# Práctica 5 : Fingerbot usando ESP32

Marlon Sneider Mora Cortes - 20152005034, Sara Valentina Barrero Medina -20191005170
Miguel Ángel Fuentes Ramírez-20182005007
*Ingeniería Electrónica*, *Universidad Distrital "Francisco José de Caldas"*
Bogotá, Colombia

***Resumen*—En este documento muestra el diseño y la implementación de un Fingerbot o Switchbot (dispositivo utilizado para controlar el estado de interruptores mecánicos), para ello se hace uso de un ESP32 como microcontrolador principal, una bateria, un servomotor que acciona el mecanismo, el modulo RTC (Real Time Clock) DS1307 que permite obtener la hora y fecha actual, además se desarrolla una aplicación en Blynk con la cual el usuario puede programar y configurar el dispositivo.**

***Palabras Clave*–**Sistemas embebidos, IoT, Fingerbot, Esp32, WIFI, Blynk, aplicaciones, web.

## I. Introducción

En la actualidad, la implementación de sistemas embebidos se ha convertido en una herramienta fundamental para mejorar y automatizar diversas actividades en la industria y en la vida cotidiana. Los sistemas embebidos son dispositivos electrónicos que se integran en otros productos o sistemas para controlar, monitorizar o automatizar una tarea específica.

El proyecto 5 de la asignatura Sistemas Embebidos 1 propone la elaboración de un Fingerbot el cual es un pequeño robot que permite de forma remota encender o apagar los dispositivos de nuestro hogar que cuenten con un botón convencional de tipo pulsador o de tipo conmutador, como es el caso de algunas lamparas, cafeteras, computadores, impresoras, televisores y todos aquellos dispositivos que pueden ser activados apretando un botón. El consumo de energía del Fingerbot es muy bajo lo que permite su funcionamiento durante largos periodos de tiempo sin intervención humana, además su reducido tamaño y peso permite ser instalado en gran cantidad de dispositivos, en cuanto a la conectividad, algunos son controlados a través de bluetooth y otros de WIFI, en este proyecto el dispositivo es controlado remotamente a través de WIFI haciendo uso de la tarjeta de desarrollo ESP32 y empleando IoT (Internet de las cosas) como medio para configurar las tareas del robot y transmitir los datos de forma remota.

La configuración y el control del Fingerbot se realiza generalmente mediante aplicaciones móviles, en este caso se realiza el diseño en Blynk, la cual es una aplicación que permite crear aplicaciones de IoT de manera sencilla, allí el usuario puede activar de manera remota el mecanismo del robot, programar eventos o tareas y realizar configuraciones.

Durante el proyecto se realiza el diseño y la fabricación de una caja en impresión 3D, en la cual se introducen todos los elementos de hardware mencionados anteriormente y se adapta el mecanismo que hará contacto con el botón.

## II. Formulación del Problema

Se requiere el diseño y la fabricación de un dispositivo tipo Fingerbot que permita el accionamiento de un botón de manera remota, además de una aplicación móvil que permita su configuración.

Algunas consideraciones a tener en cuanta son las siguientes: El dispositivo estará alimentado por una batería de 3.3 V, un servomotor accionara el mecanismo que permite presionar el botón, las tareas de tiempo se llevaran a cabo mediante un modulo RTC (Real Time Clock), el dispositivo estará protegido por una caja plástica, el dispositivo podrá ser controlado y configurado remotamente por WiFi mediante una aplicación móvil.

La aplicación móvil debe permitir al usuario las siguientes acciones:

- Sincronizar el RTC.
- Enviar la orden de apagar o encender el interruptor.
- Configurar el tipo de interruptor a controlar (pulsador o conmutador).
- Seleccionar modo de operación (temporizador o programación semanal) e indicar si el evento es de encendido o apagado.
- Programar múltiples eventos.
- recibir confirmación de que la configuración fue almacenada exitosamente.

Finalmente se requiere que el estado encendido o apagado y los parámetros del modo de operación sean almacenados en la memoria $E^2PROM$ que comparte con el RTC de tal manera que cuando se agote la batería esta información no se pierda y sea ejecutada.

## III. Diseño y Modelo de solución

### A. Hardware

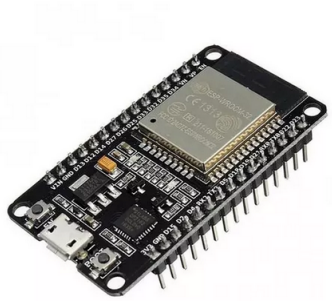

Figure 1: ESP32

*1) ESP32:*

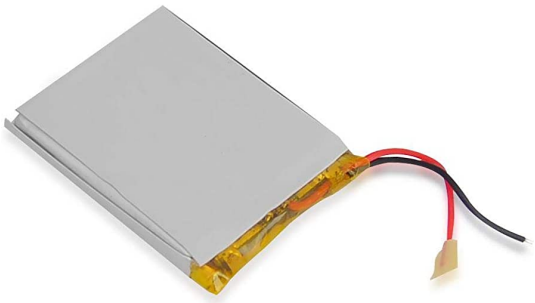

Figure 2: Batería

*2) Batería:*

Figure 3: Servomotor

*3) Servomotor:*

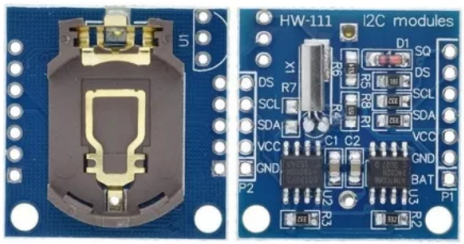


Figure 4: RTC

*4) RTC:* Una vez configurado este modulo RTC permite obtener mediante su interfaz I2C la hora, minutos y segundos actuales, también la fecha en número día, día de la semana, mes y año, tiene un muy bajo consumo, menos de 500 nA en modo respaldo de batería con oscilador en marcha, además posee una memoria EEPROM AT24C32 que permite almacenar 32Kbits (4K Bytes) de datos de manera permanente.

Figure 5: Fingerbot. Elaboración propia

*5) Caja y mecanismo de accionamiento:*

### B. Diseño Aplicación BLYNK

Para el diseño de esta aplicación se tuvieron en cuenta los requerimientos solicitados, por lo que se diseñaron 2 partes, la primera como se aprecia en la figura 6, en donde se puede actualizar la hora del RTC a partir del servidor horario

configurado en Blynk, así como también se dispone de un botón que activa y desactiva la cerradura en tiempo real, en la parte inferior se tienen los elementos dispuestos para la configuración de los eventos tanto de temporizador como de programación, dichos eventos se muestran en la segunda pantalla como se aprecia en las figuras 7 y 8, para confirmar cada una de las acciones se empela el widget disponible en Blynk que permite el envió de notificaciones desde el ESP, en la parte de eventos se tienen 3 botones, el de actualizar permite ver el estado actual delos eventos almacenados en la memoria EEPROM del ESP32, el botón de borrar permite eliminar un evento seleccionado en la tabla, y el de borrar todo permite eliminar todos los elementos almacenados en la memoria, todas estas acciones se realizan utilizando la librera Preferences, la app también permite el envió de una notificación en el momento que se debe de realizar un evento, envía un notificación el tipo de acción realizada (Encendido o Apagado), el tipo de evento (Temporizador o Programación) y el tipo de interruptor (Pulsador o conmutador).


Figure 6: Sección de configuración en la APP


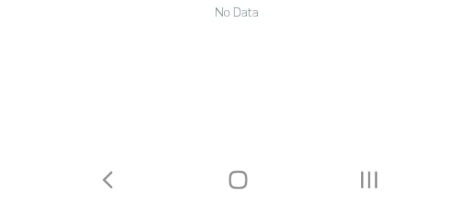


Figure 7: Sección de eventos en la APP


Figure 8: Sección de eventos en la APP

## IV. Resultados

Se desarrollo el siguiente montaje :

Figure 9: Montaje Final

Se puede acceder a la aplicación Bynk desarrollada empleando el siguiente código QR :

Figure 10: QR clonado APP

En la figura 10 se aprecia el montaje final realizado.

## V. Conclusiones

- La implementación del protocolo I2C para la transmisión de datos, en este caso para el control del RTC, facilita la tarea, ya que solo se requieren dos líneas de bus; una línea de datos en serie (SDA) y una línea de reloj en serie SCL, esto resulta muy útil a la hora de incorporar nuevos módulos en un montaje físico.
- Gracias a que esta tarjeta tiene la posibilidad de ser programada en el entorno de desarrollo correspondiente a Arduino IDE el cual utiliza C como lenguaje de programación y nos facilita diversas librerías que son de gran utilidad al momento de usar diferentes características o componentes externos como por ejemplo los sensores empleados, la configuración del WiFi de la tarjeta y el Bluetooth de Baja energía.
- La implementación de un modulo RTC permite desarrollar un sistema IoT en varios modos, ya sea en modo WiFi en donde es posible actualizar la fecha actual del RTC, así como mantener la ejecución de los eventos a pesar de apagar el dispositivo o de cambiar a modo AP.

```
//////////////////////////////////////////
//                WIFI                  //
//////////////////////////////////////////

//Credenciales de la red WiFi a conectarse:
// const char* ssid = "UDNet_Academia";
// const char* password = "6Lj3P_UD_$";


//const char* ssid = "ADM_UDistrital";
//const char* password = "PKFJuurjf986_$";

const char* ssid = "Sara";
const char* password = "1234sara";

//const char* ssid = "Invitados_UDistrital";
//const char* password = "1nv1tad0s"

//const char* ssid = "Familia Ramirez";
//const char* password = "FamiliaRamirezFuentes2020";

// Para reiniciar la memoria :

#include <nvs_flash.h>

// ----------------- AP -------------------

//Credenciales de la red WiFi a crear:
const char* ssid1= "WiFi-ESP32";
const char* password1 = "Embebidos1";
String header;  //Almacena solicitud HTTP

String nombre;
String BotonPresionado;
String BotonPresionado1;
String AccionSelect;
String hora;
String dias;
String checkboxValue;
String cadenas;

char cade3[50];
char* token;
String diasF[7];
String hora1[2];

//----------------- RTC--------------------

#include "RTClib.h"
#include <TimeLib.h>
RTC_DS1307 RTC_DS;

//DateTime time_rtc = RTC_DS.now();
//////////////////////////////////////////
//                BLYNK                 //
//////////////////////////////////////////

/* Comment this out to disable prints and save space */
#define BLYNK_PRINT Serial
#include <WiFi.h>
#include <WiFiClient.h>
#include <WebServer.h>
#include <BlynkSimpleEsp32.h>


WiFiClient client;
WebServer server(80);  //Establecer n mero del puerto del servidor Web asignado al objeto "server"

char auth[] = "K6z43HeYuyQ_4ehqkDHDhEV2oLdPTbxX";  // Put in the Auth Token for the project from
    Blynk. You should receive it in your email.
```

```
70
71 //////////////////////////////////////////
72 //                Variables             //
73 //////////////////////////////////////////
74 //ServoMotor
75 #include <ESP32Servo.h>
76 #define SERVO_PIN 26  // ESP32 pin GIOP26 connected to servo motor
77 Servo servoMotor;
78
79 // ----------------- PINES ----------------
80 String Estado_LED_B = "Off";  //Almacena el estado actual del LED_B
81 String Estado_LED_R = "Off";  //Almacena el estado actual del LED_R
82
83 #define LED_B 2   //Define GPIO2
84 #define LED_R 5   //Define GPIO5
85 #define pinP1 19  //Pulsador de Cambio AP-WIFI
86 // GPIO 22 -- > SCL
87 // GPIO 21 -- > SDA
88
89 //--------------- Pulsadores -----------------
90 volatile uint8_t bandera;
91 volatile uint16_t tiempo;
92 volatile uint32_t rebote;
93
94 //----------- Variables Auxiliares RED----------
95
96 boolean Conexion_WIFI;
97 boolean Conexion_AP;
98 boolean AP, wifi, Cambio_Operacion;
99
100 //----------- Variables Auxiliares RTC----------
101 int year_actual;
102 int month_actual;
103 int day_actual;
104 int hour_actual;
105 int minute_actual;
106 int second_actual;
107
108 int year_evento;
109 int month_evento;
110 int day_evento;
111 int hour_evento;
112 int minute_evento;
113 int second_evento;
114
115 //-------------------- Para WIDGETS---------
116
117 //En caso de ser un evento tipo programacion :
118 String hora_prog;  //Horas en minutos
119 String dias_prog;  //Dias del 1-7
120
121 //En caso de ser un evento tipo temporizador
122 String hora_temp;  //Dias del 1-7
123
124 int Tipo_Estado;       // 0 --> APAGAR, 1 --> ENCENDER
125 int Tipo_Interruptor;  // 0 --> Pulsador, 1 --> Conmutador
126 int Modo_Operacion;    // 0 --> Programacion, 1 --> Temporizador
127
128 // Se emplean en funciones para procesar la informaci n guardada y mostrarla correctamente :
129
130 String DiasSemana[8] = { "", "Lunes", "Martes", "Miercoles", "Jueves", "Viernes", "Sabado", "Domingo
        " };
131 String DiasSeleccionados[7];
132 int dias_elegidos[7];
133
134
135 String Fecha_evento;
136
137 String Hora_evento_temp;
138
139 int Fila_Select, Fila_Deselect;
```

```
String Cantidad_Dias_Programados;

// ------- VARIABLES PARA LA MEMORIA EEPROM -------
#include <Preferences.h>

Preferences preferences;

//Lo que se lee
String Estado_Leida = "";
String Evento_Leida = "";
String Interruptor_Leida = "";
String Fecha_Leida = "";
String Hora_Leida = "";

//Las que se guardan en la memoria

String Estado_Memoria = "";        // ON - OFF
String Evento_Memoria = "";        // Programacion - Temporizador
String Interruptor_Memoria = "";   //Pulsador - Conmutador
String Fecha_Memoria = "";
String Hora_Memoria = "";

int cantidad_eventos = 0;
int pos = 115;

String Fecha_Final_Evento_temporizado;  // La informacion en formato 2023-12-31  24:59:30 de cada evento (YYYY-MM-DD HH:MM:SS)

int Hora_Ejecucion, Minuto_Ejecucion, Segundo_Ejecucion;
int Hora_Ejecucion_2, Minuto_Ejecucion_2, Segundo_Ejecucion_2;

BLYNK_CONNECTED() {  //Sincronizar todo los widgets de la aplicaci n al conectar el ESP32
  Blynk.syncAll();
}

void requestTime() {
  Blynk.sendInternal("rtc", "sync");
}

BLYNK_WRITE(InternalPinRTC) {
  time_t date_blynk = param.asLong();
  Set_RTC(year(date_blynk), month(date_blynk), day(date_blynk), hour(date_blynk), minute(date_blynk), second(date_blynk));  // A o/mes/dia hora/minuto/segundo
  Serial.println("Fecha Actualizada a : " + String(date_blynk));
  Blynk.notify("Fecha Actualizada a : " + String(year(date_blynk)) + "/" + String(month(date_blynk)) + "/" + String(day(date_blynk)) + " " + String(hour(date_blynk)) + ":" + String(minute(date_blynk)) + ":" + String(second(date_blynk)));
}

BLYNK_WRITE(V0) {                 //Genera evento al oprimir el Widget Bot n de actualizar RTC
  int pinValue = param.asInt();  //Leer el valor del pin virtual V1
  if (pinValue == 0) {
    //No se requiere actualizaci n
    Serial.println("Esperando Actualizacion");
  } else {
    Serial.println("Actualizando la Hora ");
    requestTime();
  }
}

BLYNK_WRITE(V5) {                 //Genera evento al oprimir el Widget Bot n de encender el interruptor
  int pinValue = param.asInt();  //Leer el valor del pin virtual V1
  if (pinValue == 0) {
    //No se requiere actualizaci n
    Blynk.notify("Apagando Interruptor");
    //Serial.println("Apagando Interruptor");
    servoMotor.write(0);
  } else {
    Blynk.notify("Encendiendo interruptor");
    //Serial.println("Encendiendo interruptor");
```

```
206     servoMotor.write(pos);
207   }
208 }
209
210 BLYNK_WRITE(V3) {
211   switch (param.asInt()) {
212     case 1:
213       {  // Pulsador
214         Tipo_Interruptor = 0;
215         break;
216       }
217     case 2:
218       {  // Conmutador
219         Tipo_Interruptor = 1;
220
221         break;
222       }
223   }
224   Serial.println("V3 :" + String(param.asInt()));
225 }
226
227 BLYNK_WRITE(V4) {
228   switch (param.asInt()) {
229     case 1:
230       {  // Programacion
231         Modo_Operacion = 0;
232
233         break;
234       }
235     case 2:
236       {  // Temporizador
237         Modo_Operacion = 1;
238
239         break;
240       }
241   }
242   Serial.println("V4 :" + String(param.asInt()));
243 }
244
245 BLYNK_WRITE(V10) {
246   switch (param.asInt()) {
247     case 1:
248       {                     // Item 1
249         Tipo_Estado = 0;   // APAGAR
250
251         break;
252       }
253     case 2:
254       {                     // Item 2
255         Tipo_Estado = 1;   // ENCENDER
256         break;
257       }
258   }
259
260   Serial.println("V10 :" + String(param.asInt()));
261 }
262
263
264 BLYNK_WRITE(V6) {
265   hora_temp = param[0].asStr();  //Almacena la hora de 0 - 86400
266   //Para obtener el tiempo de temporizador se debe comparar con la hora actual del RTC
267   //Procesar_Hora(hora_temp, true);
268   double* datos_fecha = Procesar_Hora(hora_temp);
269   Serial.println("Horas :" + String(datos_fecha[0]) + " Minutos : " + String(datos_fecha[1]));
270   Calcular_Fecha(datos_fecha[0], datos_fecha[1]);
271   Serial.println(Hora_evento_temp);
272   Serial.println("Temporizador para  : ");
273   Serial.println(String(day_evento) + "/" + String(month_evento) + "/" + String(year_evento) + " " +
        String(hour_evento) + ":" + String(minute_evento) + ":" + String(second_evento));
274   //Serial.println(horas_temp); //En cuantas horas se debe ejecutar el evento desde su programacion
```

```
  //Serial.println(minutos_temp);//En cuantos minutos se debe ejecutar el evento desde su
    programacion
}



BLYNK_WRITE(V7) {
  // En este apartado se obtiene la fecha y hora  para un evento de tipo programacion

  hora_prog = param[0].asStr();  //Horas en segundos
  dias_prog = param[3].asStr();  // 1 = Lunes
  Procesar_Fecha(dias_prog);
  dias_prog = Cantidad_Dias_Programados + "," + dias_prog;
  Serial.println("Dias Programados : " + String(dias_prog));
  Serial.println("Hora Programada : " + String(hora_prog));
}

BLYNK_WRITE(V8) {  //Genera evento al oprimir el Widget Bot n de guardar un evento

  int pinValue = param.asInt();  //Leer el valor del pin virtual V1
  if (pinValue == 1) {
    Estado_Leida = "";
    Evento_Leida = "";
    Interruptor_Leida = "";
    Fecha_Leida = "";
    Hora_Leida = "";
    Estado_Memoria = "";        // ON - OFF
    Evento_Memoria = "";        // Programacion - Temporizador
    Interruptor_Memoria = "";  //Pulsador - Conmutador
    Fecha_Memoria = "";
    Hora_Memoria = "";
    Serial.println("Caracteristicas : ");
    Lectura_Memoria();
    Procesar_Datos_Guardar();
    Guardado_Memoria();
    Lectura_Memoria();
    Blynk.notify("Evento guardado");
    Serial.println("++++++++++++++++++++++ Evento guardado ++++++++++++++++++++++++ ");
    Serial.println("Estado :" + Estado_Leida);
    Serial.println("Evento :" + Evento_Leida);
    Serial.println("Interruptor :" + Interruptor_Leida);
    Serial.println("Fecha :" + Fecha_Leida);
    Serial.println("Hora :" + Hora_Leida);
    Serial.println("Eventos Guardados :" + String(cantidad_eventos));
    Serial.println("++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++ ");
  }
}

BLYNK_WRITE(V9) {
  String cmd = param[0].asStr();
  if (cmd == "select") {
    //row in table was deselected.

    Fila_Deselect = param[1].asInt();

    Serial.println("Fila Deseleccionada :" + String(Fila_Deselect));
  }
  if (cmd == "deselect") {

    //row in table was selected.
    Fila_Select = param[1].asInt();

    Serial.println("Fila Seleccionada :" + String(Fila_Select));
  }
}



BLYNK_WRITE(V11) {  //Genera evento al oprimir el Widget Bot n de actualizar la tabla

```

```
345   int pinValue = param.asInt();  //Leer el valor del pin virtual V11
346   if (pinValue == 1) {
347
348     //Serial.println("Actualizando Tabla : ");
349     Lectura_Memoria();
350     Blynk.virtualWrite(V9, "clr");
351     Procesar_Datos_Graficar();
352     Blynk.notify("Tabla Actualizada");
353   }
354 }
355
356
357 BLYNK_WRITE(V13) {  //Genera evento al oprimir el Widget Bot n de borrar un evento
358
359   int pinValue = param.asInt();  //Leer el valor del pin virtual V13
360   if (pinValue == 1) {
361     Serial.println("Eliminar Evento " + Fila_Select);
362     Blynk.virtualWrite(V9, "clr");
363
364     Estado_Leida = "";
365     Evento_Leida = "";
366     Interruptor_Leida = "";
367     Fecha_Leida = "";
368     Hora_Leida = "";
369     Estado_Memoria = "";        // ON - OFF
370     Evento_Memoria = "";        // Programacion - Temporizador
371     Interruptor_Memoria = "";   //Pulsador - Conmutador
372     Fecha_Memoria = "";
373     Hora_Memoria = "";
374
375     Lectura_Memoria();
376     Procesar_Datos_Eliminar(Fila_Select);
377     Procesar_Datos_Graficar();
378     Blynk.notify("Evento Eliminado");
379   }
380 }
381
382
383
384
385
386
387
388
389 BLYNK_WRITE(V14) {  //Genera evento al oprimir el Widget Bot n de borrar todos los eventos
390
391   int pinValue = param.asInt();  //Leer el valor del pin virtual V14
392   if (pinValue == 1) {
393     Blynk.virtualWrite(V9, "clr");
394     nvs_flash_erase();  // erase the NVS partition and...
395     nvs_flash_init();   // initialize the NVS partition.
396     Blynk.notify("Se han eliminado todos los evento, se reiniciara el sistema ");
397     Serial.println("Borrada Memoria EEPROM ");
398     // Restart ESP
399     ESP.restart();
400   }
401 }
402
403 //------------------ INTERRUPCIONES ------------
404 void IRAM_ATTR Int_P1() {
405   Cambio_Operacion = true;
406   digitalWrite(LED_B, !digitalRead(LED_B));
407   if (Conexion_WIFI) {
408     //Configuracion_AP();
409     Serial.println("Cambiando a AP");
410     AP = true;
411     wifi = false;
412
413   } else {
414     AP = false;
415     wifi = true;
```

```
    Serial.println("Cambiando a WIFI");
  }
  if (tiempo == 250) {
    tiempo = 100;
  } else {
    tiempo = 250;
  }
  bandera = 1;        //Activa bandera para indicar ingreso a rutina de interrupci n
  rebote = millis();  //Lee el valor actual de la funci n millis()
  Serial.println("Tiempo= " + (String)tiempo + "ms");
  Serial.println(rebote);
  detachInterrupt(digitalPinToInterrupt(pinP1));  //Deshabilita int.
}
//////////////////////////////////////////
//                SETUP                 //
//////////////////////////////////////////


void setup() {
  Serial.begin(115200);
  pinMode(LED_B, OUTPUT);
  pinMode(LED_R, OUTPUT);
  digitalWrite(LED_B, LOW);
  digitalWrite(LED_R, LOW);
  pinMode(pinP1, INPUT_PULLUP);                                    //Configurar GPIO16 como entrada para leer estado de pulsador N.A.
  attachInterrupt(digitalPinToInterrupt(pinP1), Int_P1, FALLING);  //Habilita int. generada por el GPIO16

  preferences.begin("Estado", false);        // ON - OFF
  preferences.begin("Evento", false);        // Programacion - Temporizador
  preferences.begin("Interruptor", false);   //Pulsador - Conmutador
  preferences.begin("Fecha", false);
  preferences.begin("Hora", false);
  preferences.begin("cantidadeventos", false);

  tiempo = 250;
  bandera = 0;
  //Por defecto se inicia la conexi n WIFI y en blynk
  //Conexion_RedWifi();
  Config_Blynk();
  Config_RTC();

  servoMotor.attach(SERVO_PIN);  // attaches the servo on ESP32 pin
  servoMotor.write(0);
}

//////////////////////////////////////////
//                 LOOP                 //
//////////////////////////////////////////
void loop() {
  Pulsador1_();

  if (AP and Cambio_Operacion) {
    WiFi.disconnect();
    Configuracion_AP();
    Cambio_Operacion = false;
  } else if (wifi and Cambio_Operacion) {
    // Conexion_RedWifi();
    Config_Blynk();
    Cambio_Operacion = false;
  }
  if (Conexion_AP) {
    Cliente_AP();
  }
  if (Conexion_WIFI) {
    Blynk.run();
    //Blynk.virtualWrite(V9, "add", 1, "Temperature C", Fecha_evento);
  }
  //Serial.println("Fecha Ejecucion  : ");
```

```
484   //Fecha_Final_Evento_temporizado = String(year_evento)+ "-"+ String(month_evento) +"-"+ String(
        day_evento)+ " " + String(hour_evento) + ":" + String(minute_evento) + ":" + String(
        second_evento);
485
486   Ejecucion_Eventos();
487 }
488
489
490 void Ejecucion_Eventos() {
491   int Estado, Evento, Interruptor;
492   String Fecha, Hora;
493   char Datos[500];
494   int id = 0;
495   Lectura_Memoria();
496
497   for (int i = 0; i < cantidad_eventos; i++) {
498     //Imprimir todos los elementos con la informacion de id
499     Estado = Separar(Estado_Leida, ";", true, i).toInt();
500     Evento = Separar(Evento_Leida, ";", true, i).toInt();
501     Interruptor = Separar(Interruptor_Leida, ";", true, i).toInt();
502     Fecha = Separar(Fecha_Leida, ";", true, i);
503     Hora = Separar(Hora_Leida, ";", true, i);
504     String Tipo;
505     if (Interruptor == 0) {
506       Tipo = "Pulsador";
507     } else {
508       Tipo = "Conmutador";
509     }
510
511     if (Evento == 0) {
512       Fecha_Actual();
513       //Evento Tipo Programaci n
514       //1.Verificar que dia es hoy
515       int num_dia_actual = RTC_DS.now().dayOfTheWeek();  // contiene  un numero de 1-7 con la
        informacion del dia actual
516       String Dia_Actual = DiasSemana[num_dia_actual];
517       //Serial.println("Hoy es : " + String(Dia_Actual));
518
519       //2. Extraer que dias estan programados, en la variable Fecha se almacenan de la forma
        numero_dias,1,2,3,4
520       int Dias_programados = Separar(Fecha, ",", true, 0).toInt();
521       int dia_programado;
522       //Serial.println("Informacion completa :" + Fecha);
523       //Serial.println("Dias Programados " + String(Dias_programados));
524
525       //3.Verificar si hoy se tienen eventos programados:
526       for (int i = 1; i <= Dias_programados; i++) {
527         dia_programado = Separar(Fecha, ",", true, i).toInt();
528         if (num_dia_actual == dia_programado) {
529           //4.Verificar la hora y minutos de estos eventos porogramados
530           double* datos_fecha = Procesar_Hora_segundos(String(Hora));
531           int hora_programada = int(datos_fecha[0]);
532           int minuto_pogramado = int(datos_fecha[1]);
533
534           if (hora_programada == hour_actual) {
535             //Se esta en la misma hora del evento programado
536             Serial.println("Programado para  " + String(hora_programada) + " : " + String(
        minuto_pogramado));
537             if (minuto_pogramado == minute_actual) {
538               //Se esta en el mismo minito del evento
539               if (second_actual == 0) {
540
541                 //EJECUTAR EVENTO
542                 if (Estado == 0) {  //Apagar
543                   Blynk.notify("Interruptor tipo " + Tipo + " Apagado por programaci n");
544                   Serial.println("Apagando Interruptor");
545                   Blynk.virtualWrite(V9, "clr");
546                   servoMotor.write(0);
547
548                 } else {
549                   Blynk.notify("Interruptor tipo " + Tipo + " Encendido por programaci n");
```

```
                    Blynk.virtualWrite(V9, "clr");
                    servoMotor.write(pos);
                  }
                }
              }
            } else {
              Serial.println("Evento programado para hoy " + String(Dia_Actual));
              Serial.println("Programado para  " + String(hora_programada) + " : " + String(
    minuto_pogramado));
            }

            //Serial.println("Posicion " + String(i) + " : " + dia_programado);
          }
        }
        Serial.println(".....................................................");

      } else {
        //Evento Tipo Temporizador
        //Serial.println("-----------------------------------------------------------------");
        int Hora_Temporizador = Separar(String(Hora), ",", true, 0).toInt();  // La hora en la que se
    debe ejecutar el evento
        double* datos_fecha = Procesar_Hora_segundos(String(Hora_Temporizador));
        //Serial.println("DATE :" + String((int)datos_fecha[0]) + ":" + String((int)datos_fecha[1]) +
    ":" + String((int)datos_fecha[2]));
        String cadena = Calcular_diferenciaHoraria_2(datos_fecha[0], datos_fecha[1], datos_fecha[2]);
         //Se obtiene cuantas horas y minutos  faltan para realizar el evento

        if (Hora_Ejecucion_2 == 0) {
          if (Minuto_Ejecucion_2 == 0) {
            if (Segundo_Ejecucion_2 == 0) {
              if (Estado == 0) {  //Apagar
                Blynk.notify("Interruptor tipo " + Tipo + " Apagado Luego de un temporizador");
                Serial.println("Apagando Interruptor");
                Blynk.virtualWrite(V9, "clr");
                servoMotor.write(0);
                Procesar_Datos_Eliminar(i);
                Procesar_Datos_Graficar();
              } else {
                Blynk.notify("Interruptor tipo " + Tipo + " Encendido luego de un temporizador");
                Serial.println("Apagando Interruptor");
                Blynk.virtualWrite(V9, "clr");
                servoMotor.write(pos);
                Procesar_Datos_Eliminar(i);
                Procesar_Datos_Graficar();
              }
            }
          }
        }
      }
    }
  }
}
void Procesar_Fecha(String dias) {
  int j = 0;
  for (int i = 0; i < dias.length(); i++) {
    if (dias[i] != ',') {
      int valor = dias[i] - '0';
      DiasSeleccionados[j] = DiasSemana[valor];
      dias_elegidos[j] = valor;
      //Serial.println(DiasSeleccionados[j]);
      //Serial.println(dias_elegidos[j]);
      j++;
    }
  }
  Cantidad_Dias_Programados = String(j);
  //Fecha_evento = String(j); // Se almacena la cantidad de dias que se repite el evento
  if (j == 7) {
    Fecha_evento = Fecha_evento + "," + "Todos los dias";
  } else {
    for (int i = 0; i < j; i++) {
      //Serial.println(DiasSeleccionados[i]);
      Fecha_evento = Fecha_evento + "," + DiasSeleccionados[i];
```

```
    }
  }
}

String Calcular_diferenciaHoraria_2(double horas_temp, double minutos_temp, double segundos_temp) {
  Fecha_Actual();

  /*
    En este punto las variables horas_temp,minutos_temp,segundos_temp  tienen la hora exacta en la
    que se debe realizar el evento
  */
  Serial.println("Hora actual :" + String(hour_actual) + ":" + String(minute_actual) + ":" + String(
    second_actual));
  Serial.println("Horas de Temporizador :" + String((int)horas_temp) + ":" + String((int)
    minutos_temp) + ":" + String(segundos_temp));

  int resta = horas_temp - hour_actual;
  int hora_evento, minuto_evento, segundo_evento;

  if (horas_temp > hour_actual) {
    hora_evento = horas_temp - hour_actual;
  } else {
    hora_evento = (24 - hour_actual) + horas_temp;
  }

  if (minutos_temp > second_actual) {
    minuto_evento = minutos_temp - minute_actual;
  } else {
    hora_evento = hora_evento - 1;
    minuto_evento = (60 - minute_actual) + minutos_temp;
  }

  if (segundos_temp > second_actual) {
    segundo_evento = segundos_temp - second_actual;
  } else {
    minuto_evento = minuto_evento - 1;
    segundo_evento = (60 - second_actual) + segundos_temp;
  }


  //************************************************************
  if (minute_actual == minutos_temp) {
    Minuto_Ejecucion_2 = 0;
    minuto_evento = 0;
  } else {
    Minuto_Ejecucion_2 = minuto_evento;
  }
  if (horas_temp == hour_actual) {
    if (minute_actual > minutos_temp) {
      //Se tiene un temporizador de 24 hrs
      Hora_Ejecucion_2 = 23;
      hora_evento = 23;
    } else {
      Hora_Ejecucion_2 = 0;
      hora_evento = 0;
    }

  } else {
    Hora_Ejecucion_2 = hora_evento;
  }
  if (second_actual == segundos_temp) {
    Segundo_Ejecucion_2 = 0;
    segundo_evento = 0;
  } else {
    Segundo_Ejecucion_2 = segundo_evento;
  }


  String cadena = "en " + String(hora_evento) + " Horas  " + String(minuto_evento) + " minutos y " +
     String(segundo_evento) + " seg";
  return cadena;
```

```
}

void Procesar_Datos_Eliminar(int Fila) {

  String Fecha, Hora, Estado, Evento, Interruptor;

  Estado_Leida = "";
  Evento_Leida = "";
  Interruptor_Leida = "";
  Fecha_Leida = "";
  Hora_Leida = "";
  Estado_Memoria = "";        // ON - OFF
  Evento_Memoria = "";        // Programacion - Temporizador
  Interruptor_Memoria = "";   //Pulsador - Conmutador
  Fecha_Memoria = "";
  Hora_Memoria = "";

  Lectura_Memoria();
  bool corrimiento = false;
  bool previo = true;
  int contador = cantidad_eventos;
  for (int i = 0; i < cantidad_eventos; i++) {
    //Leer todos los elementos con la informacion de id

    if (i == Fila) {
      //Posicion que se desea desplazar
      Serial.println("Eliminar la fila " + String(i));
      cantidad_eventos--;
      i++;
    }
    Estado = Separar(Estado_Leida, ";", true, i);
    Evento = Separar(Evento_Leida, ";", true, i);
    Interruptor = Separar(Interruptor_Leida, ";", true, i);
    Fecha = Separar(Fecha_Leida, ";", true, i);
    Hora = Separar(Hora_Leida, ";", true, i);


    Estado_Memoria = Estado_Memoria + ";" + Estado;
    Evento_Memoria = Evento_Memoria + ";" + Evento;
    Interruptor_Memoria = Interruptor_Memoria + ";" + Interruptor;
    Fecha_Memoria = Fecha_Memoria + ";" + Fecha;
    Hora_Memoria = Hora_Memoria + ";" + Hora;
  }

  Guardado_Memoria();

  Estado_Memoria = "";        // ON - OFF
  Evento_Memoria = "";        // Programacion - Temporizador
  Interruptor_Memoria = "";   //Pulsador - Conmutador
  Fecha_Memoria = "";
  Hora_Memoria = "";

  Estado = "";
  Evento = "";
  Interruptor = "";
  Fecha = "";
  Hora = "";

  Lectura_Memoria();
}



void Procesar_Datos_Graficar() {
  int Estado, Evento, Interruptor;
  String Fecha, Hora;
  char Datos[500];
  int id = 0;

  for (int i = 0; i < cantidad_eventos; i++) {
    //Imprimir todos los elementos con la informacion de id
```

```
    Estado = Separar(Estado_Leida, ";", true, i).toInt();
    Serial.print("Estado " + String(i) + ": ");
    Serial.println(String(Estado));
    Evento = Separar(Evento_Leida, ";", true, i).toInt();
    Serial.print("Evento " + String(i) + ": ");
    Serial.println(String(Evento));
    Interruptor = Separar(Interruptor_Leida, ";", true, i).toInt();
    Serial.print("Interruptor " + String(i) + ": ");
    Serial.println(String(Interruptor));
    Fecha = Separar(Fecha_Leida, ";", true, i);
    Serial.print("Fecha " + String(i) + ": ");
    Serial.println(Fecha);
    Hora = Separar(Hora_Leida, ";", true, i);
    Serial.print("Hora " + String(i) + ": ");
    Serial.println(String(Hora));
    String Tipo;
    if (Estado == 0) {
      Tipo = "Apagar";
    } else {
      Tipo = "Encender";
    }

    if (Evento == 0) {
      //Evento Tipo Programaci n
      Fecha_evento = "";
      Plot_Fecha(Fecha);                          //Fecha_evento Contiene los dias que se debe
    ejecutar
      double* datos_fecha = Procesar_Hora(Hora);  //Contiene la Hora procesada en formato
      String cadena = Fecha_evento + " a las " + String(Hora_evento_temp);
      Blynk.virtualWrite(V9, "add", i, cadena, Tipo);

    } else {
      //Evento Tipo Temporizador
      Serial.println("---------------------- Procesar_Datos_Graficar ------------------------");
      int Hora_Temporizador = Separar(String(Hora), ",", true, 0).toInt();  // La hora en la que se
    debe ejecutar el evento
      Serial.print("Hora de Temporizacion : ");
      Serial.println(String(Hora_Temporizador));
      //double* datos_fecha = Procesar_Hora(String(Hora_Temporizador));
      double* datos_fecha = Procesar_Hora_segundos(String(Hora_Temporizador));
      Serial.println("DATE :" + String((int)datos_fecha[0]) + ":" + String((int)datos_fecha[1]) + ":
    " + String((int)datos_fecha[2]));
      String cadena = Calcular_diferenciaHoraria_2(datos_fecha[0], datos_fecha[1], datos_fecha[2]);
     //Se obtiene cuantas horas y minutos  faltan para realizar el evento
      Blynk.virtualWrite(V9, "add", i, cadena, Tipo);
    }
  }
}

void Plot_Fecha(String dias) {
  int j = 0;
  Fecha_evento = "";
  int Dias_programados = Separar(dias, ",", true, 0).toInt();
  for (int i = 1; i <= Dias_programados; i++) {
    int valor = Separar(dias, ",", true, i).toInt();
    DiasSeleccionados[j] = DiasSemana[valor];
    j++;
    //Serial.println("Posicion " + String(i) + " : " + Separar(Fecha, ",", true, i));
  }
  if (j == 7) {
    Fecha_evento = "Todos los dias";
  } else {
    for (int i = 0; i < j; i++) {
      //Serial.println(DiasSeleccionados[i]);
      if (Fecha_evento == "") {
        Fecha_evento = DiasSeleccionados[i];
      } else {
        Fecha_evento = Fecha_evento + "," + DiasSeleccionados[i];
      }
    }
  }
```

```
822 }
823
824 String Calcular_diferenciaHoraria(double horas_temp, double minutos_temp, double segundos_temp) {
825   Fecha_Actual();
826
827   /*
828     En este punto las variables horas_temp,minutos_temp,segundos_temp  tienen la hora exacta en la
        que se debe realizar el evento
829   */
830   //Serial.println("Hora actual :" + String(hour_actual) + ":" + String(minute_actual) + " " +
        String(tiempo_actual));
831   //Serial.println("Horas de Temporizador :" + String((int)horas_temp) + ":" + String((int)
        minutos_temp) + " " + String(tiempo_timer));
832
833   int resta = horas_temp - hour_actual;
834   int hora_evento, minuto_evento, segundo_evento;
835
836   if (horas_temp > hour_actual) {
837     hora_evento = horas_temp - hour_actual;
838   } else {
839     hora_evento = (24 - hour_actual) + horas_temp;
840   }
841
842   if (minutos_temp > second_actual) {
843     minuto_evento = minutos_temp - minute_actual;
844   } else {
845     hora_evento = hora_evento - 1;
846     minuto_evento = (60 - minute_actual) + minutos_temp;
847   }
848
849   if (segundos_temp > second_actual) {
850     segundo_evento = segundos_temp - second_actual;
851   } else {
852     minuto_evento = minuto_evento - 1;
853     segundo_evento = (60 - second_actual) + segundos_temp;
854   }
855
856
857   //************************************************************
858   if (minute_actual == minutos_temp) {
859     Minuto_Ejecucion = 0;
860     minuto_evento = 0;
861   } else {
862     Minuto_Ejecucion = minuto_evento;
863   }
864   if (horas_temp == hour_actual) {
865     if (minute_actual > minutos_temp) {
866       //Se tiene un temporizador de 24 hrs
867       Hora_Ejecucion = 23;
868       hora_evento = 23;
869     } else {
870       Hora_Ejecucion = 0;
871       hora_evento = 0;
872     }
873
874   } else {
875     Hora_Ejecucion = hora_evento;
876   }
877   if (second_actual == segundos_temp) {
878     Segundo_Ejecucion = 0;
879     segundo_evento = 0;
880   } else {
881     Segundo_Ejecucion = segundo_evento;
882   }
883
884
885   String cadena = "en " + String(hora_evento) + " Horas  " + String(minuto_evento) + " minutos y " +
        String(segundo_evento) + " seg";
886   return cadena;
887 }
888
```

```
889
890 String Separar(String cadena, char Separador[], bool Editar, int identificador) {
891   char Datos[500];
892   cadena.toCharArray(Datos, 500);
893   char* token = strtok(Datos, Separador);
894   String Cadena_separada;
895   int id = 0;
896   while (token != NULL) {
897     Cadena_separada = String(token);
898     //Serial.print("Registro func " + String(id) + ": ");
899     //Serial.println(Cadena_separada);
900     token = strtok(NULL, Separador);
901     if (Editar) {
902       if (id == identificador) {
903         id = 0;
904         token = "";
905         break;
906       }
907       id++;
908     }
909   }
910   return Cadena_separada;
911 }
912 void Procesar_Datos_Guardar() {
913   /*
914     Se almacenan 5 variables tipo String, cada evento nuevo se separa con un ;
915     Estado_Memoria : Tipo_Estado : un 0 representa se debe apagar el interruptor , un 1 encenderlo
916     Evento_Memoria : Modo_Operacion : un 0 representa que es un tipo de evento en modo programacion,
        un 1 representa que es un tipo de evento en modo temporizador
917     Interruptor_Memoria : Tipo_Interruptor : un 0 representa que es un interruptor tipo Pulsador, un
        1 representa que es un interruptor tipo conmutador
918     Fecha_Memoria :
919
920     En caso de ser un evento programado se almacenara 1 variable extraida de la app en el siguiente
        formato:
921       year_evento,month_evento,day_evento
922
923     En caso de ser un evento temporizado se almacenar 1 variable extraida de la app en el siguiente
        formato:
924       dias_prog --> 1,2,3,4,5,6,7 --> Lunes = 1 ...
925
926     Hora_Memoria :
927       procesarhoratemp =  hour_evento*3600+minute_evento*60+second_evento; // En caso de ser un
        temporizador
928
929      hora_prog --> es un valor numerico que representa la hora en segundos 0 - 24H* 3600 --> 0-86400
930   */
931   Estado_Memoria = "";        // ON - OFF
932   Evento_Memoria = "";        // Programacion - Temporizador
933   Interruptor_Memoria = "";   //Pulsador - Conmutador
934   Fecha_Memoria = "";
935   Hora_Memoria = "";
936
937   long procesarhoratemp = hour_evento * 3600 + minute_evento * 60 + second_evento;
938
939   //Proceso de adicion de evento :
940   cantidad_eventos++;
941   Estado_Memoria = Estado_Leida + ";" + String(Tipo_Estado);
942   Evento_Memoria = Evento_Leida + ";" + String(Modo_Operacion);
943   Interruptor_Memoria = Interruptor_Leida + ";" + String(Tipo_Interruptor);
944
945   if (Modo_Operacion == 0) {
946     // Programacion
947     //Almacenar info en modo Programacion
948     Fecha_Memoria = Fecha_Leida + ";" + String(dias_prog);
949     Hora_Memoria = Hora_Leida + ";" + String(hora_prog);
950   } else {
951     //Almacenar info en modo Temporizador
952     //year_evento,month_evento,day_evento
953     Fecha_Memoria = Fecha_Leida + ";" + String(year_evento) + "," + String(month_evento) + "," +
        String(day_evento);
```

```
    Hora_Memoria = Hora_Leida + ";" + String(procesarhoratemp) + "," + String(hora_temp);
  }
}


void Lectura_Memoria() {

  //1. Se requiere extaer los datos ya almacenados, respetar su posison y agregar un nuevo evento al final
  Estado_Leida = preferences.getString("Estado", "");
  Evento_Leida = preferences.getString("Evento", "");
  Interruptor_Leida = preferences.getString("Interruptor", "");
  Fecha_Leida = preferences.getString("Fecha", "");
  Hora_Leida = preferences.getString("Hora", "");
  cantidad_eventos = preferences.getInt("cantidadeventos", 0);
  /*
    Serial.println("----------------Datos Leidos:--------------------");
    Serial.println("Estado :" + Estado_Leida);
    Serial.println("Evento :" + Evento_Leida);
    Serial.println("Interruptor :" + Interruptor_Leida);
    Serial.println("Fecha :" + Fecha_Leida);
    Serial.println("Hora :" + Hora_Leida);
    Serial.println("Eventos Guardados :" + String(cantidad_eventos));
    Serial.println("----------------Fin Datos Leidos:--------------------");*/
}
void Guardado_Memoria() {
  //Almacenar

  if (cantidad_eventos == 0) {
    Estado_Memoria = "";        // ON - OFF
    Evento_Memoria = "";        // Programacion - Temporizador
    Interruptor_Memoria = "";   //Pulsador - Conmutador
    Fecha_Memoria = "";
    Hora_Memoria = "";
  }
  preferences.putString("Estado", Estado_Memoria);
  preferences.putString("Evento", Evento_Memoria);
  preferences.putString("Interruptor", Interruptor_Memoria);
  preferences.putString("Fecha", Fecha_Memoria);
  preferences.putString("Hora", Hora_Memoria);
  preferences.putInt("cantidadeventos", cantidad_eventos);

  Serial.println("Memoria actualizada.");
  //Cerrar lectura de memoria
  //preferences.end();
}

void Fecha_Actual() {
  //Obtener la fecha actual
  year_actual = RTC_DS.now().year();
  month_actual = RTC_DS.now().month();
  day_actual = RTC_DS.now().day();
  hour_actual = RTC_DS.now().hour();
  minute_actual = RTC_DS.now().minute();
  second_actual = RTC_DS.now().second();
}

void Calcular_Fecha(double horas_temp, double minutos_temp) {
  Fecha_Actual();
  //Serial.println("Hora actual :" + String(hour_actual) + ":" + String(minute_actual));
  //Serial.println("Horas de Temporizador :" + String((int)horas_temp) + ":" + String((int)minutos_temp));
  int horas_timer = hour_actual + (int)horas_temp;
  int minutos_timer = minute_actual + (int)minutos_temp;

  second_evento = second_actual;
  if (minutos_timer >= 60) {
    horas_timer = horas_timer + 1;
    minutos_timer = minutos_timer - 60;
    hour_evento = horas_timer;
    minute_evento = minutos_timer;
```

```
1023    } else {
1024      hour_evento = horas_timer;
1025      minute_evento = minutos_timer;
1026    }
1027
1028    if (horas_timer >= 24) {
1029      day_evento = day_actual + 1;
1030      hour_evento = horas_timer - 24;
1031    } else {
1032      day_evento = day_actual;
1033    }
1034
1035    if (month_actual == 1 || month_actual == 3 || month_actual == 5 || month_actual == 7 ||
        month_actual == 8 || month_actual == 10 || month_actual == 12) {
1036      if (day_evento > 31) {
1037        month_evento = month_actual + 1;
1038      } else {
1039        month_evento = month_actual;
1040      }
1041    } else if (month_actual = 2) {
1042      if (day_evento > 28) {
1043        month_evento = month_actual + 1;
1044      } else {
1045        month_evento = month_actual;
1046      }
1047    } else {
1048      if (day_evento > 30) {
1049        month_evento = month_actual + 1;
1050      } else {
1051        month_evento = month_actual;
1052      }
1053    }
1054    if (month_evento > 12) {
1055      year_evento = year_actual + 1;
1056    } else {
1057      year_evento = year_actual;
1058    }
1059  }
1060
1061
1062  double* Procesar_Hora(String hora) {
1063    static double values[2];  // En primera posicion las horas en segunda posicion los minutos
1064
1065    double horas_convertidas, minutos_convertidos;
1066    float horas_f_temp = (hora.toFloat()) / 3600;                    // Hora en formato militar
1067    minutos_convertidos = modf(horas_f_temp, &horas_convertidas);  // mandar apuntador de parteEntera
1068    minutos_convertidos = minutos_convertidos * 60;                  //minutos 0 - 60
1069    Hora_evento_temp = String((int)horas_convertidas) + ":" + String(int(round(minutos_convertidos)));
1070    //Serial.println("Hora Procesada =  " + Hora_evento_temp);
1071
1072    values[0] = horas_convertidas;
1073    values[1] = minutos_convertidos;
1074    return values;
1075  }
1076
1077  double* Procesar_Hora_segundos(String hora) {
1078    static double values[3];  // En primera posicion las horas en segunda posicion los minutos
1079
1080    double horas_convertidas, minutos_convertidos, segundos_convertidos;
1081    float horas_f_temp = (hora.toFloat()) / 3600;                    // Hora en formato militar
1082    minutos_convertidos = modf(horas_f_temp, &horas_convertidas);  // mandar apuntador de parteEntera
1083    minutos_convertidos = minutos_convertidos * 60;                  //minutos 0 - 60
1084
1085    segundos_convertidos = modf(minutos_convertidos, &minutos_convertidos);  // mandar apuntador de
        parteEntera
1086    segundos_convertidos = segundos_convertidos * 60;                        //segundos 0 - 60
1087    Hora_evento_temp = String((int)horas_convertidas) + ":" + String((int)minutos_convertidos) + ":" +
         String((int)segundos_convertidos);
1088    //Serial.println("Hora Procesada =  " + Hora_evento_temp);
1089
1090    values[0] = horas_convertidas;
```

```
  values[1] = minutos_convertidos;
  values[2] = segundos_convertidos;
  return values;
}



void Config_Blynk() {
  Conexion_WIFI = true;
  Conexion_AP = false;
  Blynk.begin(auth, ssid, password, "iot.laserud.co", 8080);
  Serial.println("Conectando a Blynk usando la red  ");
  Serial.println(ssid);
  while (Blynk.connect() == false) {
    Serial.print(".");
  }
  Serial.println("WiFi conectado a Blynk");
  digitalWrite(LED_B, LOW);
}
void Config_RTC() {
  if (!RTC_DS.begin()) {
    Serial.println("Couldn't find RTC");
    Serial.flush();
    while (1) delay(10);
  } else {
    Serial.println("RTC running");
  }
}

void Set_RTC(int Year, int Month, int Day, int Hour, int Minute, int Second) {
  Serial.println("DS1307 actualizado con la fecha y hora ingresada :" + String(Year) + "/" + String(
    Month) + "/" + String(Day) + " " + String(Hour) + ":" + String(Minute) + ":" + String(Second));
  RTC_DS.adjust(DateTime(Year, Month, Day, Hour, Minute, Second));  // A o/mes/dia hora/minuto/
    segundo
  //Serial.println (RTC_DS.now().timestamp(DateTime::TIMESTAMP_DATE));
  Serial.print(RTC_DS.now().year(), DEC);
  Serial.print('/');
  Serial.print(RTC_DS.now().month(), DEC);
  Serial.print('/');
  Serial.print(RTC_DS.now().day(), DEC);
  Serial.print(' ');
  Serial.print(RTC_DS.now().hour(), DEC);
  Serial.print(':');
  Serial.print(RTC_DS.now().minute(), DEC);
  Serial.print(':');
  Serial.print(RTC_DS.now().second(), DEC);
  Serial.println();
  // Actualizar la APP de Blynk
  String currentTime = String(RTC_DS.now().hour()) + ":" + RTC_DS.now().minute() + ":" + RTC_DS.now
    ().second();
  String currentDate = String(RTC_DS.now().day()) + " " + RTC_DS.now().month() + " " + RTC_DS.now().
    year();
  // Send time
  Blynk.virtualWrite(V1, currentTime);
  // Send date
  Blynk.virtualWrite(V2, currentDate);
}



void Configuracion_AP() {
  Conexion_WIFI = false;
  Conexion_AP = true;
  /*
  Serial.println("Estableciendo el AP (Access Point)...");
  WiFi.softAP(ssid_AP, password_AP); //Crear AP con ssid y contrase a establecidos previamente
  Serial.println("AP " + String(ssid_AP) + " creado Correctamente!!!");
  IPAddress IP = WiFi.softAPIP(); //Obtener direcci n IP del AP
  Serial.print("Direcci n IP del AP: ");
  Serial.println(IP);
```

```
1158    server.begin();*/
1159    server.on("/", handleRoot);
1160    server.on("/submit", handleSubmit);
1161    server.on("/presionar", handlePresionar);
1162    server.on("/presionar1", handlePresionar1);
1163    server.on("/enviar-hora", handleEnviarHora);
1164    server.on("/enviar-checkbox", handleEnviarDia);
1165    server.on("/Accion1On", handleAccion1);
1166    server.on("/Accion2On", handleAccion2);
1167    server.on("/On1", handlePrender);
1168    server.on("/Off1", handleApagar);
1169    server.on("/saved1", handleGuardar);
1170
1171    server.begin();
1172    Serial.println("Servidor web iniciado");
1173    Serial.println(WiFi.localIP());
1174  }
1175  void Cliente_AP() {
1176    server.handleClient();
1177  }
1178
1179
1180  //*************** PULSADORES ************************
1181
1182  void Pulsador1_() {
1183    delay(tiempo);
1184    if (millis() - rebote > 300 && bandera) {
1185      bandera = 0;
1186      attachInterrupt(digitalPinToInterrupt(pinP1), Int_P1, FALLING);  //Habilita nuevamente int.
1187    }
1188  }
1189
1190
1191
1192  void Conexion_RedWifi() {
1193    Conexion_WIFI = true;
1194    Conexion_AP = false;
1195    Serial.println("Conectando a ");
1196    Serial.println(ssid);
1197    WiFi.begin(ssid, password);              //Conectarse a la red WiFi con ssid y contrase a
       indicada
1198    while (WiFi.status() != WL_CONNECTED) {  //Esperar hasta que se realice la conexi n con el router
1199      delay(500);
1200      Serial.print(".");
1201    }
1202    Serial.println("");
1203    Serial.println("WiFi conectado");
1204
1205    digitalWrite(LED_B, LOW);
1206  }
1207
1208
1209  void handleRoot() {
1210    String message = "<html>";
1211    message += "<head>";
1212    message += "<div align='center'>";
1213    message += "<h1>Proyecto final</h1>";
1214    message += "</div>";
1215    message += "</head>";
1216    message += "<body>";
1217    message += "<div>";
1218    message += "<h1>Seleccione el modo</h1>";
1219    message += "<label for='nombre'>metodo:</label>";
1220    message += "<button onclick='presionarBoton()'>Temporizador</button>";
1221    message += "<script>";
1222    message += "function presionarBoton() {";
1223    message += "  fetch('/presionar', { method: 'POST' })";
1224    message += "    .then(response => {";
1225    message += "      if (response.ok) {";
1226    message += "        console.log('Bot n presionado enviado al servidor');";
1227    message += "      } else {";
```

```
1228    message += "        console.error('Error al enviar la petici n');";
1229    message += "      }";
1230    message += "    })";
1231    message += "    .catch(error => {";
1232    message += "      console.error('Error en la conexi n:', error);";
1233    message += "    });";
1234    message += "}";
1235    message += "</script>";
1236    message += "<button onclick='presionarBoton1()'>Programacion</button>";
1237    message += "<script>";
1238    message += "function presionarBoton1() {";
1239    message += "  fetch('/presionar1', { method: 'POST' })";
1240    message += "    .then(response => {";
1241    message += "      if (response.ok) {";
1242    message += "        console.log('Bot n presionado enviado al servidor');";
1243    message += "      } else {";
1244    message += "        console.error('Error al enviar la petici n');";
1245    message += "      }";
1246    message += "    })";
1247    message += "    .catch(error => {";
1248    message += "      console.error('Error en la conexi n:', error);";
1249    message += "    });";
1250    message += "}";
1251    message += "</script>";
1252    message += "<\div>";
1253
1254    message += "<div>";
1255    message += "<h1>Seleccione la accion</h1>";
1256    message += "<label for='nombre'>accion:</label>";
1257    message += "<button onclick='accion1()'>Conmutador</button>";
1258    message += "<script>";
1259    message += "function accion1() {";
1260    message += "  fetch('/Accion1On', { method: 'POST' })";
1261    message += "    .then(response => {";
1262    message += "      if (response.ok) {";
1263    message += "        console.log('Bot n presionado enviado al servidor');";
1264    message += "      } else {";
1265    message += "        console.error('Error al enviar la petici n');";
1266    message += "      }";
1267    message += "    })";
1268    message += "    .catch(error => {";
1269    message += "      console.error('Error en la conexi n:', error);";
1270    message += "    });";
1271    message += "}";
1272    message += "</script>";
1273    message += "<button onclick='accion2()'>Interruptor</button>";
1274    message += "<script>";
1275    message += "function accion2() {";
1276    message += "  fetch('/Accion2On', { method: 'POST' })";
1277    message += "    .then(response => {";
1278    message += "      if (response.ok) {";
1279    message += "        console.log('Bot n presionado enviado al servidor');";
1280    message += "      } else {";
1281    message += "        console.error('Error al enviar la petici n');";
1282    message += "      }";
1283    message += "    })";
1284    message += "    .catch(error => {";
1285    message += "      console.error('Error en la conexi n:', error);";
1286    message += "    });";
1287    message += "}";
1288    message += "</script>";
1289    message += "<\div>";
1290
1291    message += "<div>";
1292    message += "<h1>Modo en tiempo real</h1>";
1293    message += "<label for='nombre'>Prender o apagar:</label>";
1294    message += "<button onclick='Prender1()'>Prender</button>";
1295    message += "<script>";
1296    message += "function Prender1() {";
1297    message += "  fetch('/On1', { method: 'POST' })";
1298    message += "    .then(response => {";
```

```
1299    message += "      if (response.ok) {";
1300    message += "        console.log('Bot n presionado enviado al servidor');";
1301    message += "      } else {";
1302    message += "        console.error('Error al enviar la petici n');";
1303    message += "      }";
1304    message += "    })";
1305    message += "    .catch(error => {";
1306    message += "      console.error('Error en la conexi n:', error);";
1307    message += "    });";
1308    message += "}";
1309    message += "</script>";
1310    message += "<button onclick='Apagar1()'>Apagar</button>";
1311    message += "<script>";
1312    message += "function Apagar1() {";
1313    message += "  fetch('/Off1', { method: 'POST' })";
1314    message += "    .then(response => {";
1315    message += "      if (response.ok) {";
1316    message += "        console.log('Bot n presionado enviado al servidor');";
1317    message += "      } else {";
1318    message += "        console.error('Error al enviar la petici n');";
1319    message += "      }";
1320    message += "    })";
1321    message += "    .catch(error => {";
1322    message += "      console.error('Error en la conexi n:', error);";
1323    message += "    });";
1324    message += "}";
1325    message += "</script>";
1326    message += "<\div>";
1327
1328    message += "<div>";
1329    message += "<h1>Hora del evento</h1>";
1330    message += "<label for='hora'>Hora:</label>";
1331    message += "<input type='time' id='hora'>";
1332    message += "<button onclick='obtenerHora()'>Enviar Hora</button>";
1333    message += "<script>";
1334    message += "function obtenerHora() {";
1335    message += "  var hora = document.getElementById('hora').value;";
1336    message += "  fetch('/enviar-hora', {";
1337    message += "    method: 'POST',";
1338    message += "    headers: {";
1339    message += "      'Content-Type': 'text/plain'";
1340    message += "    },";
1341    message += "    body: hora";
1342    message += "  })";
1343    message += "  .then(response => {";
1344    message += "    if (response.ok) {";
1345    message += "      console.log('Hora enviada al ESP32');";
1346    message += "    } else {";
1347    message += "      console.error('Error al enviar la hora');";
1348    message += "    }";
1349    message += "  })";
1350    message += "  .catch(error => {";
1351    message += "    console.error('Error en la conexi n:', error);";
1352    message += "  });";
1353    message += "}";
1354    message += "</script>";
1355    message += "<\div>";
1356    message += "<div>";
1357    message += "<label for='Lunes'>Lunes:</label>";
1358    message += "<input type='checkbox' id='Lunes'>";
1359    message += "<label for='Martes'>Martes:</label>";
1360    message += "<input type='checkbox' id='Martes'>";
1361    message += "<label for='Miercoles'>Miercoles:</label>";
1362    message += "<input type='checkbox' id='Miercoles'>";
1363    message += "<label for='Jueves'>Jueves:</label>";
1364    message += "<input type='checkbox' id='Jueves'>";
1365    message += "<label for='Viernes'>Viernes:</label>";
1366    message += "<input type='checkbox' id='Viernes'>";
1367    message += "<label for='Sabado'>Sabado:</label>";
1368    message += "<input type='checkbox' id='Sabado'>";
1369    message += "<label for='Domingo'>Domingo:</label>";
```

```
1370    message += "<input type='checkbox' id='Domingo'>";
1371    message += "<button onclick='enviarCheckbox()'>Enviar dias</button>";
1372    message += "<script>";
1373    message += "function enviarCheckbox() {";
1374    message += "  var checkboxLunes = document.getElementById('Lunes');";
1375    message += "  var checkboxValueLunes = checkboxLunes.checked;";
1376    message += "  var checkboxMartes = document.getElementById('Martes');";
1377    message += "  var checkboxValueMartes = checkboxMartes.checked;";
1378    message += "  var checkboxMiercoles = document.getElementById('Miercoles');";
1379    message += "  var checkboxValueMiercoles = checkboxMiercoles.checked;";
1380    message += "  var checkboxJueves = document.getElementById('Jueves');";
1381    message += "  var checkboxValueJueves = checkboxJueves.checked;";
1382    message += "  var checkboxViernes = document.getElementById('Viernes');";
1383    message += "  var checkboxValueViernes = checkboxViernes.checked;";
1384    message += "  var checkboxSabado = document.getElementById('Sabado');";
1385    message += "  var checkboxValueSabado = checkboxSabado.checked;";
1386    message += "  var checkboxDomingo = document.getElementById('Domingo');";
1387    message += "  var checkboxValueDomingo = checkboxDomingo.checked;";
1388    message += "  fetch('/enviar-checkbox', {";
1389    message += "    method: 'POST',";
1390    message += "    headers: {";
1391    message += "      'Content-Type': 'text/plain'";
1392    message += "    },";
1393    message += "   body: checkboxValueLunes.toString() + ',' + checkboxValueMartes.toString()+ ',' +
          checkboxValueMiercoles.toString()+ ',' + checkboxValueJueves.toString()+ ',' +
          checkboxValueViernes.toString()+ ',' + checkboxValueSabado.toString()+ ',' +
          checkboxValueDomingo.toString()";
1394    message += "  })";
1395    message += "  .then(response => {";
1396    message += "    if (response.ok) {";
1397    message += "      console.log('Checkbox enviado al ESP32');";
1398    message += "    } else {";
1399    message += "      console.error('Error al enviar el checkbox');";
1400    message += "    }";
1401    message += "  })";
1402    message += "  .catch(error => {";
1403    message += "    console.error('Error en la conexi n:', error);";
1404    message += "  });";
1405    message += "}";
1406    message += "</script>";
1407    message += "<\div>";
1408
1409
1410    message += "<div>";
1411    message += "<h1>Guardado</h1>";
1412    message += "<label for='nombre'>Guardar evento:</label>";
1413    message += "<button onclick='Guardar1()'>Guardar</button>";
1414    message += "<script>";
1415    message += "function Guardar1() {";
1416    message += "  fetch('/saved1', { method: 'POST' })";
1417    message += "    .then(response => {";
1418    message += "      if (response.ok) {";
1419    message += "        console.log('Bot n presionado enviado al servidor');";
1420    message += "      } else {";
1421    message += "        console.error('Error al enviar la petici n');";
1422    message += "      }";
1423    message += "    })";
1424    message += "    .catch(error => {";
1425    message += "      console.error('Error en la conexi n:', error);";
1426    message += "    });";
1427    message += "}";
1428    message += "</script>";
1429    message += "<\div>";
1430
1431    message += "</body></html>";
1432    server.send(500, "text/html", message);
1433  }
1434
1435  void handleSubmit() {
1436    nombre = BotonPresionado;
1437    String message = "Nombre recibido: " + nombre;
```

```
  server.send(200, "text/plain", message);
}

// //Determinar el tipo de accion

void handlePresionar() {
  BotonPresionado = "Temporizador";
  Modo_Operacion = 1;
  Serial.println(BotonPresionado);
  Serial.println(Modo_Operacion);
  server.send(200, "text/plain", "Bot n presionado");
}
void handlePresionar1() {
  BotonPresionado = "Programacion";
  Modo_Operacion = 0;
  Serial.println(BotonPresionado);
  Serial.println(Modo_Operacion);
  server.send(200, "text/plain", "Bot n presionado");
}

void handleAccion1() {
  AccionSelect = "Conmutador";
  Tipo_Interruptor = 1;
  Serial.println(AccionSelect);
  Serial.println(Tipo_Interruptor);
  server.send(200, "text/plain", "Bot n presionado");
  // Tipo_Accion();
}
void handleAccion2() {
  AccionSelect = "Interruptor";
  Tipo_Interruptor = 0;
  Serial.println(AccionSelect);
  Serial.println(Tipo_Interruptor);
  server.send(200, "text/plain", "Bot n presionado");
  // Tipo_Accion();
}

void handlePrender() {
  Tipo_Estado = 0;
  Serial.print("Prendido: ");
  Serial.println(Tipo_Estado);
  server.send(200, "text/plain", "Bot n presionado");
}
void handleApagar() {
  Tipo_Estado = 1;
  Serial.print("Apagado: ");
  Serial.println(Tipo_Estado);
  server.send(200, "text/plain", "Bot n presionado");
}
void handleGuardar() {
  Serial.println("Descomentar...");
}

void handleEnviarDia() {
  checkboxValue = server.arg("plain");
  for (int j = 0; j < 7; j++) {
    diasF[j] = Separar(checkboxValue, ",", true, j);
  }
  for (int j = 0; j < 7; j++) {
    if (diasF[j] == "true") {
      DiasSeleccionados[j] = j + 1;
    } else {
      DiasSeleccionados[j] = "0";
    }
  }
  for (int j = 0; j < 7; j++) {
    Serial.println(DiasSeleccionados[j]);
  }
}
void handleEnviarHora() {
  hora = server.arg("plain");
```

```
1509    Serial.println("Hora:");
1510    Serial.println(hora);
1511
1512    for (int j = 0; j < 2; j++) {
1513      hora1[j] = Separar(hora, ":", true, j);
1514    }
1515    for (int i = 0; i < 2; i++) {
1516      Serial.println(hora1[i]);
1517    }
1518    Serial.println(horaFinal(hora1));
1519  }
1520
1521  double horaFinal(String hora[2]) {
1522    int horaT = hora[0].toInt();
1523    int minutoT = hora[1].toInt();
1524    int horaF;
1525    horaF = (horaT * 3600) + (minutoT * 60);
1526    return horaF;
1527  }
```

Listing 1: Código Proyecto 5