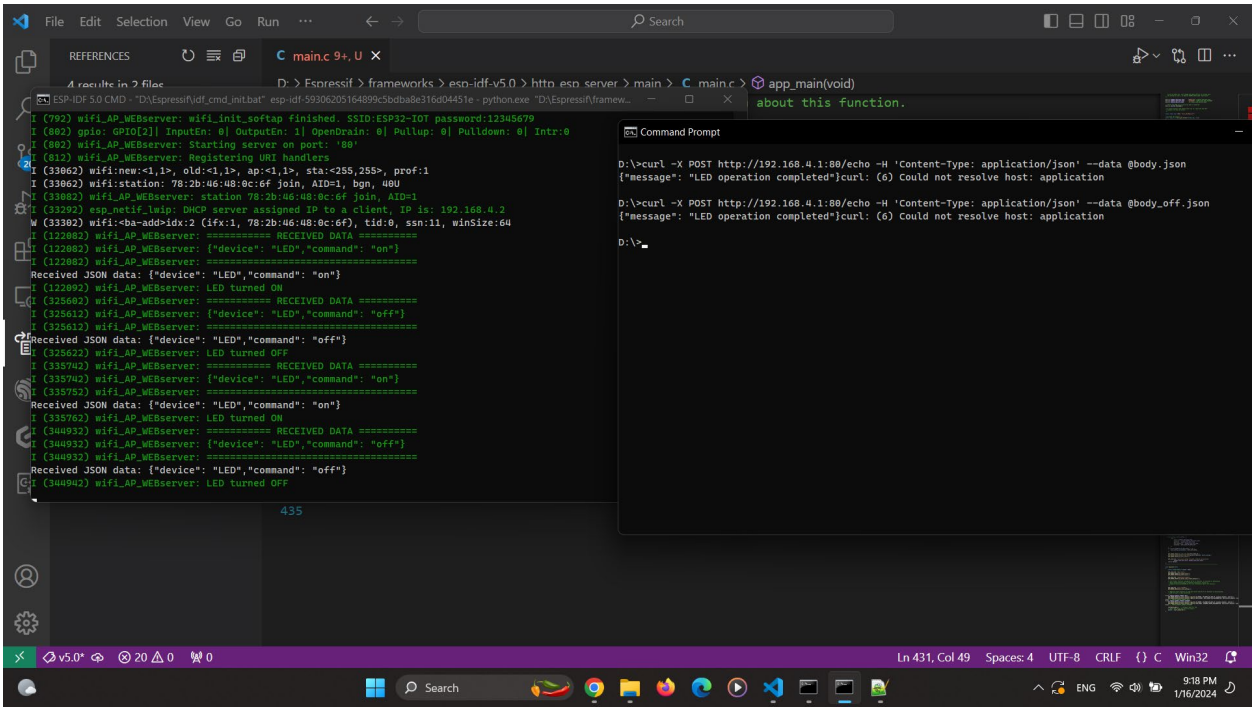

اتصال به نقطه دسترسی esp



روشن کردن LED



خاموش کردن LED




درخواست‌های ارسالی و log های esp در پاسخ

همان‌گونه که در تصویر هم مشاهده می‌شود، درخواست post به سرورesp را از طریق curl کامندلاین زدیم که body را نیز از فایل json از پیش ساخته شده body.json پاس می‌دهیم.

curl -X POST http://192.168.4.1:80/echo -H 'Content-Type: application/json' --data @body.json

- کدها و فایل‌های اشاره شده در هر بخش در فایل تمرین ضمیمه شده است.