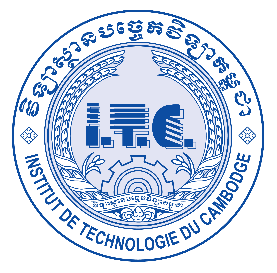
****

Department of Electrical and

Energy Engineering

Institute of Technology of Cambodia

**Student Project**

**Topic : IOT Controlling and Monitoring System**

**Group: I5-GEE-EA-Gruop2**

**Lecturer by :** SUM Rithea

**Student :** PUN Phorn (e20200408)

Academic Year5

2024-2025

Contents

[1. INTRODUCTION 2](#_Toc190097597)

[1.1 Project Statement 2](#_Toc190097598)

[1.2 Objective of the Study 2](#_Toc190097599)

[1.3 Scope of Work 2](#_Toc190097600)

[2. DESIGN METHODOLOGY 3](#_Toc190097601)

[2.1 Hardware Design 3](#_Toc190097602)

[2.1.1 Optocoupler Circuit 4](#_Toc190097603)

[2.1.2 Transistor and Relay Circuit 5](#_Toc190097604)

[2.2 Software Development 7](#_Toc190097605)

[2.2.1 FreeRTOS Task with HTTPS Communication Implementation 8](#_Toc190097606)

[2.2.2 Firebase Integration 9](#_Toc190097607)

[2.2.3 Web Application Development 11](#_Toc190097608)

[2.2.4 Firebase Hosting 12](#_Toc190097609)

[3. RESULT AND DISCUSSION 13](#_Toc190097610)

[3.1 PCB Result and Testing 14](#_Toc190097611)

[3.2 Firebase and ESP32 client testing for a web app. 15](#_Toc190097612)

[4. CONCLUSION 16](#_Toc190097613)

**STUDENT PROJECT**

# INTRODUCTION

## **Project Statement**

In recent years, the Internet of Things (IoT) has emerged as a transformative technology with applications spanning smart homes, industrial automation, healthcare, and environmental monitoring. IoT systems enable devices to collect, process, and exchange data in real time, offering unprecedented levels of connectivity and control. One prominent application of IoT is in smart lighting systems, which enhance energy efficiency, convenience, and user experience by dynamically adjusting lighting based on environmental conditions or user preferences.

This project involves designing a custom ESP32 development board integrated with relays for smart home functions and monitoring data that capture by sensors. The board is programmed using the ESP-IDF framework, enabling it to communicate with Firebase for data storage, updates, and retrieval. A web application is developed to control the ESP32 and view real-time data.

## **Objective of the Study**

The main objectives of this study are:

* To design and manufacture a custom PCB (Printed Circuit Board) for the ESP32 with relays integration.
* To write the program using ESP32 -IDF framework to read data that is captured from sensors, send it to firebase, and retrieve data from firebase to control relays.
* To create a user-friendly web application for monitoring and controlling devices connected to the ESP32.

## **Scope of Work**

The project covers the following aspects:

* Writing a program in the ESP-IDF framework to interact with Firebase for storing sensors data and controlling relays.
* Developing a website to interface with Firebase for remote control and data visualization.
* Hosting Web app with Firebase.

# DESIGN METHODOLOGY

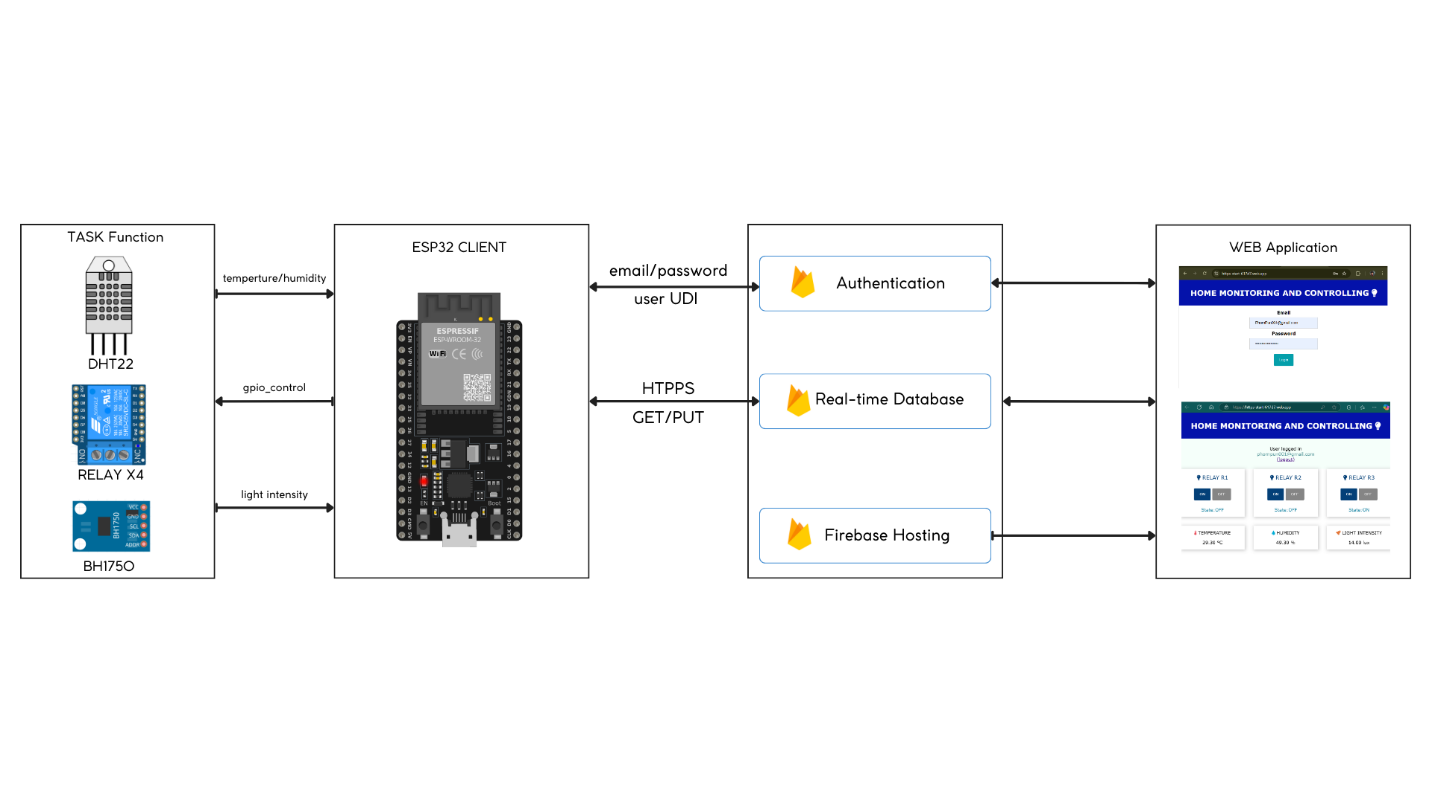
This section outlines the methodology used to design and implement the IoT-based home monitoring and control System. The process was divided into three main phases: Hardware Design, Software Development, and Testing and Validation. Each phase is described in detail below. 

Figure 1: System Overview

The Figure 1 demonstrates the system overview. The proposed system includes an ESP32 and a sensor functioning as a client, while Firebase serves as the server, providing authentication, a real-time database for data storage, and Firebase Hosting for web application hosting.

## **Hardware Design**

The hardware design focused on integrating all components into a compact and reliable system. Key considerations included component selection, PCB layout, and electrical calculations to ensure proper operation.

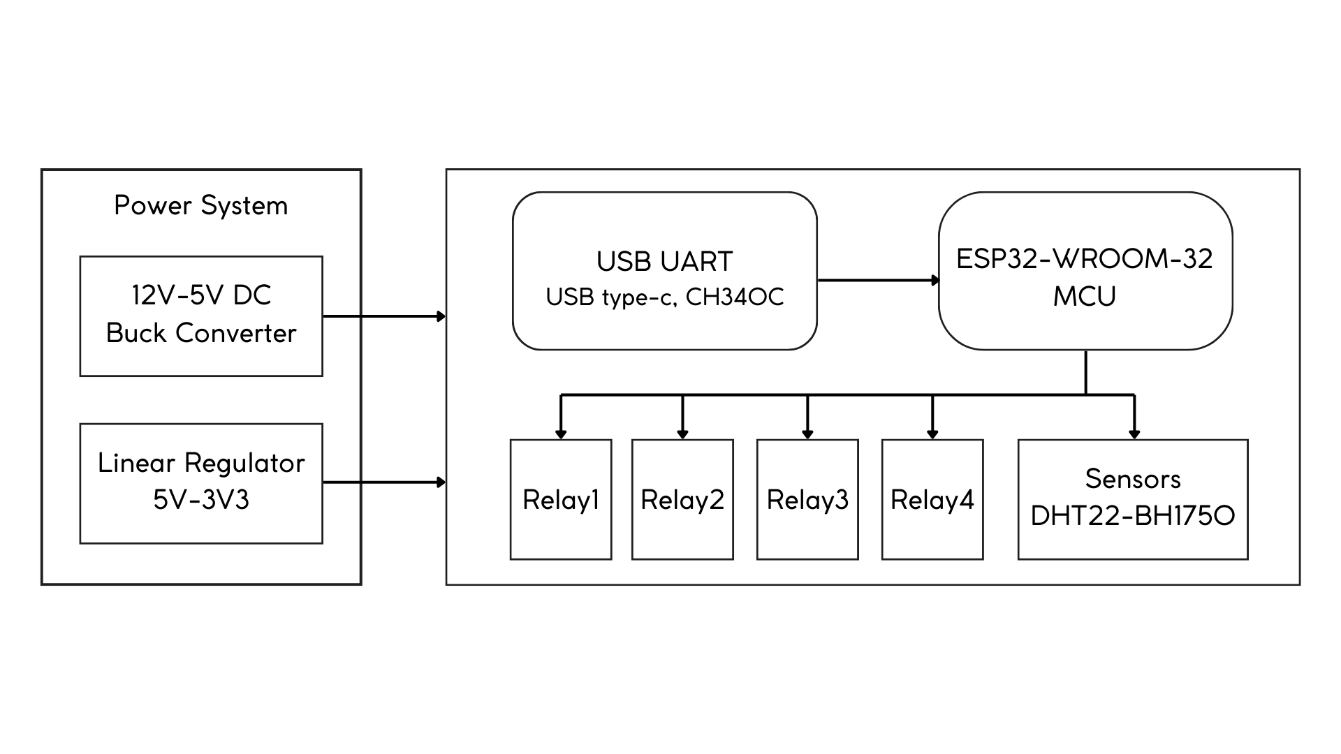


Figure 2: Block diagram of PCB design

Figure 2 illustrates the block diagram of the PCH design. The block includes the power system, which consists of a DC buck converter from 12V to 5V and a linear regulator from 5V to 3.3V. This power is supplied to the entire board. The right block represents the diagram for the USB UART connected to the ESP32, and four relays and sensors are controlled by the ESP32.

### **Optocoupler Circuit**

The PC817C optocoupler was used to provide electrical isolation between the ESP32 microcontroller and the relay driver circuit. This ensures that high-voltage spikes from the relay coil do not damage the sensitive GPIO pins of the ESP32.

* Circuit Design

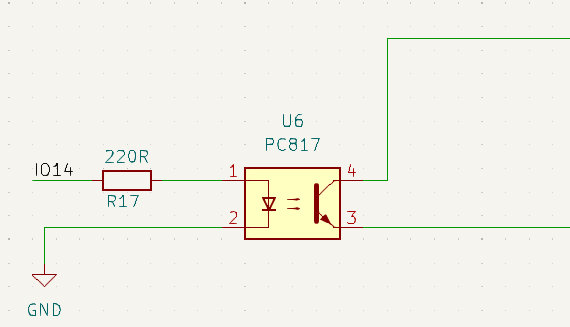


Figure 3: optocoupler circuit

The optocoupler consists of an internal LED and a phototransistor. The LED is driven by the ESP32's GPIO pin through a current-limiting resistor (R).

* LED Control Circuits

Figure 4(a) shows an example of controlling LED drive current by switching the supply voltage VIN on and off. Figure 4(b) indicates a load line in the (a) circuit. In this case, the resistor R is as follows.

|  |  |  |
| --- | --- | --- |
|  | ( a ) | ( b) |

Figure 4: optocoupler DC drive

The forward current has chosen , (ESP32 GPIO), and (PC817 specification).

Therefore, the resistor should be selected as .

* Simulation:

The optocoupler circuit was simulated in LTSPICE software to verify its operation:

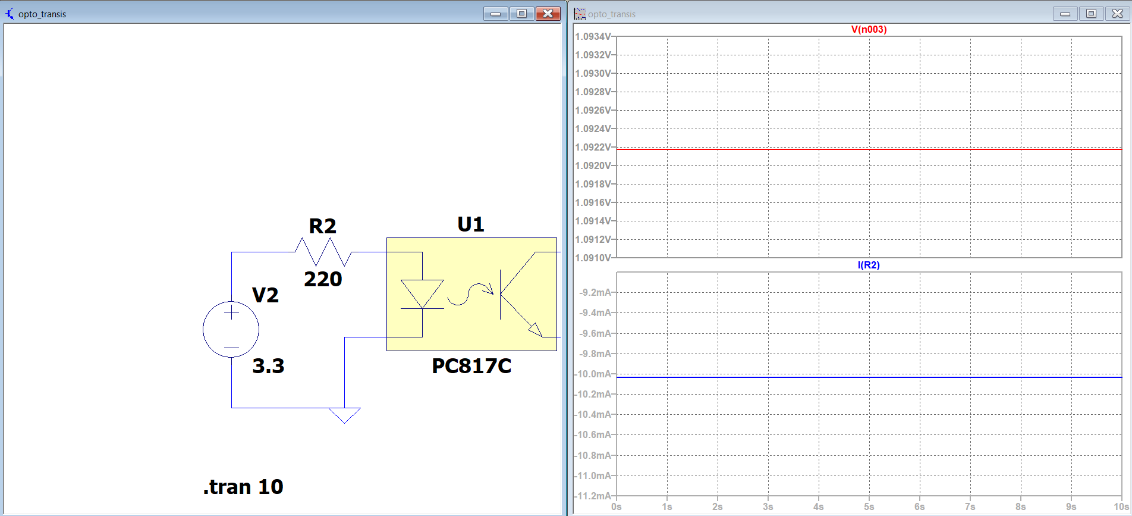


Figure 5: Simulation of PC817C circuit

Figure 5 displays the LTSPICE simulation for the PC817c with resistor R2, revealing that the forward voltage (VF) exceeds 1.0922V and the forward current (IF) is above 10mA. The results indicate a slight discrepancy between the simulation and calculations.

### **Transistor and Relay Circuit**

The **2N3904 NPN transistor** was used to drive the relay coil, ensuring sufficient current to activate the relay while protecting the optocoupler.

* Circuit Design

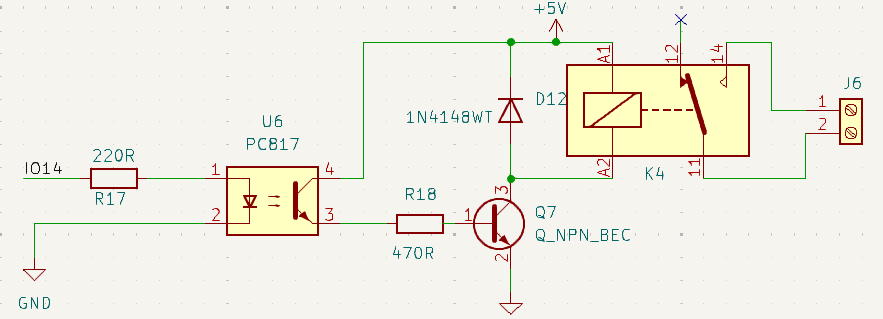


Figure 6: Circuit control relay completed

The transistor acts as a switch, controlled by the output of the optocoupler. A base resistor (**R17**) limits the current flowing into the transistor's base. The relay coil is connected to the collector, with the emitter grounded.

* Resistor Calculation

In the saturation region, the phototransistor's collector current becomes largely independent of the LED forward current , provided that is sufficient to drive the phototransistor into saturation.

**Saturation Behavior**

In **saturation mode,** the phototransistor operates in a region where: the collector-emitter voltage is very low, typically less than 0.2–0.3V. At this point, the collector current is primarily determined by the external load connected to the phototransistor, not by the base current .

First define the collector current:

|  |  |
| --- | --- |
| since, |  |

Following by 2N3904 datasheet

Therefore, the resistor should be selected as .

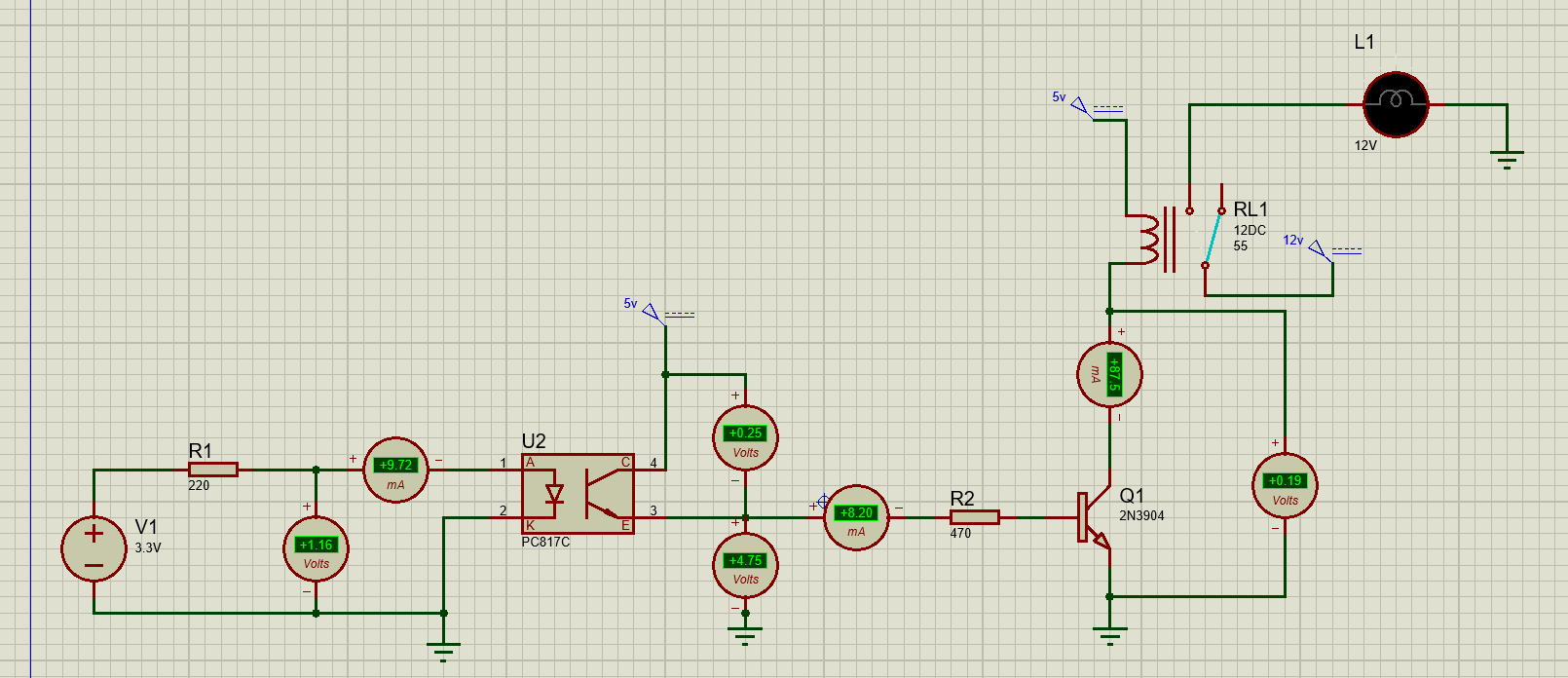


Figure 7: The rest of relay circuit simulation

After calculating the collector current, base current, and resistor values for the 2N3904 used with the relay, the Proteus simulator was employed to simulate the circuit. The simulation results indicate a slight discrepancy between theory and practice.

## **Software Development**

The software development phase was a critical component of the project, involving programming the ESP32microcontroller using the ESP-IDF platform**,** integrating it with Firebase Realtime Database for secure data storage and retrieval, and developing a web applicationfor user interaction. The web application was hosted on Firebase to ensure seamless real-time communication between the hardware and the user interface.

### **FreeRTOS Task with HTTPS Communication Implementation**

The ESP-IDF platformuses FreeRTOSto handle multitasking, ensuring efficient operation of

sensor readings, data transmission, and relay control. Each task was designed to run independently while communicating securely with Firebase via HTTPS using the esp\_http\_client library.

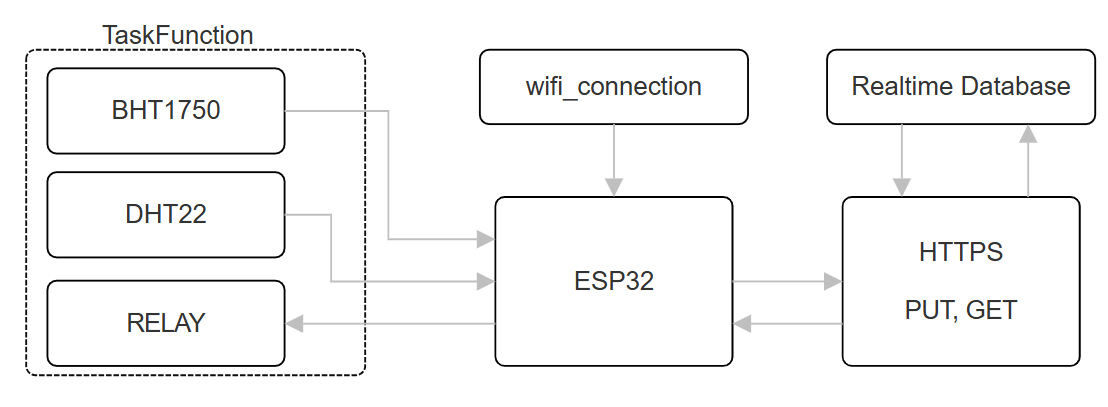


Figure 8: Structure of FreeRTOS and HTTPS Implement in ESP-IDF

* Task 1: DHT22 Sensor Reading and Data Transmission (dht\_task)

Reads temperature and humidity data from the **DHT22 sensor** every 1 seconds and sends it to Real-time data through URL.

* Task 2: BH1750 Sensor Reading and Data Transmission (bh1750\_task)

Read light intensity data from the BH1750 sensor via I2C every second, and send the data as a JSON payload to Firebase.

* + Task 3: Button Control and Relay State Update (button\_task)

Retrieve button states from Firebase via an HTTP GET request, parse the JSON response, and update the GPIO pins controlling the relays accordingly.

* Secure HTTPS Communication Between ESP32 and Firebase Using TLS Encryption.

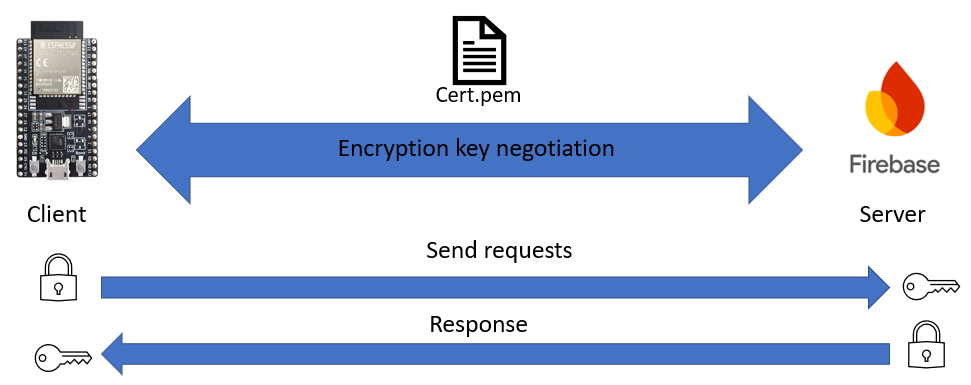


Figure 9: ESP32 with Firebase data encryption

This diagram shows secure communication between an ESP32 client and Firebase server using HTTPS and TLS encryption. The TLS handshake establishes a secure connection with encryption key negotiation, verified by a certificate (Cert.pem). The client encrypts requests before sending them, and the server decrypts, processes, and encrypts responses before returning them. This ensures data integrity, confidentiality, and authentication for secure ESP32-Firebase communication.

### **Firebase Integration**

Firebase serves as the backend service for real-time data storage, retrieval, and synchronization.

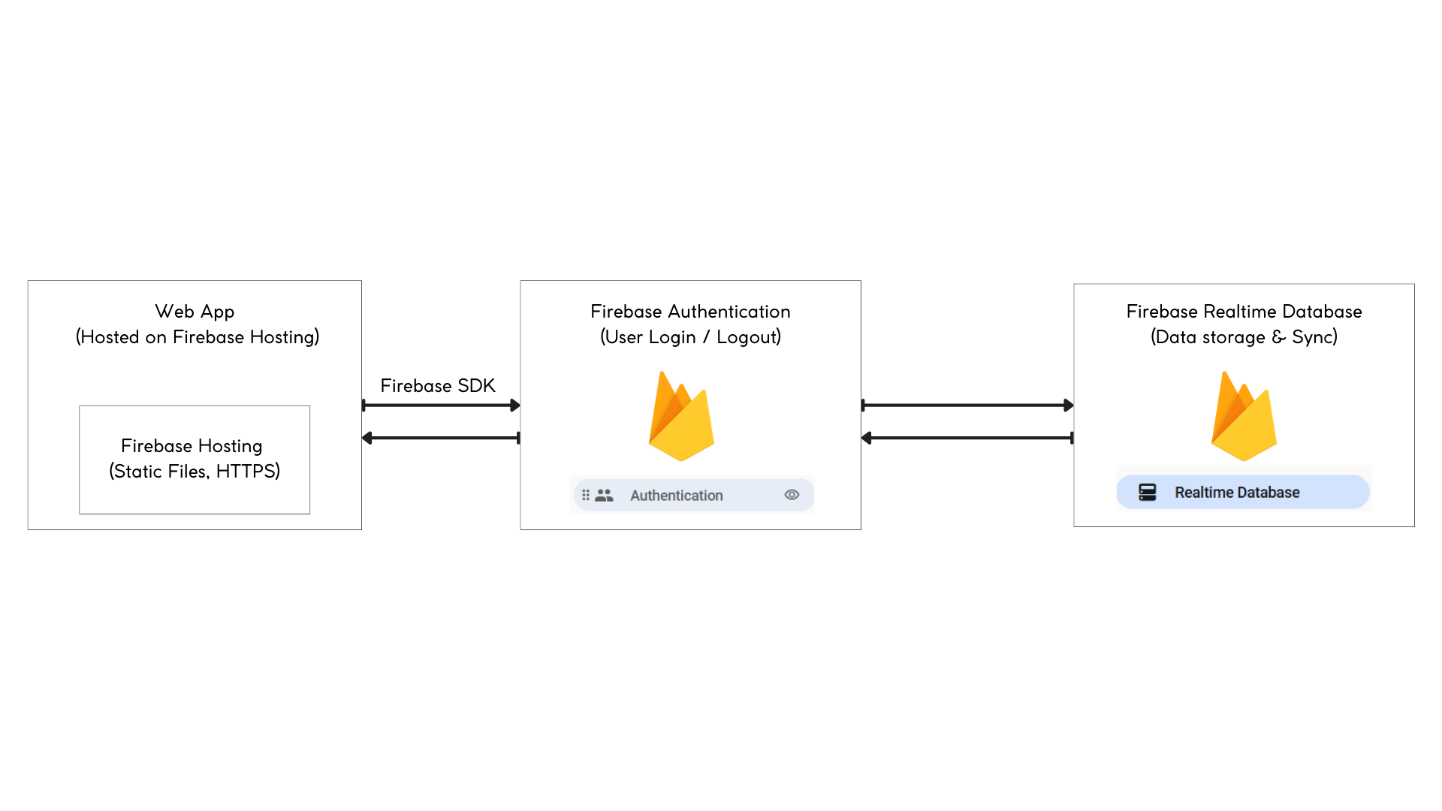


Figure 10: Block diagram of Firebase integrate with Web app

* Firebase Authentication

Firebase Authentication ensures secure access to the system by managing user identities and permissions. Users are required to authenticate themselves (e.g., via email/password or Google Sign-In) before accessing the system. Once authenticated, Firebase generates a unique token for the user, which is used to authorize subsequent requests to the Realtime Database. Firebase Authentication rules ensure that only authenticated users can read or write data to specific nodes in the database.

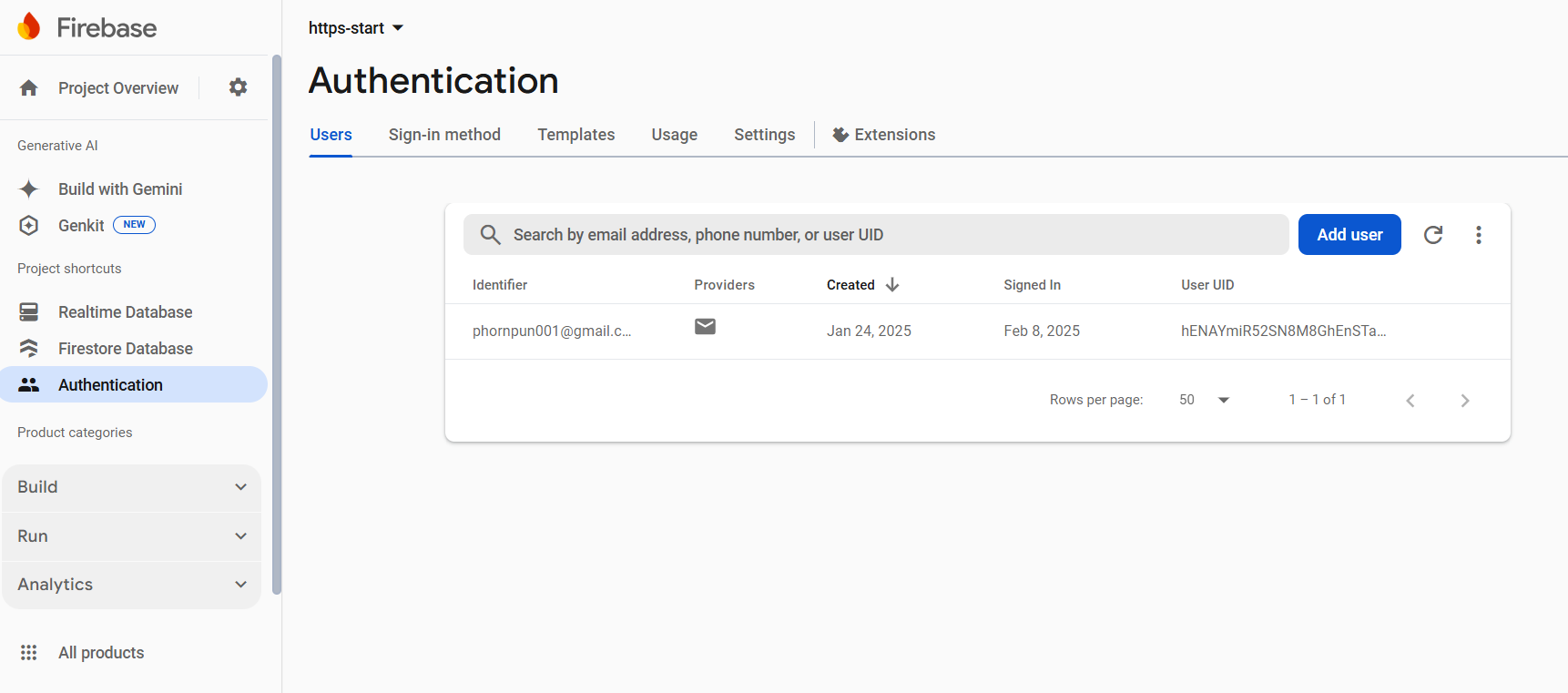


Figure 11: Set up Authentication

* Firebase Realtime Database

The Realtime Database serves as the central hub for storing and synchronizing data between the ESP32 and the web application. Stores sensor data (temperature, humidity, light intensity) collected by the ESP32 and control commands Relay states sent by users via the web application.

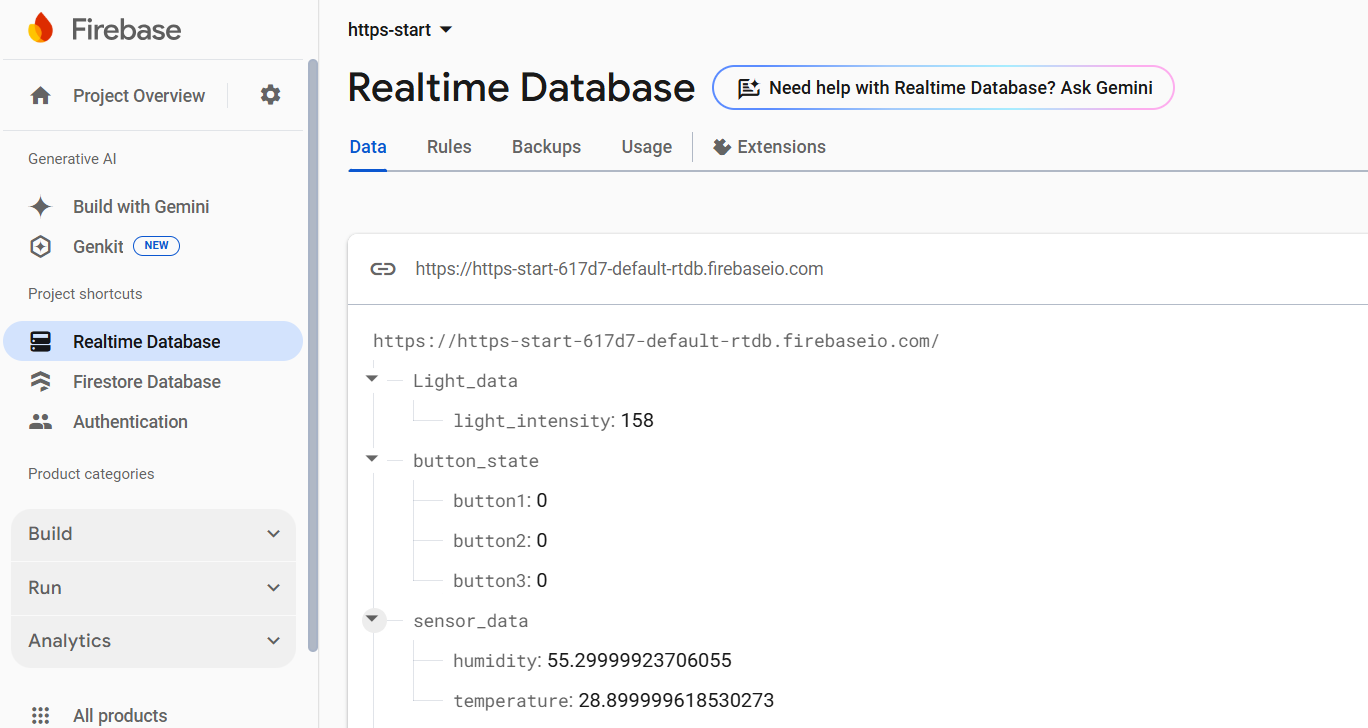
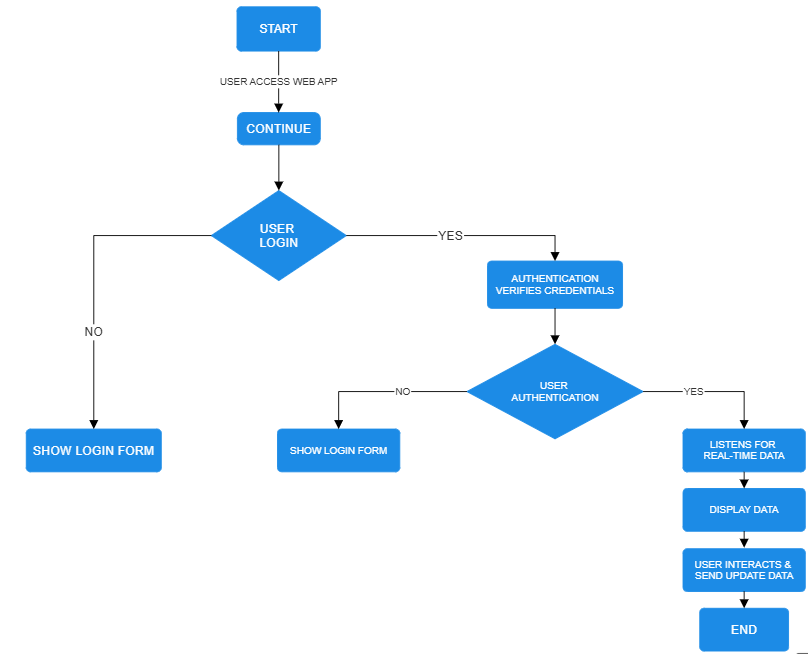


Figure 12: set up Realtime Database

### **Web Application Development**

A web application was developed to provide users with a user-friendly interface for monitoring sensor data and controlling the LEDs remotely. The web app was built using HTML, CSS, and JavaScript and hosted on Firebase for real-time updates and cross-platform accessibility.



The flowchart outlines the web app's process for real-time monitoring. It begins with user login via email and password, verified by Firebase Authentication. If valid, users access the main interface. The app retrieves and displays real-time sensor data from Firebase. Users can interact with the app to update relay states or perform actions, which are stored and synchronized by Firebase.

### **Firebase Hosting**

Firebase Hosting is a fully managed service provided by Firebase that allows developers to deploy and serve web applications securely and efficiently. It is designed to host static assets such as HTML, CSS, JavaScript, images, and other files required for a web application. Firebase Hosting ensures that your web app is delivered over HTTPS with minimal latency, leveraging Google’s global Content Delivery Network (CDN) for fast and reliable performance.

* Deployment of Web App Files:

The deployment process involves preparing web application files, configuring Firebase Hosting, and deploying the files to Firebase's servers.

* Log In to Firebase

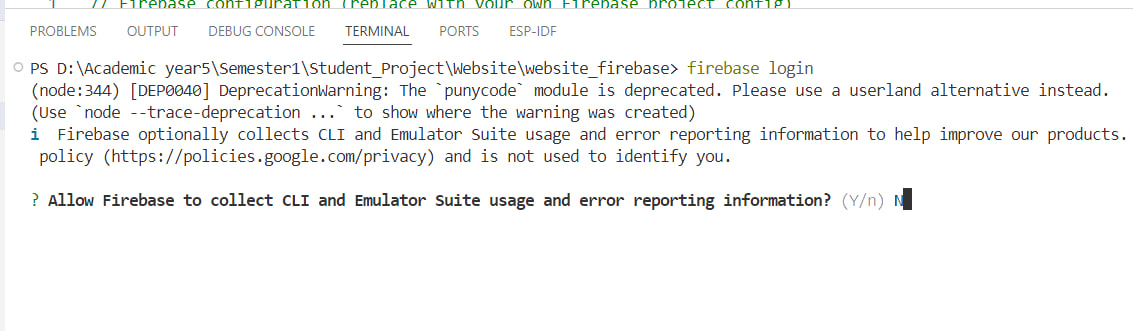


Figure 13: login firebase

This command opens a browser window where log in to Firebase account. Once logged in, the CLI gains access to Firebase projects.

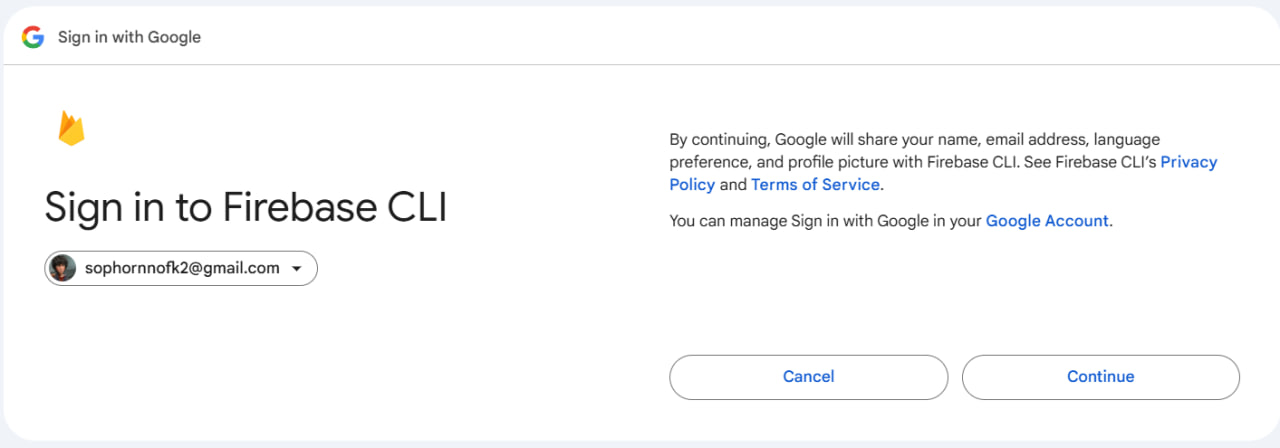
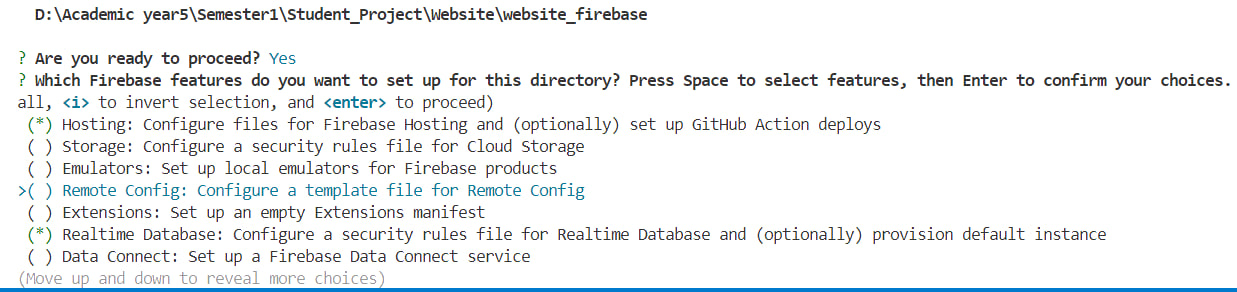


Figure 14: Sign in firebase with google

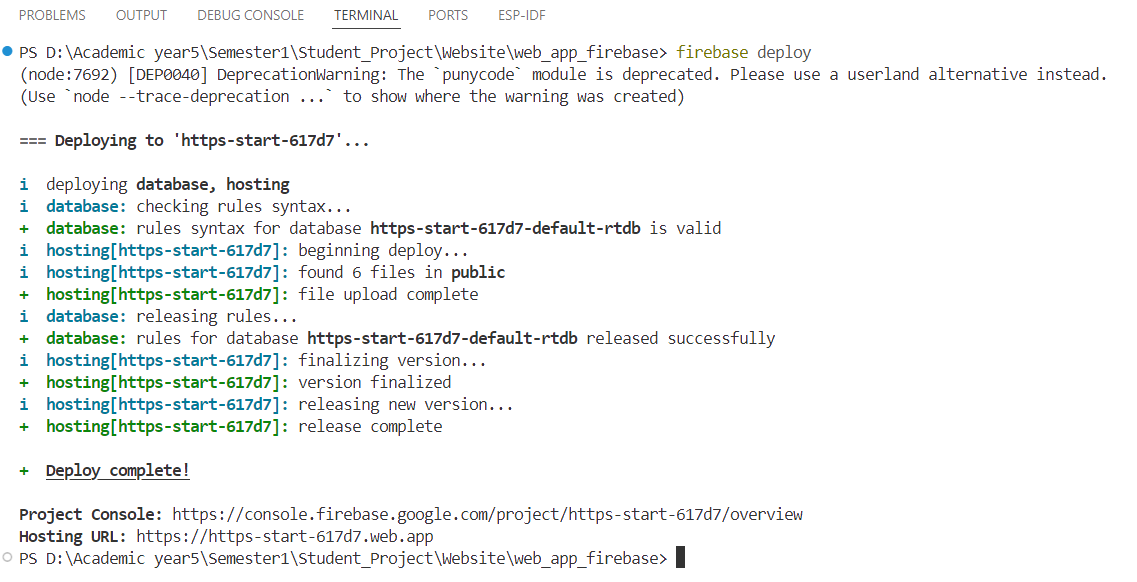
After this, a new window will open up in web browser, allowing us the opportunity to log into our Firebase account where we can access all our projects.

* Initialize Firebase in Project



In the initialize Firebase in Project was selected:

* Realtime Database: Configure security rules file for Realtime Database and (optionally) provision default instance.
* Hosting: Configure files for Firebase Hosting and (optionally) set up GitHub Action deploys
* Deploy to Firebase Hosting



URL

Figure 15: Deploy to Firebase Hosting

After running the Firebase deploy command, Firebase uploads the files from the public directory to its servers and provides a public URL for the app upon successful deployment.

# RESULT AND DISCUSSION

## **PCB Result and Testing**

|  |  |
| --- | --- |
|  |  |

Figure 16: The PCB 3D and outcome

The Figure 16, the left figure shows a 3D of the PCB, including components such as an **ESP32 module, relays, capacitors, and connectors.** The right image is the **actual manufactured PCB,** populated with components, showing a close resemblance to the 3D model.

|  |  |  |
| --- | --- | --- |
|  |  |  |

Figure 17: ESP32 Flashing

The first image shows the ESP32 connected via USB, with a red LED indicating power. The second image displays terminal output indicating a fault during the firmware flashing process with ESP-IDF. The final image is a screenshot of Device Manager, confirming the computer recognizes the USB-SERIAL CH340 (COM3).

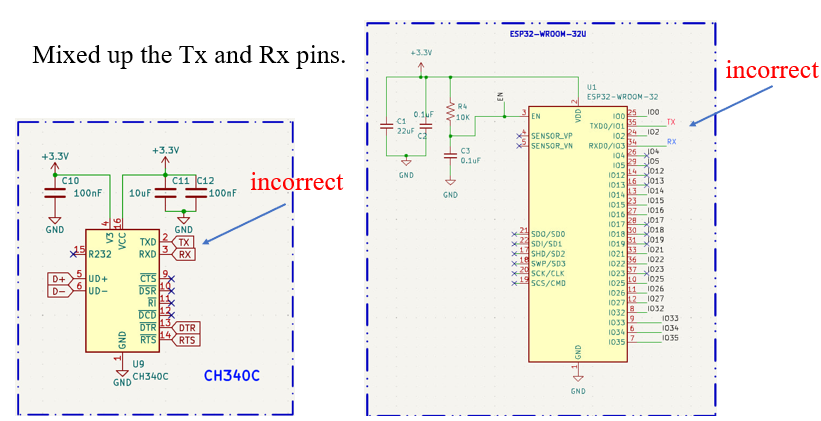


Figure 18: Fault between Tx and Rx of the ESP32 and CH340

This figure highlights a wiring error in the schematic design, where the Tx (transmit) and Rx (receive) pins of the ESP32 and CH340 USB-to-serial converter were swapped.

## **Firebase and ESP32 client testing for a web app.**

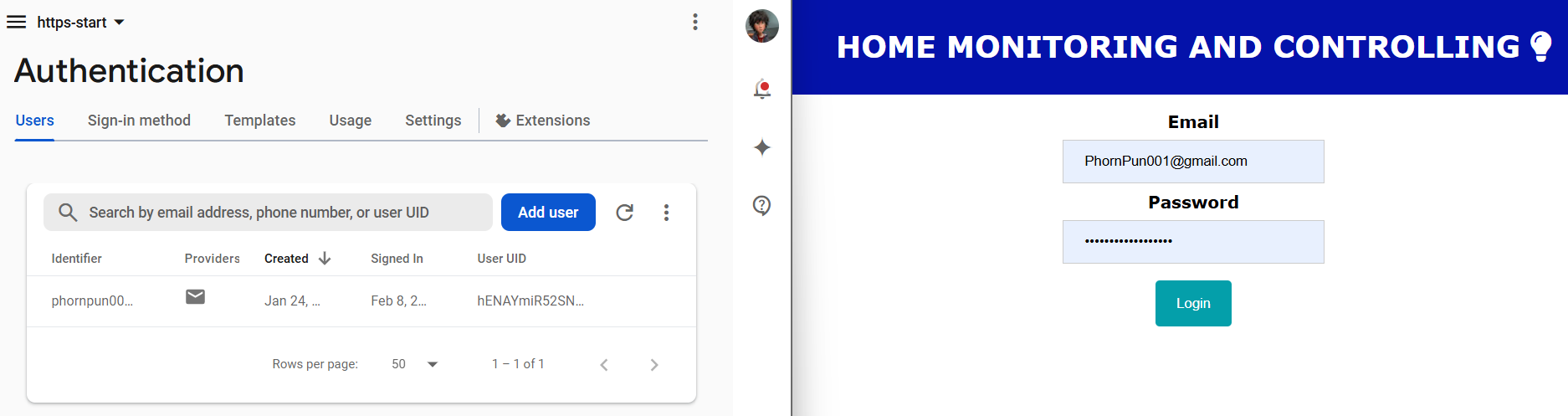


Figure 19: Authentication verifies

The figure shows the web app's login interface, where users enter their email and password to authenticate and gain access.

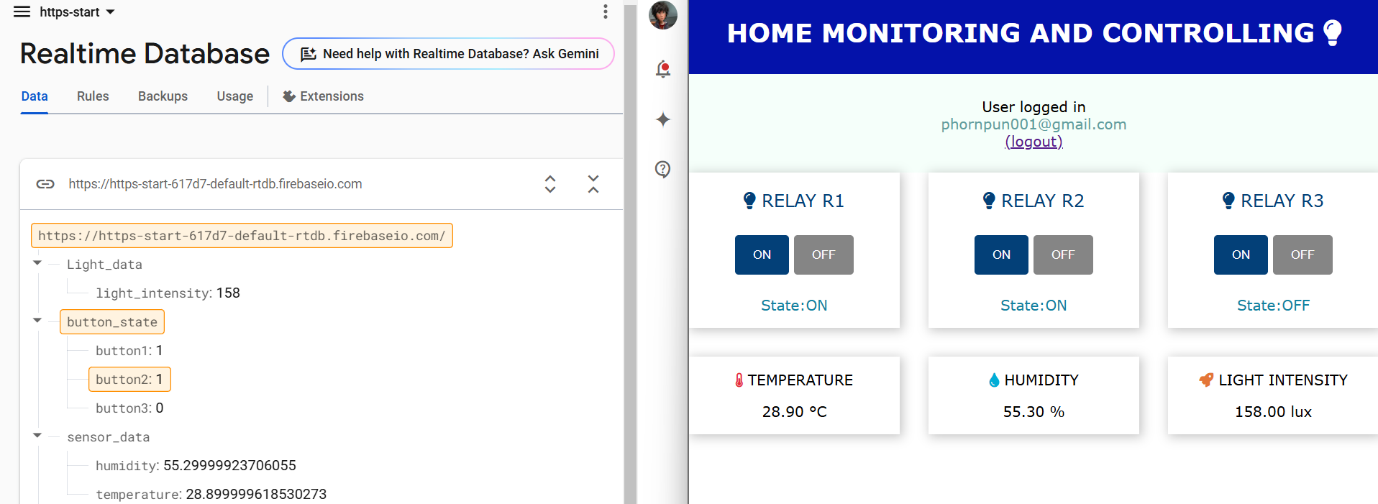


Figure 20: The Real-time Database interact with Web app

Figure 20 displays the interaction between the Firebase Realtime Database and the web application. On the left, the Firebase interface shows real-time data storage, including sensor readings (temperature, humidity, and light intensity) and button states. On the right, the web app interface provides a dashboard for controlling relays and viewing sensor data, with a user logged in.

|  |  |
| --- | --- |
| Button1: 1 |  |

Figure 21: Testing ESP32 with Web app

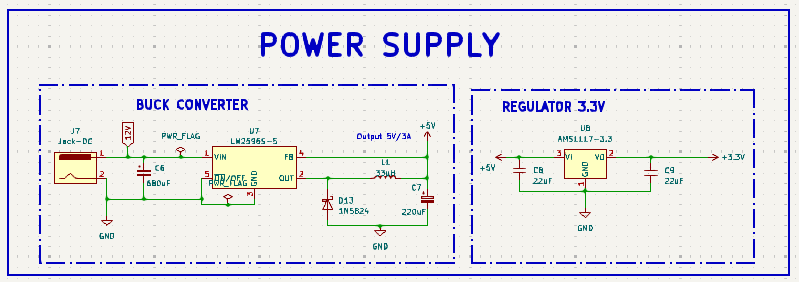
Figure 21 demonstrates the testing phase of the ESP32 with the web app. The left side shows a hardware setup on a breadboard, including an ESP32, relays, and wiring. The middle section shows Firebase reflecting the change in button state, while the right side highlights the corresponding update in the web interface, where the relay status is toggled.

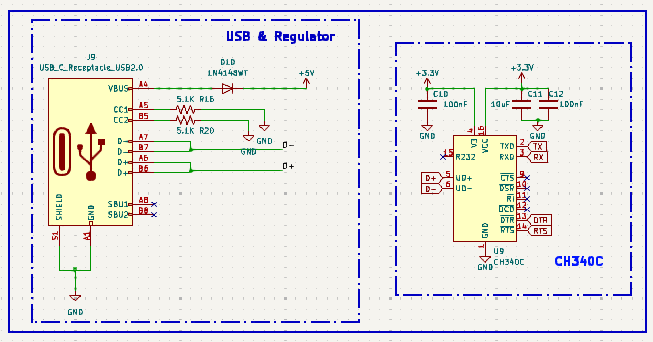
# CONCLUSION

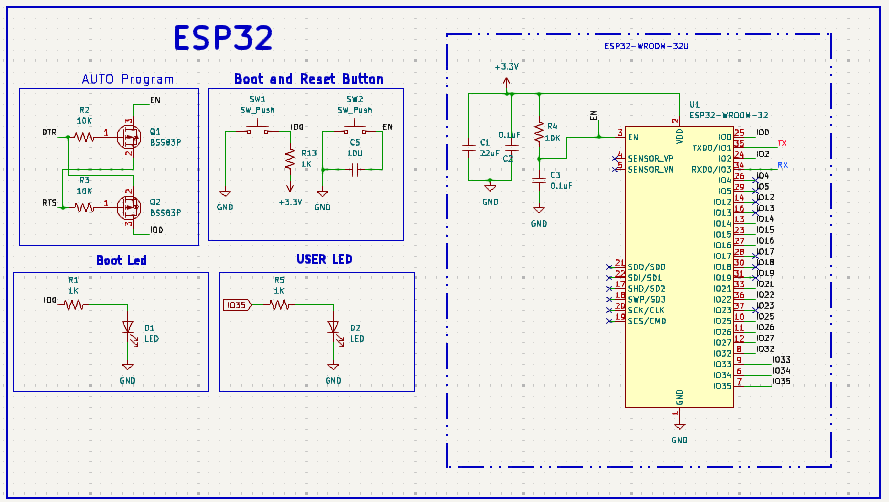
This project aims to develop a comprehensive IoT solution featuring an ESP32-based board equipped with sensors and a relay for real-time monitoring and control applications. The ESP32 is programmed using the ESP-IDF framework, enabling efficient hardware interaction and secure communication via the HTTPS protocol for data exchange with Firebase. Additionally, a web application hosted on Firebase Hosting provides a user-friendly interface for monitoring sensor data and controlling the relay remotely.

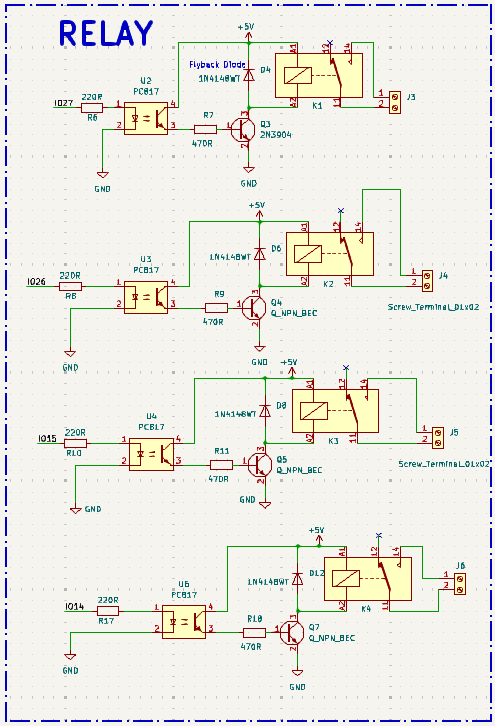
**Appendix:**

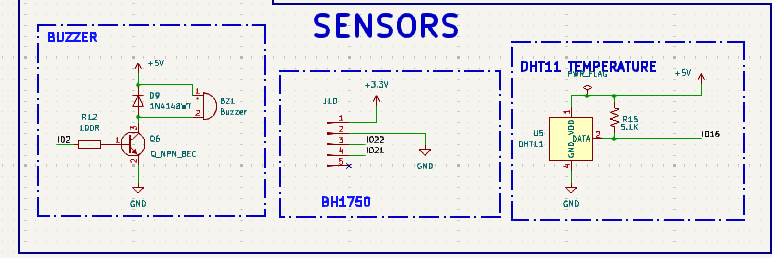
* **PCB**

****

****

****

****

****

* **Program:**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188  189  190  191  192  193  194  195  196  197  198  199  200  201  202  203  204  205  206  207  208  209  210  211  212  213  214  215  216  217  218  219  220  221  222  223  224  225  226  227  228  229  230  231  232  233  234  235  236  237  238  239  240  241  242  243  244  245  246  247  248  249  250  251  252  253  254  255  256  257  258  259  260  261  262  263  264  265  266  267  268  269  270  271  272  273  274  275  276  277  278  279  280  281  282  283  284  285  286  287  288  289  290  291  292  293  294  295  296  297  298  299  300  301  302  303  304  305  306  307  308  309  310  311  312  313  314  315  316  317  318  319  320  321  322  323  324  325  326  327  328  329  330 | *#include <stdio.h>*  *#include <string.h>*  *#include "freertos/FreeRTOS.h"*  *#include "freertos/task.h"*  *#include "freertos/event\_groups.h"*  *#include "esp\_log.h"*  *#include "nvs\_flash.h"*  //*#include "esp\_netif.h"*  *#include "esp\_http\_client.h"*  *#include "cJSON.h"*  *#include "dht.h"*  *#include "bh1750.h"*  *#include "protocol\_examples\_common.h"*  // --- Constants **and** Definitions ---  *#define I2C\_SDA GPIO\_NUM\_21*  *#define I2C\_SCK GPIO\_NUM\_22*  *#define SENSOR\_TYPE DHT\_TYPE\_AM2301*  *#define CONFIG\_DATA\_GPIO GPIO\_NUM\_4*  //button pins  *#define BUTTON1\_GPIO GPIO\_NUM\_2*  *#define BUTTON2\_GPIO GPIO\_NUM\_19*  *#define BUTTON3\_GPIO GPIO\_NUM\_23*  // Firebase URLs  *#define FIREBASE\_DHT\_URL "https://https-start-617d7-default-rtdb.firebaseio.com/sensor\_data.json"*  *#define FIREBASE\_LIGHT\_URL "https://https-start-617d7-default-rtdb.firebaseio.com/Light\_data.json"*  *#define FIREBASE\_BUTTON\_URL "https://https-start-617d7-default-rtdb.firebaseio.com/button\_state.json"*  *#define MAX\_BUFFER\_SIZE 256*  // External Certificates  extern const uint8\_t certificate\_pem\_start[] asm("\_binary\_certificate\_pem\_start");  extern const uint8\_t certificate\_pem\_end[] asm("\_binary\_certificate\_pem\_end");  // Event Groups **and** Tags  static const char \*TAG\_WIFI = "WiFi";  static const char \*TAG = "HTTP\_CLIENT";  static const char \*TAG\_DHT = "DHT\_SENSOR";  static const char \*TAG\_BH1750 = "BH1750\_SENSOR";  static const char \*TAG\_BUTTON = "BUTTON";  // --- Function Prototypes ---  void dht\_task(void \*params);  void bh1750\_task(void \*params);  void button\_task(void \*params);  void firebase\_task(void \*pvParameters);  /\* Callback **or** event handler \*/  esp\_err\_t http\_event\_handler(esp\_http\_client\_event\_t\* evt)  {  switch (evt->event\_id)  {  case HTTP\_EVENT\_ERROR:  ESP\_LOGI(TAG, "HTTP\_EVENT\_ERROR");  **break**;  case HTTP\_EVENT\_ON\_CONNECTED:  ESP\_LOGI(TAG, "HTTP\_EVENT\_ON\_CONNECTED");  **break**;  case HTTP\_EVENT\_ON\_DATA:  ESP\_LOGI(TAG, "HTTP\_EVENT\_ON\_DATA, len=**%d**, data=**%.\*s**", evt->data\_len, evt->data\_len, (char\*)evt->data);  **if** (evt->data\_len > MAX\_BUFFER\_SIZE)  **return** ESP\_FAIL;  **if** (evt->user\_data)  {  memcpy(evt->user\_data, evt->data, evt->data\_len);  }  **break**;  case HTTP\_EVENT\_DISCONNECTED:  ESP\_LOGD(TAG, "HTTP\_EVENT\_DISCONNECTED");  **break**;  default:  **break**;  }  **return** ESP\_OK;  }  /\* Functions GET method \*/  esp\_err\_t http\_client\_get\_req(char\* data, const char\* url)  {  esp\_err\_t ret\_code = ESP\_FAIL;  // HTTP client configuration  esp\_http\_client\_config\_t config = {  .event\_handler = http\_event\_handler,  .method = HTTP\_METHOD\_GET,  .port = 80,  .url = url,  .user\_data = data,  .cert\_pem = (const char \*)certificate\_pem\_start,  };  esp\_http\_client\_handle\_t client = esp\_http\_client\_init(&config);  esp\_err\_t err = esp\_http\_client\_perform(client);  **if** (err == ESP\_OK)  {  int status = esp\_http\_client\_get\_status\_code(client);  **if** (status == 200)  {  ESP\_LOGI(TAG, "HTTP GET status: **%d**", status);  ret\_code = ESP\_OK;  }  **else**  {  ESP\_LOGE(TAG, "HTTP GET status: **%d**", status);  }  }  **else**  {  ESP\_LOGE(TAG, "Failed to send GET request");  }  esp\_http\_client\_cleanup(client);  **return** ret\_code;  }  /\* HTTP POST function \*/  esp\_err\_t http\_client\_post\_req(const char\* data\_post, const char\* post\_url)  {  esp\_err\_t ret\_code = ESP\_FAIL;  // HTTP client configuration  esp\_http\_client\_config\_t config = {  .event\_handler = http\_event\_handler,  .method = HTTP\_METHOD\_PUT,  .port = 80,  .url = post\_url,  .cert\_pem = (const char \*)certificate\_pem\_start, // Use your certificate    };  esp\_http\_client\_handle\_t client = esp\_http\_client\_init(&config);  esp\_http\_client\_set\_header(client, "Content-Type", "application/json");  esp\_http\_client\_set\_post\_field(client, data\_post, strlen(data\_post));    esp\_err\_t err = esp\_http\_client\_perform(client);  **if** (err == ESP\_OK)  {  int status = esp\_http\_client\_get\_status\_code(client);  **if** (status == 200)  {  ESP\_LOGI(TAG, "HTTP POST status: **%d**", status);  ret\_code = ESP\_OK;  }  **else**  {  ESP\_LOGE(TAG, "HTTP POST status: **%d**", status);  }  }  **else**  {  ESP\_LOGE(TAG, "Failed to send POST request");  }  esp\_http\_client\_cleanup(client);  **return** ret\_code;  }  void button\_task(void\* arg) {  char data[MAX\_BUFFER\_SIZE] = {0};  gpio\_config\_t io\_conf = {  .pin\_bit\_mask = (1ULL << BUTTON1\_GPIO) | (1ULL << BUTTON2\_GPIO) | (1ULL << BUTTON3\_GPIO),  .mode = GPIO\_MODE\_OUTPUT,  .pull\_up\_en = GPIO\_PULLUP\_DISABLE,  .pull\_down\_en = GPIO\_PULLDOWN\_DISABLE,  .intr\_type = GPIO\_INTR\_DISABLE  };  gpio\_config(&io\_conf);  gpio\_set\_level(BUTTON1\_GPIO, 0);  gpio\_set\_level(BUTTON2\_GPIO, 0);  gpio\_set\_level(BUTTON3\_GPIO, 0);  **while** (1)  {  // Send HTTP GET request to Firebase to retrieve button states  **if** (http\_client\_get\_req(data, FIREBASE\_BUTTON\_URL) == ESP\_OK)  {  ESP\_LOGI(TAG\_BUTTON, "Received button states: **%s**", data);  // Parse the JSON response  cJSON\* json = cJSON\_Parse(data);  **if** (json == NULL)  {  ESP\_LOGE(TAG\_BUTTON, "Failed to parse JSON response.");  }  **else**  {  // Extract button states **from** **the** **JSON**  cJSON\* button1 = cJSON\_GetObjectItem(json, "button1");  cJSON\* button2 = cJSON\_GetObjectItem(json, "button2");  cJSON\* button3 = cJSON\_GetObjectItem(json, "button3");  **if** (button1 && button2 && button3)  {  int button1\_state = button1->valueint;  int button2\_state = button2->valueint;  int button3\_state = button3->valueint;  ESP\_LOGI(TAG\_BUTTON, "Button1: **%d**, Button2: **%d**, Button3: **%d**", button1\_state, button2\_state, button3\_state);  // Control LEDs based on Firebase button states  gpio\_set\_level(BUTTON1\_GPIO, button1\_state);  gpio\_set\_level(BUTTON2\_GPIO, button2\_state);  gpio\_set\_level(BUTTON3\_GPIO, button3\_state);  }  **else**  {  ESP\_LOGE(TAG\_BUTTON, "Failed to extract button states from JSON.");  }  cJSON\_Delete(json); // Free the JSON object  }  }  **else**  {  ESP\_LOGE(TAG\_BUTTON, "Failed to retrieve button states from Firebase.");  }  vTaskDelay(pdMS\_TO\_TICKS(1000));  }  vTaskDelete(NULL);  }  // --- DHT Sensor Task ---  void dht\_task(void\* arg)  {  float temp, humidity;  **while** (1)  {  **if** (dht\_read\_float\_data(SENSOR\_TYPE, CONFIG\_DATA\_GPIO, &humidity, &temp) == ESP\_OK)  {  ESP\_LOGI(TAG\_DHT, "Humidity: **%.1f%%**, Temp: **%.1f**C", humidity, temp);  // Create JSON payload  cJSON\* json = cJSON\_CreateObject();  cJSON\_AddNumberToObject(json, "temperature", temp);  cJSON\_AddNumberToObject(json, "humidity", humidity);  char\* data = cJSON\_Print(json);  // Send data to Firebase  **if** (http\_client\_post\_req(data, FIREBASE\_DHT\_URL) == ESP\_OK)  {  ESP\_LOGI(TAG, "DHT data uploaded successfully.");  }  **else**  {  ESP\_LOGE(TAG, "Failed to upload DHT data.");  }  cJSON\_Delete(json);  free(data);  }  **else**  {  ESP\_LOGE(TAG\_DHT, "Failed to read DHT sensor.");  }  vTaskDelay(pdMS\_TO\_TICKS(1000));  }  vTaskDelete(NULL);  }  void bh1750\_task(void\* arg)  {  i2c\_dev\_t dev = { 0 };  **if** (bh1750\_init\_desc(&dev, BH1750\_ADDR\_LO, 0, I2C\_SDA, I2C\_SCK) != ESP\_OK ||  bh1750\_setup(&dev, BH1750\_MODE\_CONTINUOUS, BH1750\_RES\_HIGH) != ESP\_OK)  {  ESP\_LOGE(TAG\_BH1750, "Failed to initialize BH1750.");  vTaskDelete(NULL);  }  **while** (1)  {  uint16\_t lux;  **if** (bh1750\_read(&dev, &lux) == ESP\_OK)  {  ESP\_LOGI(TAG\_BH1750, "Light Intensity: **%d** lux", lux);  // Create JSON payload  cJSON\* json = cJSON\_CreateObject();  cJSON\_AddNumberToObject(json, "light\_intensity", lux);  char\* data = cJSON\_Print(json);  // Send data to Firebase  **if** (http\_client\_post\_req(data, FIREBASE\_LIGHT\_URL) == ESP\_OK)  {  ESP\_LOGI(TAG, "BH1750 data uploaded successfully.");  }  **else**  {  ESP\_LOGE(TAG, "Failed to upload BH1750 data.");  }  cJSON\_Delete(json);  free(data);  }  **else**  {  ESP\_LOGE(TAG\_BH1750, "Failed to read BH1750.");  }  vTaskDelay(pdMS\_TO\_TICKS(1000)); // Delay **for** 1 second  }  vTaskDelete(NULL);  }  void app\_main(void) {  ESP\_ERROR\_CHECK(nvs\_flash\_init());  ESP\_ERROR\_CHECK(esp\_netif\_init());  ESP\_ERROR\_CHECK(esp\_event\_loop\_create\_default());  esp\_err\_t err = ESP\_FAIL;  **while** (err != ESP\_OK)  {  err = example\_connect();  **if** (err != ESP\_OK)  {  ESP\_LOGE(TAG, "Unable to connect to WiFi.");  vTaskDelay(pdMS\_TO\_TICKS(10000));  }  }  **if** (i2cdev\_init() != ESP\_OK) {  ESP\_LOGE(TAG\_WIFI, "Failed to initialize I2C.");  **return**;  }  // Start sensor tasks  xTaskCreate(dht\_task, "DHT Task", 4096, NULL, 2, NULL);  xTaskCreate(bh1750\_task, "BH1750 Task", 4096, NULL, 2, NULL);  xTaskCreate(button\_task, "Button Task", 4096, NULL, 2, NULL);  vTaskDelete(NULL);  } |