BD526: Internet of Things and Real Time Data Analytics

College of Innovative Technology and Engineering, Dhurakij Pundit University

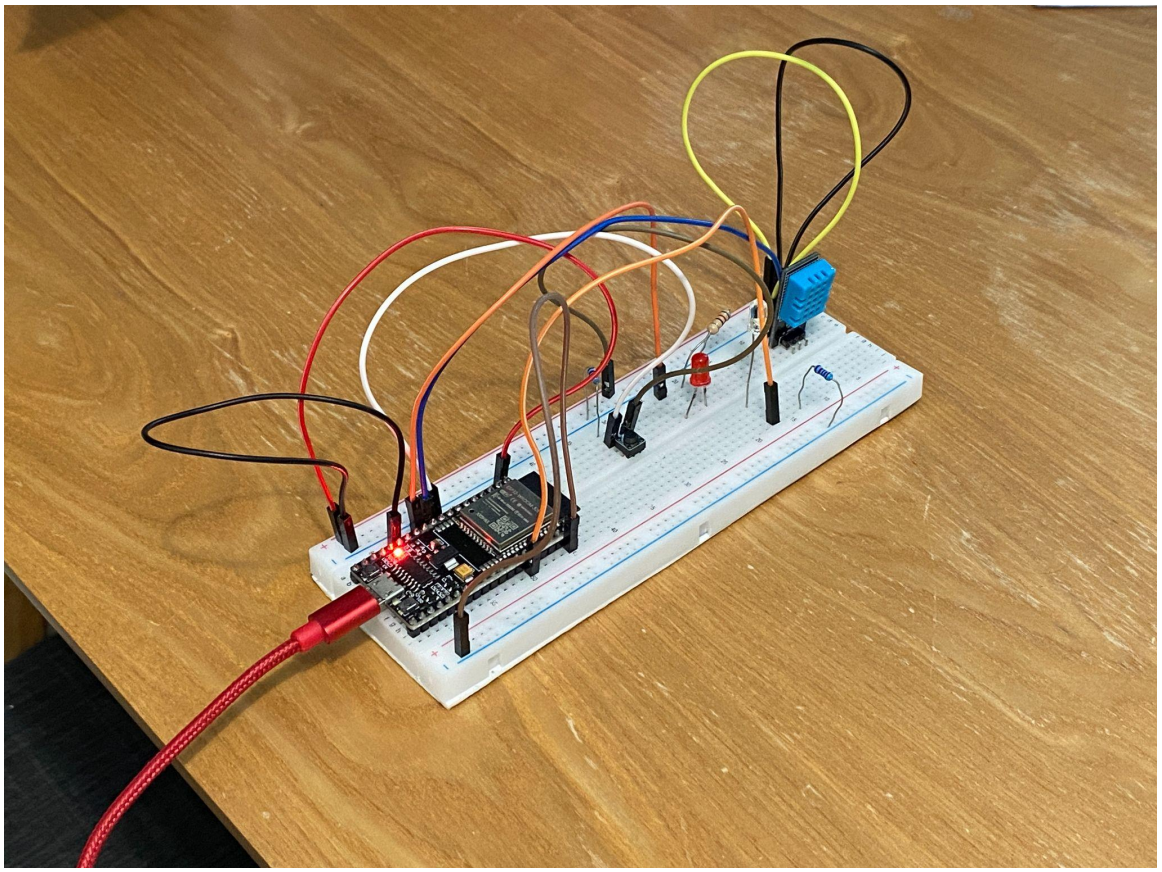
Project Title: ESP32 + Blynk IoT platform

Author	Witchakorn Wanasanwongkot	**StudentID**	65130459
	Thongpan Supong		65130504

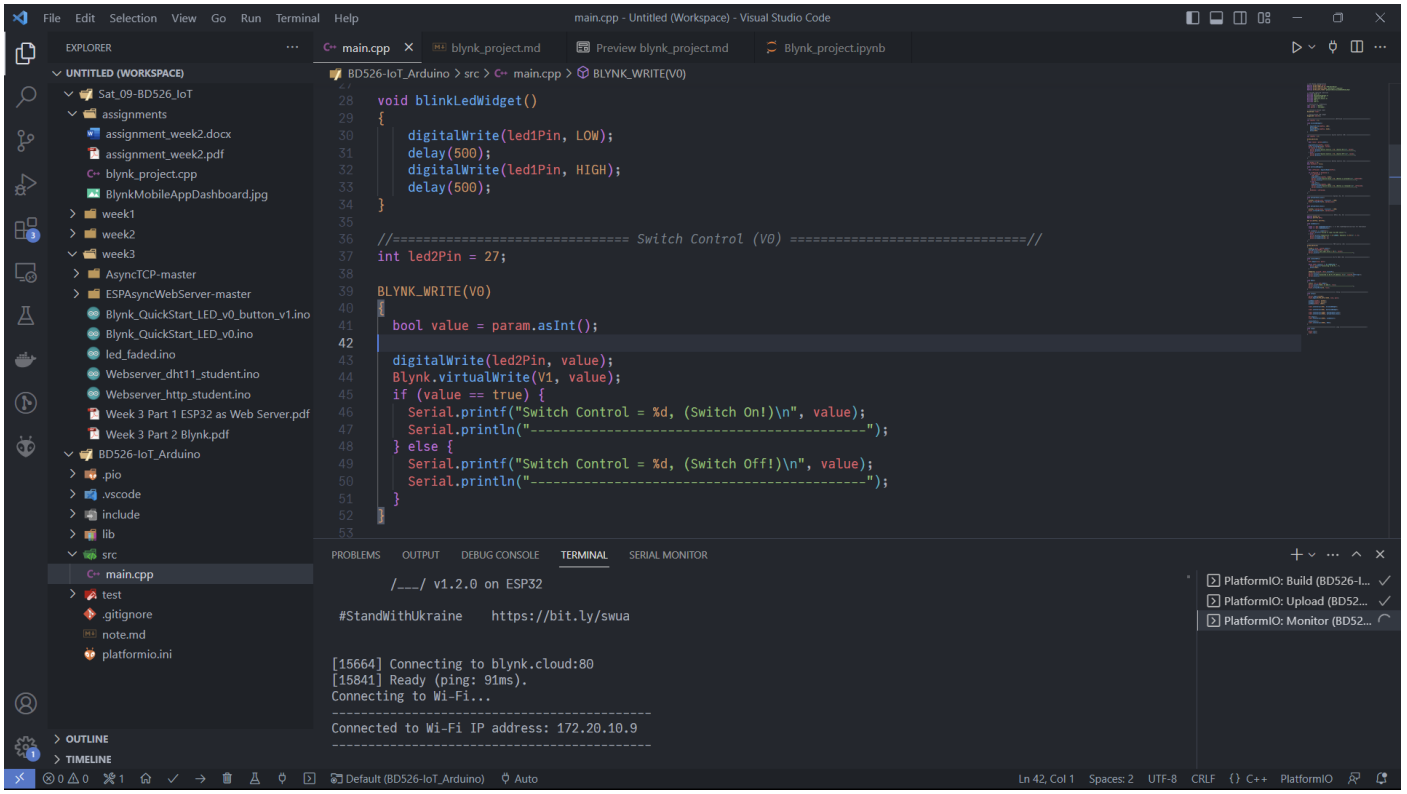
Table of Contents

Reference Images

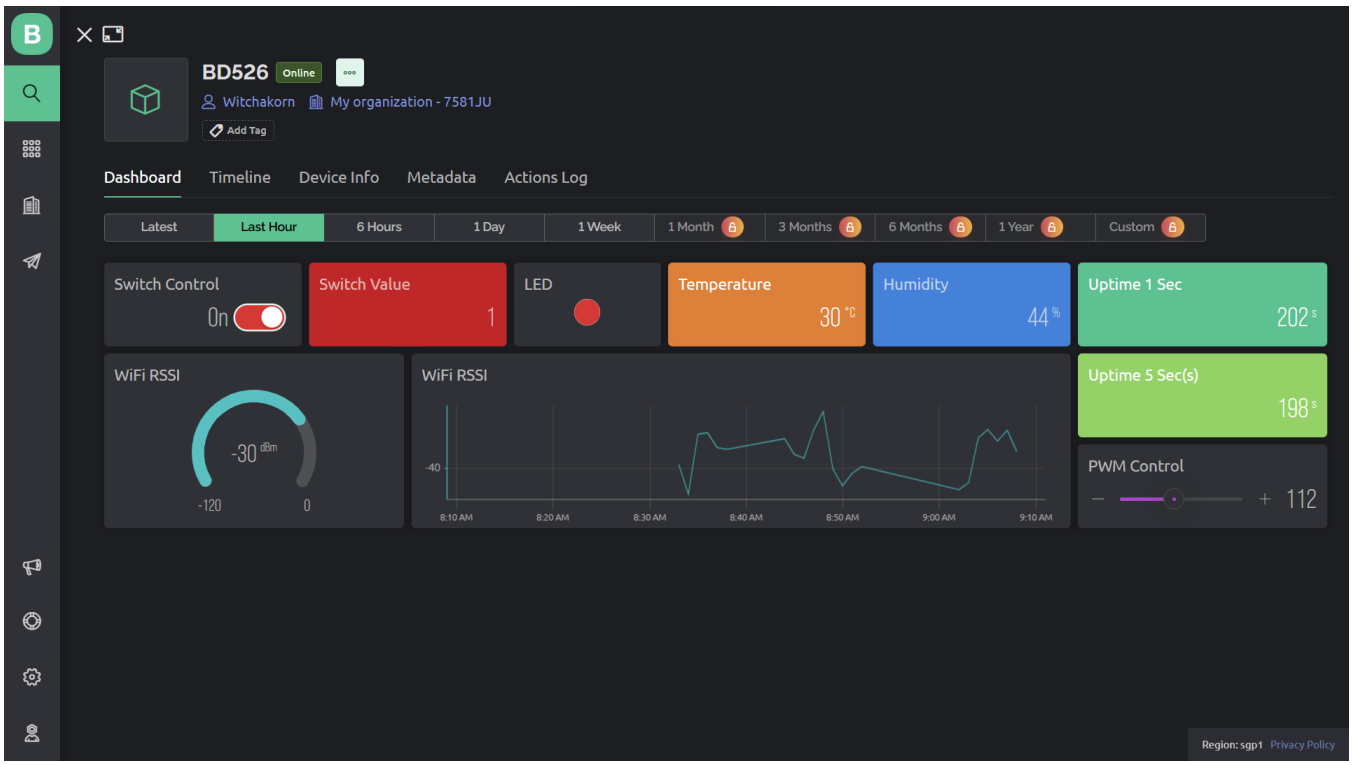
ESP32-WROOM-32E (Wi-Fi + Bluetooth)



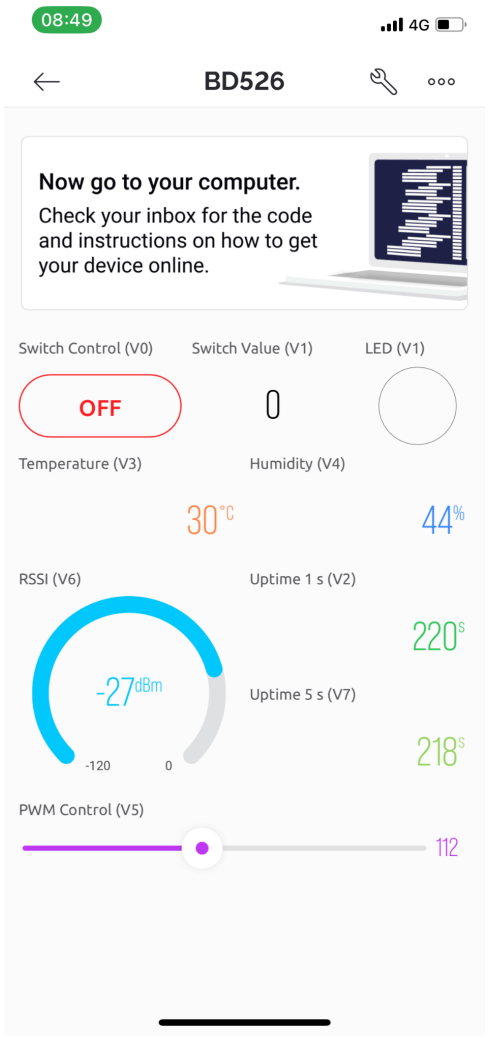
Arduino Code



Blynk Web App Dashboard



Blynk Mobile App Dashboard



Arduino Code Explanation

Blynk configuration for an ESP32 device

```
// Set Blynk configuration
#define BLYNK_PRINT Serial
#define BLYNK_TEMPLATE_ID "TMPL66Ct5RtaA"
#define BLYNK_TEMPLATE_NAME "Quickstart Template"
#define BLYNK_AUTH_TOKEN "g5XeoArvNVnnrZ1zcZS8ZhAVNnOLjHhg"

// Include required libraries
#include <WiFi.h>
#include <BlynkSimpleEsp32.h>
#include <Adafruit_GFX.h>
#include <Adafruit_Sensor.h>
#include <FS.h>
#include <DHT.h>

// Define Wi-Fi credentials
char ssid[] = "Natto";
char pass[] = "natto053";

// Initialize Blynk timer
BlynkTimer timer;

// Define Blynk LED widget
WidgetLED led1(V1);
```

This code serves as the basic setup for establishing a connection to the Blynk server and configuring the necessary components for Blynk functionality on the ESP32 device.

- ★ Include the necessary libraries.
- ★ Define the Wi-Fi credentials for connecting to the network as `ssid` and `pass`.
- ★ Initialize the Blynk timer as `BlynkTimer timer`, which allows for scheduling timed events.
- ★ Define the Blynk LED widget as `WidgetLED led1(V1)`, which represents a virtual LED in the Blynk app interface.

LED Blink

```
//================================== LED Blink ==================================//
int led1Pin = 21;

void blinkLedWidget()
{
    digitalWrite(led1Pin, LOW);
    delay(500);
    digitalWrite(led1Pin, HIGH);
    delay(500);
}
```

This code snippet demonstrates a simple method to control an LED and achieve a blinking behavior using the ESP32 board.

- ★ Define the pin number `led1Pin` as 21, which represents the GPIO pin connected to an LED on the ESP32 board.
- ★ Define the `blinkLedWidget()` function, that is responsible for toggling the state of the LED.
- ★ It first sets the pin to a low voltage level (`LOW`), turning off the LED, and then introduces a delay of 500 milliseconds using the `delay()` function.
- ★ After the delay, it sets the pin to a high voltage level (`HIGH`), turning on the LED, and introduces another delay of 500 milliseconds.
- ★ This pattern of toggling the LED state with a delay is repeated continuously, creating a blinking effect.

Switch Control

```
//============================== Switch Control (V0) ==============================//
int led2Pin = 27;

BLYNK_WRITE(V0)
{
  bool value = param.asInt();

  digitalWrite(led2Pin, value);
  Blynk.virtualWrite(V1, value);
  if (value == true) {
    Serial.printf("Switch Control = %d, (Switch On!)\n", value);
    Serial.println("-------------------------------------------");
  } else {
    Serial.printf("Switch Control = %d, (Switch Off!)\n", value);
    Serial.println("-------------------------------------------");
  }
}
```

This code snippet demonstrates how to control an LED based on the value received from a Blynk app widget and provide visual feedback on the Blynk app interface and Serial monitor.

- ★ Define the pin number `led2Pin` as 27, which represents the GPIO pin connected to another LED on the ESP32 board.
- ★ Define the `BLYNK_WRITE(V0)` function, that is a callback function that gets triggered whenever the value of virtual pin `V0` in the Blynk app is changed.
- ★ Inside the function, the value of the virtual pin is retrieved using `param.asInt()`, which returns an integer value.
- ★ The `digitalWrite()` function is then used to set the state of `led2Pin` based on the value received.
- ★ Additionally, `Blynk.virtualWrite()` is used to update the state of virtual pin `V1` in the Blynk app, reflecting the current state of the LED.
- ★ Depending on the value of `value`, the function also prints a corresponding message to the Serial monitor indicating whether the switch is on or off.

Button Control

```
//============================== Button Control (V1) ==============================//
int btnPin = 25;
bool btnState = false;

void buttonLedWidget()
{
  bool isPressed = digitalRead(btnPin);

  if (isPressed != btnState) {
    if (isPressed) {
      led1.on();
      digitalWrite(led2Pin, HIGH);
      Serial.printf("Switch Value = %d, (Button is pressed!)\n", isPressed);
      Serial.println("-----------------------------------------");
    } else {
      led1.off();
      digitalWrite(led2Pin, LOW);
      Serial.printf("Switch Value = %d, (Button is released!)\n", isPressed);
      Serial.println("-----------------------------------------");
    }
    btnState = isPressed;
  }
}
```

This code snippet demonstrates how to detect button presses and releases and control LEDs based on the button state.

- ★ Define the pin number `btnPin` as 25, which represents the GPIO pin connected to a push button on the ESP32 board.
- ★ Initialize the boolean variable `btnState` as `false`, representing the initial state of the button.
- ★ Define the `buttonLedWidget()` function, that is responsible for detecting changes in the button state and updating the LED and Serial monitor accordingly. It uses the `digitalRead()` function to read the state of `btnPin` and assigns the result to the boolean variable `isPressed`.
- ★ If there is a change in the button state (i.e., `isPressed` is different from `btnState`), the function proceeds to check whether the button is pressed or released. If the button is pressed, it turns on `led1` and sets `led2Pin` to a high voltage level (`HIGH`), indicating the button press event. Conversely, if the button is released, it turns off `led1` and sets `led2Pin` to a low voltage level (`LOW`), indicating the button release event.
- ★ The function also prints a corresponding message to the Serial monitor based on the button state change.
- ★ Finally, it updates `btnState` with the current button state for future comparisons.

Uptime

```
//================================ Uptime (V2, V7) ================================//
void myTimerEvent_1sec()
{
  uint32_t uptime_1sec = millis() / 1000;
  Blynk.virtualWrite(V2, uptime_1sec);
}

void myTimerEvent_5sec()
{
  uint32_t uptime_5sec = millis() / 1000;
  Blynk.virtualWrite(V7, uptime_5sec);
}
```

These code snippets demonstrate how to use timers to perform periodic tasks and update Blynk app widgets with relevant information.

- ★ Define two timer events that are triggered at different intervals.
- ★ The `myTimerEvent_1sec()` function is executed every 1 second, while the `myTimerEvent_5sec()` function is executed every 5 seconds.
- ★ Inside each function, the `millis()` function is used to retrieve the current time in milliseconds. This value is then divided by 1000 to convert it into seconds. The resulting value represents the uptime in seconds.
- ★ The `Blynk.virtualWrite()` function is used to send this uptime value to virtual pins `V2` and `V7` in the Blynk app. By doing so, the Blynk app can display and monitor the system's uptime in real-time.

DHT (Digital Humidity and Temperature) Sensor

```
//================================ DHT11 (V3, V4) ================================//
#define DHTPIN 26
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

void sendSensor()
{
  float t = dht.readTemperature(); // or dht.readTemperature(true) for Fahrenheit
  float h = dht.readHumidity();

  if (isnan(t) || isnan(h)) {
    Serial.println("Failed to read from DHT sensor!");
  } else {
    Serial.printf("Temperature: %.2f\u00B0C, Humidity: %.2f%%\n", t, h);
    Blynk.virtualWrite(V3, t);
    Blynk.virtualWrite(V4, h);
  }
}
```

This code snippet demonstrates how to interface with a DHT sensor and retrieve temperature and humidity data, as well as display and log the values in a Blynk app.

- ★ Set up the DHT sensor by defining the pin number `DHTPIN` as 26 and specifying the

sensor type as DHT11 using the `DHTTYPE` constant. The `DHT` object is then initialized with the pin and sensor type.

- ★ The `sendSensor()` function is responsible for reading the temperature and humidity values from the DHT sensor. It calls the `readTemperature()` function to retrieve the temperature in Celsius and assigns it to the variable `t`. Similarly, it calls the `readHumidity()` function to retrieve the humidity value and assigns it to the variable `h`.
- ★ The code then checks if the temperature or humidity values are NaN (not a number) using the `isnan()` function. If either value is NaN, indicating a failed sensor reading, an error message is printed to the Serial monitor. Otherwise, the temperature and humidity values are printed to the Serial monitor with two decimal places of precision using `printf()`.
- ★ Additionally, the temperature and humidity values are sent to virtual pins `V3` and `V4` in the Blynk app using `Blynk.virtualWrite()`.

PWM (Pulse-Width Modulation) Control

```
//================================ PWM Control (V5) ================================//
BLYNK_WRITE(V5)
{
  uint8_t value = param.asInt();
  analogWrite(led2Pin, value);
  Serial.printf("LED Light Value = %d\n", value);
  Serial.println("-----------------------------------------");
}
```

This code snippet demonstrates how to receive and process data from a Blynk app by handling changes in a virtual pin's value and controlling an LED's brightness level accordingly.

- ★ Define a Blynk virtual pin handler for virtual pin `V5` using the `BLYNK_WRITE()` macro.
- ★ When the value of virtual pin `V5` is changed in the Blynk app, this handler function is automatically called.
- ★ Inside the function, the new value of the virtual pin is retrieved using `param.asInt()`, which returns an integer value. The `analogWrite()` function is then used to control the LED connected to `led2Pin` by setting its brightness level based on the value received.
- ★ The value is also printed to the Serial monitor using `printf()` for debugging purposes.

Wi-Fi RSSI (Received Signal Strength Indicator)

```
//================================ Wi-Fi RSSI (V6) ================================//
void connectWiFi()
{
  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED) {
    Serial.println("Connecting to Wi-Fi...");
    delay(1000);
  }

  IPAddress localIP = WiFi.localIP();
  Serial.println("-----------------------------------------");
  Serial.printf("Connected to Wi-Fi IP address: %s\n", localIP.toString());
  Serial.println("-----------------------------------------");
}

void RSSI()
{
  int8_t rssi = WiFi.RSSI();
  Serial.printf("RSSI: %d dBm\n", rssi);
  Serial.println("-----------------------------------------");
  Blynk.virtualWrite(V6, rssi);
}
```

This code snippet demonstrates how to connect to a Wi-Fi network, retrieve the RSSI value, and provide visual feedback on the Serial monitor and Blynk app interface.

- ★ Define the `connectWiFi()` function, that is responsible for connecting the ESP32 device to a Wi-Fi network using the provided `ssid` and `pass` credentials.
- ★ It calls the `WiFi.begin()` function to initiate the connection process. It then enters a loop that continuously checks the Wi-Fi status using `WiFi.status()`. While the device is not connected (`WL_CONNECTED`), it prints "Connecting to Wi-Fi..." to the Serial monitor and introduces a delay of 1 second using `delay(1000)`.
- ★ Once the device successfully connects to the Wi-Fi network, it retrieves the local IP address using `WiFi.localIP()` and assigns it to the `localIP` variable of type `IPAddress`.
- ★ The function concludes by printing a separator line of dashes, followed by the message "Connected to Wi-Fi IP address: " and the actual IP address to the Serial monitor.
- ★ Define the `RSSI()` function, which retrieves the received signal strength indicator (RSSI) of the Wi-Fi connection using `WiFi.RSSI()`, which returns the RSSI value in dBm (decibels per milliwatt).
- ★ The RSSI value is then printed to the Serial monitor using `printf()` for debugging purposes.
- ★ Additionally, the RSSI value is sent to the virtual pin `V6` in the Blynk app using `Blynk.virtualWrite()`.

Setup

```
//================================== Setup ==================================//
void setup()
{
  Serial.begin(115200);
  Blynk.begin(BLYNK_AUTH_TOKEN, ssid, pass);

  pinMode(led1Pin, OUTPUT);
  pinMode(led2Pin, OUTPUT);
  pinMode(btnPin, INPUT);

  timer.setInterval(500L, blinkLedWidget);

  timer.setInterval(500L, buttonLedWidget);

  timer.setInterval(1000L, myTimerEvent_1sec);
  timer.setInterval(5000L, myTimerEvent_5sec);

  dht.begin();
  timer.setInterval(5000L, sendSensor);

  connectWiFi();
  timer.setInterval(5000L, RSSI);
}
```

The `setup()` function is a standard Arduino function that is automatically called when the microcontroller starts up.

- ★ The `Serial.begin()` function is called to initialize the serial communication with a baud rate of 115200, allowing for communication between the ESP32 board and the computer via the Serial monitor.
- ★ The `Blynk.begin()` function is used to establish a connection with the Blynk server by passing the Blynk authentication token (`BLYNK_AUTH_TOKEN`) along with the Wi-Fi credentials (`ssid` and `pass`).
- ★ The pin modes for `led1Pin`, `led2Pin`, and `btnPin` are set using `pinMode()` to configure them as outputs and input, respectively. This allows for controlling LEDs and reading the button state.
- ★ The `timer.setInterval()` function is used to set up timers for various intervals.
- ★ The `blinkLedWidget()` function is scheduled to run every 500 milliseconds, creating a blinking effect on the LED connected to `led1Pin`. Similarly, the `buttonLedWidget()` function is scheduled to run every 500 milliseconds, handling the button and controlling the LEDs accordingly.
- ★ The `myTimerEvent_1sec()` function is scheduled to run every 1 second, updating virtual pin `V2` with the current uptime in seconds. Similarly, the `myTimerEvent_5sec()` function is scheduled to run every 5 seconds, updating virtual pin `V7` with the current uptime in seconds.
- ★ The `sendSensor()` function is scheduled to run every 5 seconds, reading the temperature and humidity values from the DHT sensor and updating virtual pins `V3` and `V4` with the respective values.
- ★ The `connectWiFi()` function is scheduled to run every 5 seconds, attempting to connect to the Wi-Fi network. The `RSSI()` function is also scheduled to run every 5 seconds,

retrieving the Wi-Fi signal strength (RSSI) and updating the virtual pin `V6` with the RSSI value.

Loop

```
//==================================== Loop ====================================//
void loop()
{
  Blynk.run();
  timer.run();
}
```

The `loop()` function is another standard Arduino function that continuously runs after the `setup()` function.

- ★ The `Blynk.run()` function is called to handle any incoming Blynk-related communication and events. This allows the ESP32 board to interact with the Blynk app and respond to changes in virtual pins, receive commands, and send data.
- ★ The `timer.run()` function is called to check and execute any scheduled timer events that were set up in the `setup()` function. This ensures that the timer-related tasks, such as blinking LEDs, updating sensor readings, and handling Wi-Fi-related operations, are performed at the specified intervals.
- ★ By including these two function calls within the `loop()`, the program can continuously process Blynk events and run the scheduled timer events as part of its main execution loop. This enables real-time interaction with the Blynk app and ensures the timely execution of the programmed tasks.