

# Final Técnicas Digitales III

## Apolo Network

Sistema de medición y control de temperatura sobre STM32F4 Discovery en conjunto con un servidor NodeJS comunicados a través del protocolo TCP

### Profesor:

Mg. Guillermo Friedrich

### Ayudante:

Ing. Marcelo Mallimaci

### Alumnos:

Juan Cruz Guidi  
Mauricio Mancini

L.N: 16546  
L.N: 16329

## Introducción:

Este proyecto se basa en la implementación de un sistema de control de temperatura utilizando sistemas embebidos STM32F4 y Raspberry pi 2 comunicándose a través de mensajes TCP sobre una red enrutada con un router mikrotik RB433.

El sistema de control consiste en la medición de temperatura y ventilación en diferentes puntos de un área o establecimiento. El siguiente proyecto fue diseñado para instalaciones industriales. La idea de este es poder controlar la temperatura en un área determinada, sensando el calor en diferentes puntos y accionando ventiladores para enfriar el ambiente, todo controlado por los operarios en cualquier sector de la planta.

Consiste en una serie de dispositivos que tienen sensores de temperatura y pueden activar o desactivar un ventilador para poder refrigerar el área. Estos dispositivos se conectan en red y se comunican con un servidor que recibe todos los datos de los sensores y los guarda en una base de datos. Este, realiza gráficos y, dependiendo de la temperatura, le envía la orden de prender o apagar cada uno de los ventiladores. La información está disponible para los operarios y estos pueden establecer o modificar las acciones del servidor para con los dispositivos accediendo desde cualquier dispositivo conectado a la red.

Todas las comunicaciones se realizan a través de una red y dispositivos de red tales como switches y routers. Se eligió este método ya que en todas las instalaciones existen redes ya cableadas y son aptas para trabajar en este tipo de habientes industriales.

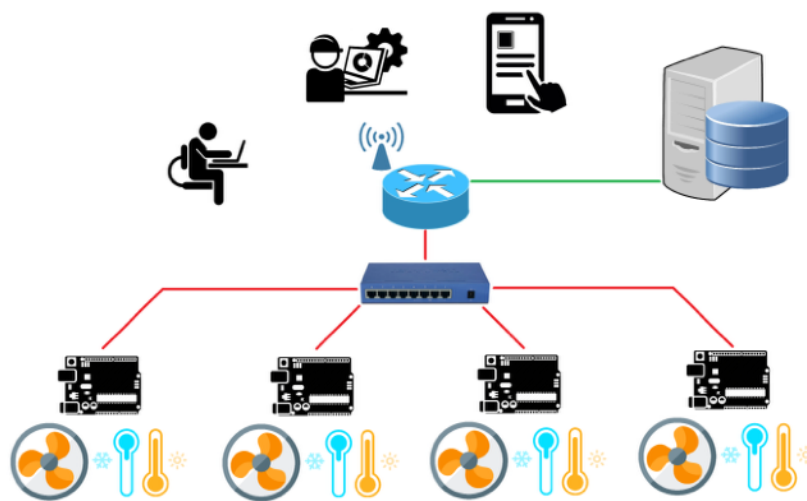


FIGURA 1 - ESQUEMA DE LA RED

En la figura 1 se observa cual es el esquema de la red a utilizar para el siguiente proyecto y donde se encuentra cada dispositivo. Como elemento central se encuentra un router que controla tres redes privadas de clase C, una de ellas de tipo wireless o inalámbrica.

En la primera red (color rojo) se conectan todos los dispositivos de medición de temperatura que se comunican con el servidor. La cantidad de estos dependerá del tamaño del área a controlar, es por eso que se utiliza un switch con la cantidad de bocas necesarias. Los dispositivos conectados deben tener conectividad Ethernet, poder leer los datos del sensor de temperatura y activar o desactivar el ventilador de refrigeración. Todas las comunicaciones con el servidor son a través del protocolo TCP/IP.

En la segunda red (color verde) se conecta el servidor donde se guardan, analizan y muestran todos los datos provenientes de la red 1. Este recibe la temperatura de cada uno de los dispositivos de medición y los almacena. Dependiendo de la temperatura, le da la orden de prender o apagar el ventilador de refrigeración de cada sector. La otra función que tiene es mostrar los datos al operario que acceda a este y hasta permitirle modificar parámetros de configuración.

En la tercera red (inalámbrica) se conectan todos los dispositivos controlados por los operarios. Estos pueden utilizar computadoras de escritorio, notebooks, smartphones o cualquier otro dispositivo que posea conectividad WiFi y un navegador instalado. Los trabajadores podrán acceder al servidor y controlar el sistema de refrigeración en tiempo real, desde cualquier punto de la planta donde tenga cobertura la red.

## Aplicación:

Para la implementación de este proyecto, se puede dividir en tres etapas; la configuración de la red, la programación del dispositivo sensor de temperatura y la programación del servidor. Para la aplicación se realizaron pequeños cambios de la red original a fines prácticos para poder demostrar su correcto funcionamiento.

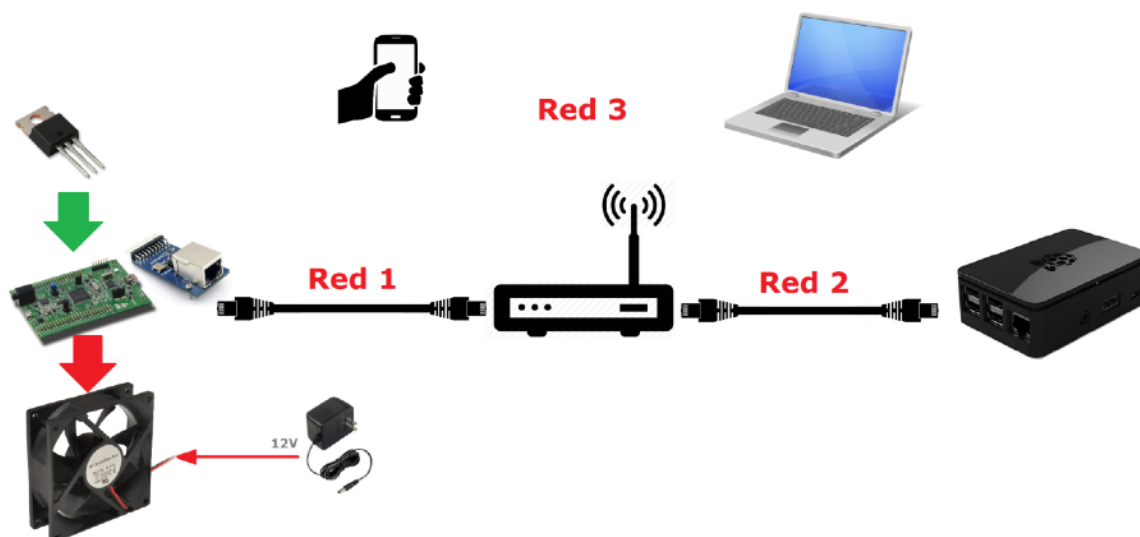


FIGURA 2 - APLICACIÓN REAL DEL PROYECTO

En la figura 2 se observa el esquema de red y dispositivos utilizados en este proyecto. Como dispositivo sensor de temperatura se utilizó una STM Discovery, con un módulo de Ethernet DP83848, un sensor LM35DT y un cooler de 12v controlado con un transistor. Como servidor se eligió una Raspberry pi 2 con un SO Linux. El router usado es un Mikrotik RB433 con tres puertos Ethernet y conectividad wireless. Del diseño original se eliminó el uso de un switch en la red 1 ya que se utilizó un solo dispositivo y se puede conectar directamente al router.

## STM Discovery:

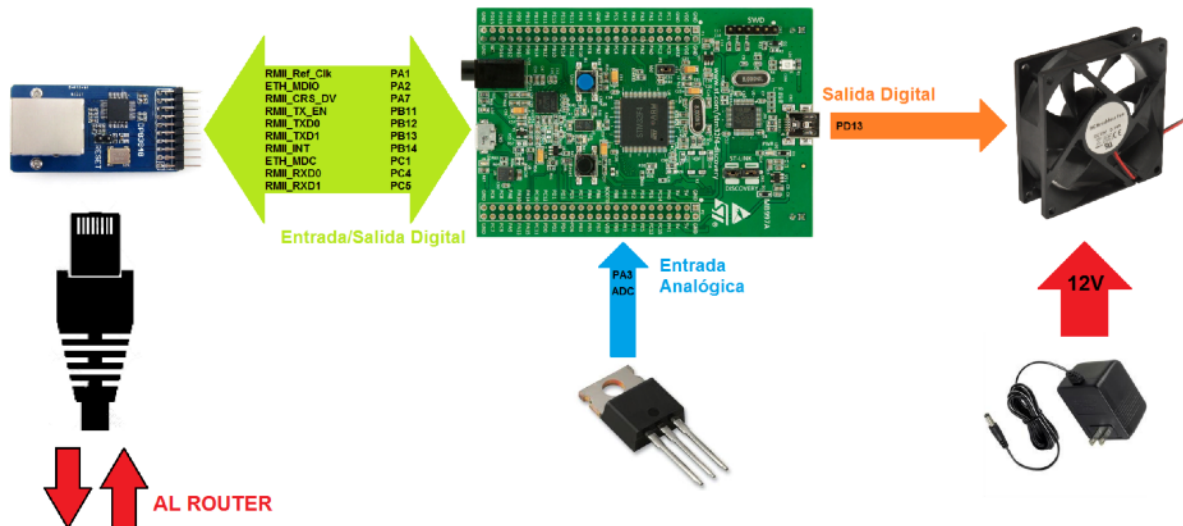


FIGURA 3 - DIAGRAMA DE ELEMENTOS DE LA DISCOVERY

En la figura 3 se puede ver el conexionado de los periféricos con la placa Discovery indicando el tipo de I/O y el puerto utilizado en cada caso. Para accionar el cooler con una salida digital de 3,3V se implementó una llave con un transistor bipolar y resistencias limitadoras de corriente. La tensión de 12V CC para encender el ventilador proviene de un transformador conectado a 220V AC.

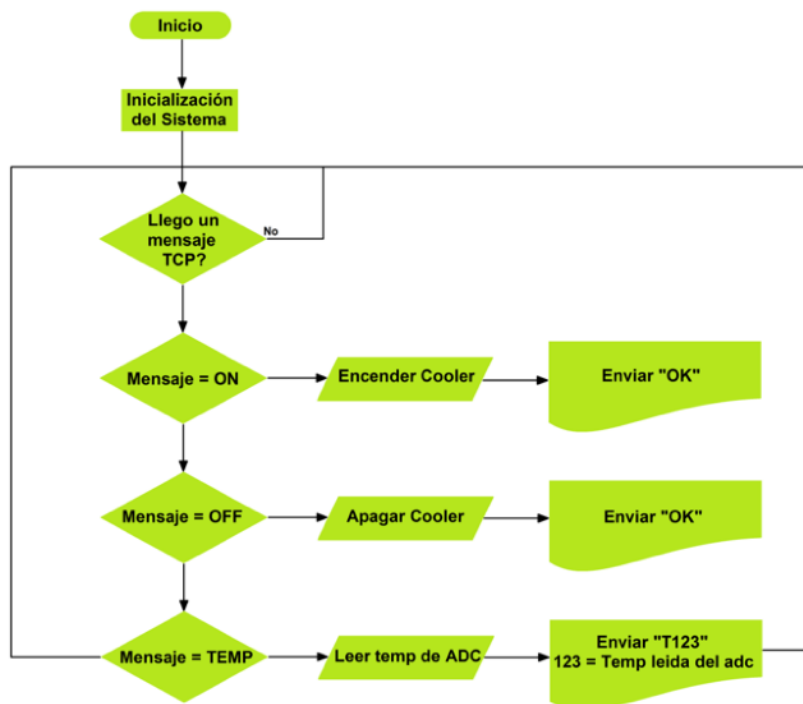


FIGURA 4.1 - DIAGRAMA DEL PROGRAMA

A continuación se explicará el funcionamiento del programa desarrollado para la placa STM32F4 a través de fragmentos de código.

```
int main(void)
{
    /* - Inicializacion del Sistema - */
    SystemInit();
    UB_Led_Init();
    ETH_BSP_Config();           //Inicializacion de Modulo DP83848
    LwIP_Init();                //Inicializacion de Red
    tcp_echo_server_init();      //Inicializacion de Servidor TCP
    UB_ADC1_SINGLE_Init();       //Inicializacion de ADC
    /* -----*/

    UB_Led_On(LED_GREEN);       //Inicializacion CORRECTA

    while(1){

        /* check if any packet received */
        if (ETH_CheckFrameReceived()) {
            /* process received ethernet packet */
            UB_Led_Toggle(LED_BLUE);
            LwIP_Pkt_Handle();
        }
        /* handle periodic timers for LwIP */
        LwIP_Periodic_Handle(LocalTime);
    }
}
```

**FIGURA 4.2 – CÓDIGO PRINCIPAL**

En la figura 4.2 se ve el main donde se muestra cual es el flujo del programa principal. Este consta de dos etapas, inicializar el sistema y esperar por mensajes TCP. En la primera parte se inicializa el sistema, los leds, los pines y el módulo DP83848, el LwIP, el cliente TCP y la lectura con el ADC. La versión del LwIP utilizada es la 1.3.1 donde se hace uso del stack TCP/IP.

```

//-----
// MAC-Adresse (02:00:00:00:00:00)
//-----
#define MAC_ADDR0  2
#define MAC_ADDR1  0
#define MAC_ADDR2  0
#define MAC_ADDR3  0
#define MAC_ADDR4  0
#define MAC_ADDR5  0

//-----
// static IP-Address (192.168.2.10)
//-----
#define IP_ADDR0  192
#define IP_ADDR1  168
#define IP_ADDR2  2
#define IP_ADDR3  10

//-----
// Netmask (255.255.255.0)
//-----
#define NETMASK_ADDR0  255
#define NETMASK_ADDR1  255
#define NETMASK_ADDR2  255
#define NETMASK_ADDR3  0

//-----
// Gateway Address (192.168.2.1)
//-----
#define GW_ADDR0  192
#define GW_ADDR1  168
#define GW_ADDR2  2
#define GW_ADDR3  1
```

**FIGURA 5 – CONFIGURACIÓN DE DIRECCIÓN DE RED**

Se utilizó esta configuración de red, con una IP fija y en el dominio 192.168.2.XXX correspondiente a la red asignada para este dispositivo. Esta dirección es utilizada cuando se conecta a la red en la función LwIP\_Init.

```
void tcp_echo_server_init(void)
{
    /* create new tcp pcb */
    tcp_echo_server_pcb = tcp_new();

    if (tcp_echo_server_pcb != NULL) {
        err_t err;

        /* bind echo_pcb to port 7 (ECHO protocol) */
        err = tcp_bind(tcp_echo_server_pcb, IP_ADDR_ANY, 7);

        if (err == ERR_OK) {
            /* start tcp listening for echo_pcb */
            tcp_echo_server_pcb = tcp_listen(tcp_echo_server_pcb);

            /* initialize LwIP tcp_accept callback function */
            tcp_accept(tcp_echo_server_pcb, tcp_echo_server_accept);
        } else {
            //printf("Can not bind pcb\n");
        }
    } else {
        //printf("Can not create new pcb\n");
    }
}
```

FIGURA 6 – INICIALIZACIÓN DE SERVIDOR TCP

En la figura 6 se muestra la inicialización del cliente TCP. Primero se crea el bind y luego se configura la función “tcp\_echo\_server\_accept” como función de callback que será llamada cuando se reciba algún paquete TCP.

```

err_t ret_err;
struct tcp_echo_server_struct *es;

LWIP_UNUSED_ARG(arg);
LWIP_UNUSED_ARG(err);

/* set priority for the newly accepted tcp connection newpcb */
tcp_setprio(newpcb, TCP_PRIO_NORMAL);

/* allocate structure es to maintain tcp connection informations */
es = (struct tcp_echo_server_struct *)mem_malloc(sizeof(struct tcp_echo_server_struct));
if (es != NULL) {
    es->state = ES_ACCEPTED;
    es->pcb = newpcb;
    es->p = NULL;

    /* pass newly allocated es structure as argument to newpcb */
    tcp_arg(newpcb, es);

    /* initialize lwip tcp_recv callback function for newpcb */
    tcp_recv(newpcb, tcp_echo_server_recv);

    /* initialize lwip tcp_err callback function for newpcb */
    tcp_err(newpcb, tcp_echo_server_error);

    /* initialize lwip tcp_poll callback function for newpcb */
    tcp_poll(newpcb, tcp_echo_server_poll, 1);

    ret_err = ERR_OK;
} else {
    /* return memory error */
    ret_err = ERR_MEM;
}
return ret_err;

```

FIGURA 7.1 – RESPUESTA A MENSAJE TCP

En la figura 7.1 se muestra la función de callback, donde se establecen las funciones `tcp_echo_server_recv`, `tcp_echo_server_error` y `tcp_echo_server_poll` como funciones para recibir datos, error en la comunicación y de poll.

```

void respuesta(struct tcp_pcb *tpcb, struct tcp_echo_server_struct *es) {

    /* es->p->payload => Mensaje recibido */

    if(strcmp(es->p->payload, "ON")==0) {
        UB_Led_On(LED_ORANGE);           //Turn ON Flag
        strcpy(temperatura, "OK");        //Respuesta al servidor
    }
    else if(strcmp(es->p->payload, "OFF")==0) {
        UB_Led_Off(LED_ORANGE);          //Turn OFF Flag
        strcpy(temperatura, "OK");        //Respuesta al servidor
    }
    else if(strcmp(es->p->payload, "TEM")==0) {
        getTemperature();                 //Obtener y enviar Temperatura
    }
    else
        strcpy(temperatura, "ERROR");     //Mensaje incorrecto

    tcp_echo_server_send(tpcb, es);       //Envio de respuesta

    return;
}

```

FIGURA 7.2 – RESPUESTA A MENSAJE TCP



En la función “respuesta”, se analiza el mensaje recibido y se guarda en la variable “temperatura” la respuesta a enviar.

Los mensajes esperados y respuestas son las siguientes:

| Mensaje | Respuesta | Significado  |
|---------|-----------|--|
| ON      | OK        | Prender el ventilador  |
| OFF     | OK        | Apagar el ventilador   |
| TEMP    | T123      | Responde con la temperatura, la forma de enviarla es con el caracter “T” seguido del valor del ACD |
| Otro    | ERROR     | El mensaje no es ninguno de los anteriores y no se puede dar una respuesta                         |

En la función “getTemperature” se lee el valor del ACD y se guarda en la variable “temperatura”.

```
void getTemperature() {  
  
    uint16_t adc_lectura;  
    char abc[6] = "T";  
  
    adc_lectura = UB_ADC1_SINGLE_Read_MW(ADC_PA3); //Lectura del valor del ADC  
  
    itoa(adc_lectura, temperatura, 10);  
  
    strcat(abc, temperatura); //Formato de envio de informacion  
    strcpy(temperatura, abc); // "T123"  
  
    return;  
}
```

FIGURA 7.3 – RESPUESTA A MENSAJE TCP

Finalmente se envía la respuesta. Se puede ver que en la función “tcp\_write”, de la figura 7.4, se envía el contenido de la variable temperatura.

```

while ((wr_err == ERR_OK) &&
      (es->p != NULL) &&
      (es->p->len <= tcp_sndbuf(tpcb)))
{
    /* get pointer on pbuf from es structure */
    ptr = es->p;

    /* enqueue data for transmission */
    wr_err = tcp_write(tpcb, temperatura, strlen((char*)temperatura), 1);

    if (wr_err == ERR_OK) {
        u16_t plen;

        plen = strlen((char*)temperatura);

        /* continue with next pbuf in chain (if any) */
        es->p = ptr->next;

        if (es->p != NULL) {
            /* increment reference count for es->p */
            pbuf_ref(es->p);
        }

        /* free pbuf: will free pbufs up to es->p (because es->p has a reference count > 0) */
        pbuf_free(ptr);

        /* Update tcp window size to be advertized : should be called when received
        data (with the amount plen) has been processed by the application layer */
        tcp_recved(tpcb, plen);
    } else if (wr_err == ERR_MEM) {
        /* we are low on memory, try later / harder, defer to poll */
        es->p = ptr;
    }
}

```

**FIGURA 7.4 – RESPUESTA A MENSAJE TCP**

## Router:

Para el ruteo de la red se utilizó un router Mikrotik RB433. Este posee tres puertos Ethernet y una antena para conexiones inalámbricas. Como primer paso se resetea la configuración a la de fábrica y se borran todas las pre-configuraciones que se realizan por defecto. Luego de esto se realizan las configuraciones de cada una de las interfaces. Luego se crean las redes indicando su dirección, la máscara y la dirección de Broadcast. El siguiente cuadro muestra cómo se realizó.

| Interface    | IP asignada | Red             | Dir. de red | Dir. Broadcast |
|--------------|-------------|-----------------|-------------|----------------|
| Puerto ETH 1 | -           | No se configura | -           | -              |
| Puerto ETH 2 | 192.168.1.1 | Red1            | 192.168.1.0 | 192.168.1.255  |
| Puerto ETH 3 | 192.168.2.1 | Red2            | 192.168.2.0 | 192.168.2.255  |
| WLAN         | 192.168.3.1 | Red3            | 192.168.3.0 | 192.168.3.255  |

Luego se configuran los servidores DHCP para cada una de las redes. Estos asignan una IP a los dispositivos que se conecten a la red, si es que estos no tienen una IP fija. El rango de IP que asigna abarca desde la dirección 192.168.X.100 a 192.168.X.254, siendo X 1, 2 o 3, dependiendo de qué red sea. Las primeras 98 direcciones se reservan para dispositivos con IP fija o estática.

## Servidor:

El servidor se realizó sobre NodeJS, un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor, basado en el lenguaje de programación ECMAScript, asíncrono, con I/O de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google.

El sistema cuenta con dos servidores que funcionan de manera independiente, un servidor TCP y un servidor HTTP, se podrían haber realizado de manera conjunta, pero se hizo así para tener un mayor control de cada uno y un código más limpio.

Se utilizan las funciones propias de NodeJS como `net.createServer` para la creación del servidor TCP y para el servidor HTTP se utiliza NodeJS en conjunto con Express, un framework de aplicaciones web que facilita la creación de paginas y aplicaciones web híbridas, y AngularJS.

Para el almacenamiento de datos se deja implementado pero comentadas las funciones necesarias para trabajar con MongoDB, una base de datos noSQL.

El servidor TCP es el encargado de recibir los datos enviados por el servidor HTTP y los enviados por la STM, funciona como el puente entre ambos.

Una de las funciones principales del servidor TCP es la de preguntarle cada 1 Seg a la STM por la temperatura, y al recibir el dato transformar el valor medido por el ADC a temperatura, guardarlo y controlar si no supera el máximo cargado para encender el ventilador o si se encuentra por debajo del mínimo para apagar el mismo.

Cada vez que el servidor recibe la temperatura se envía una respuesta a la STM antes de cerrar la conexión TCP, esto sirve en caso de que el servidor corriendo sobre la STM se reinicie, haciendo que vuelva a arrancar en el estado que se encontraba antes del inconveniente.

La función encargada de conectarse con la STM y de pedir la temperatura es la siguiente:

```

function myFunction() {
  setInterval(function(){
    var cliTemp = new net.Socket();

    cliTemp.connect(7, '192.168.2.10', function() {
      console.log('Sent TEM');
      cliTemp.write('TEM');
    });

    cliTemp.on('data', function(data) {
      console.log('Received: ' + data);

      if (data[0] == 84) {

        var datos = String(data);
        var temp = parseFloat(datos.substring(1));
        temp = (temp/17.5).toFixed(2);

        tempArray.push(temp);
        if (temp > tMax) {
          cliTemp.write('ON');
          isOn = 1;
        } else if (temp < tMin) {
          if (forcedOn != 1) {
            cliTemp.write('OFF');
            isOn = 0;
          }
        } else if (isOn == 1) {
          cliTemp.write('ON');
          isOn = 1;
        }
      }

      cliTemp.destroy(); // kill client after server's response
    });
  }, 1000);
}

myFunction();

```

**FIGURA 8 – FUNCIÓN PRINCIPAL DEL SERVIDOR**

Cada 1 segundo se crea un cliente con la función `cliTemp.connect` y se envía “TEM” a la STM, una vez recibido el evento “data”, este se procesa y controla si la temperatura es menor o mayor a los limites establecidos.

El servidor también cuenta con una función que se encarga de controlar los datos recibidos sobre el servidor TCP con el que se comunica al servidor HTTP, a fin de procesar los datos recibidos por el mismo o enviar los datos demandados.

Para que funcione correctamente el servidor HTTP se configura la ruta en la que se encuentran los archivos HTML y JS que debe utilizar para crear la web.

```
app.use(express.static(__dirname + '/public'));
```

Utilizando `express` se crea una app web que recibe las diferentes peticiones de una web normal.

```
app.get('/', function(req, res){  
    res.redirect('/index.html');  
});
```

Cuando se solicita a travez del navegador la pagina de inicio, es decir, en este caso, cuando se accede al ip del servidor HTTP a través del navegador.

En el caso que el requerimiento sea del tipo POST, la función encargada de procesar este tipo de cosas es:

```
app.post('', function(req, res, next){});
```

En el código de `httpserver.js` se verá la utilización de dicha función en diferentes oportunidades.

Para la configuración del puerto del servidor se utiliza: `app.listen(8080);`

## Uso-Servidor:

Para poder levantar el servidor se debe iniciar por consola, desde la computadora donde el mismo se encuentre, utilizando los comandos:

```
nodejs server.js
```

```
nodejs httpserver.js
```

y en pantalla se mostrara el inicio del servidor.

De esta manera, el servidor estará ejecutándose solo mientras tenga la sesión activa, es decir, en cuanto cierre el terminal, ya no funcionarán los servidores.

Para poder dejar encendido el servidor y poder cerrar la consola debe utilizarse el comando `screen` previo a `nodejs server.js` y a `nodejs httpserver.js` .

Los comandos introducidos quedarían en el orden:

```
screen -S ServidorTCP
```

```
[enter]
```

```
nodejs server.js
```

```
[ctrl + A][ctrl + D]
```

```
screen -S ServidorHTTP
```

```
[enter]
```

```
nodejs httpserver.js
```

```
[ctrl + A][ctrl + D]
```

Para salir de la ventana generada por `screen`, debe clickear `ctrl A ctrl D`, a fin de dejarla en segundo plano.

Para poder retomar el control de dicha terminal debe tipearse:

```
screen -r nameOfScreen
```

Y en caso de querer cerrarla, solo debe escribirse `exit`, y volverá a la terminal original.

Al entrar por el navegador a la dirección y puerto del servidor HTTP se verá la pantalla que se muestra en la figura 9.1:

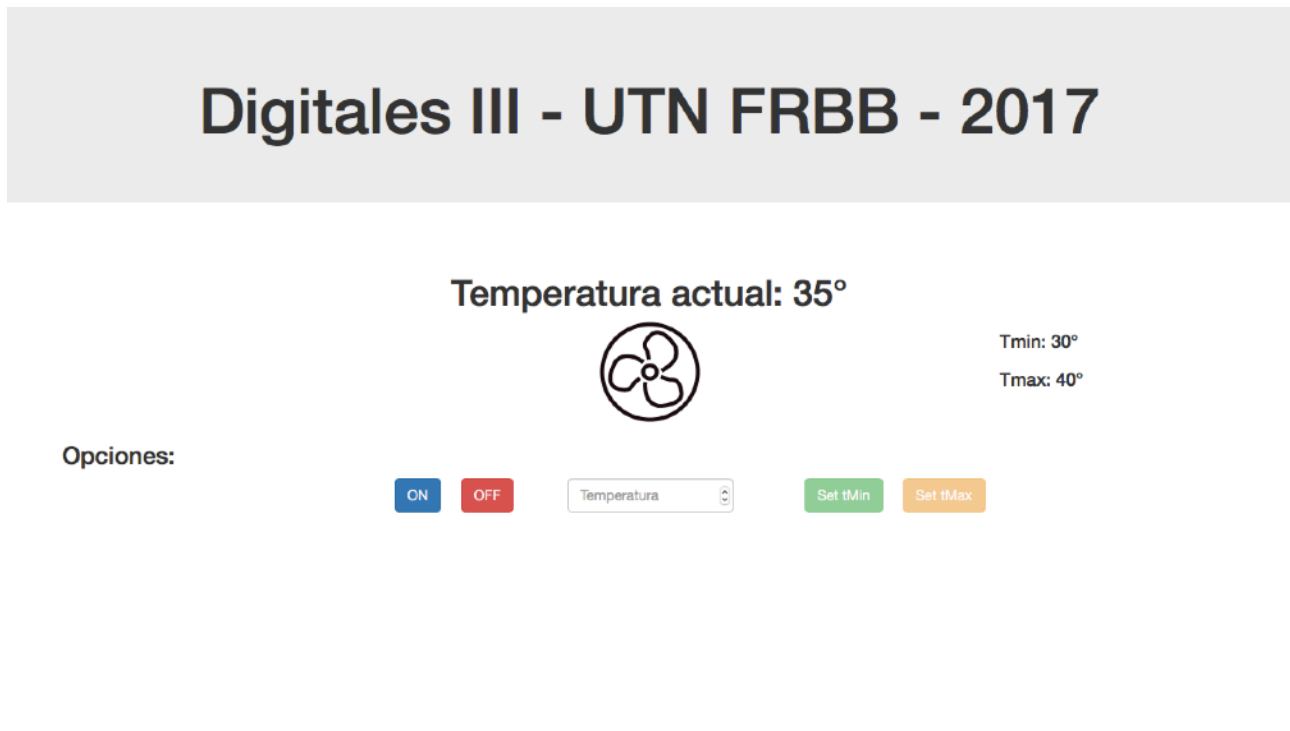


FIGURA 9.1 – VISTA DE LA WEB DESDE LA COMPUTADORA

O al entrar desde un celular, como la web realizada es responsive, se verá como muestra la figura 9.2

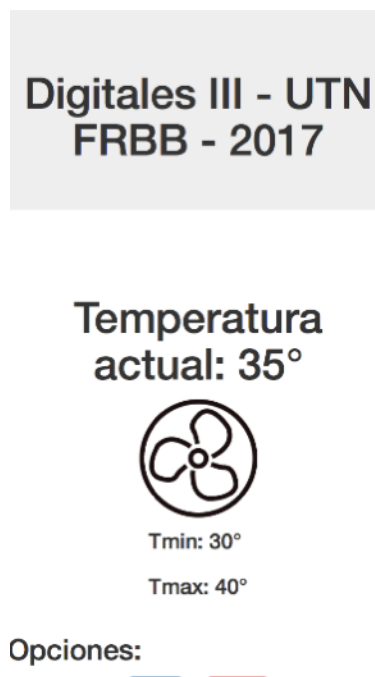


FIGURA 9.2 – VISTA DE LA WEB DESDE EL CELULAR



## ANEXO 1:

El código del proyecto puede descargarse desde <https://github.com/Sonry101/STM32F4Discovery-Digitales3-NodeJS-TCP>

El mismo se encuentra organizado como se muestra en la figura 10, a fin de separar de manera sencilla los códigos correspondientes a cada parte del proyecto.

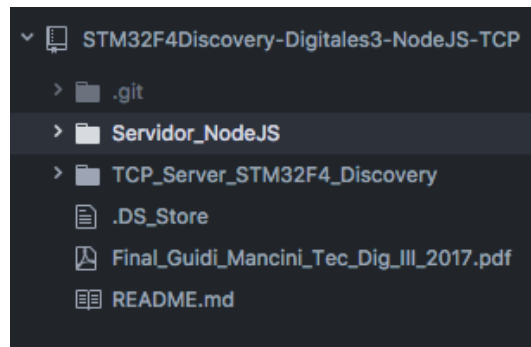


FIGURA 10 – ORGANIZACIÓN DE LA CARPETA

## **ANEXO 2:**

Node.js® es un entorno de ejecución para JavaScript construido con el motor de JavaScript V8 de Chrome. Node.js usa un modelo de operaciones E/S sin bloqueo y orientado a eventos, que lo hace liviano y eficiente. El ecosistema de paquetes de Node.js, npm, es el ecosistema mas grande de librerías de código abierto en el mundo.

El mismo se puede descargar desde su web oficial: <https://nodejs.org/es/> .Donde además se encontrará la documentación y las guías de instalación para cada plataforma en la que se desee utilizar.

NPM, es el gestor de paquetes para JavaScript, su web oficial: <https://www.npmjs.com>