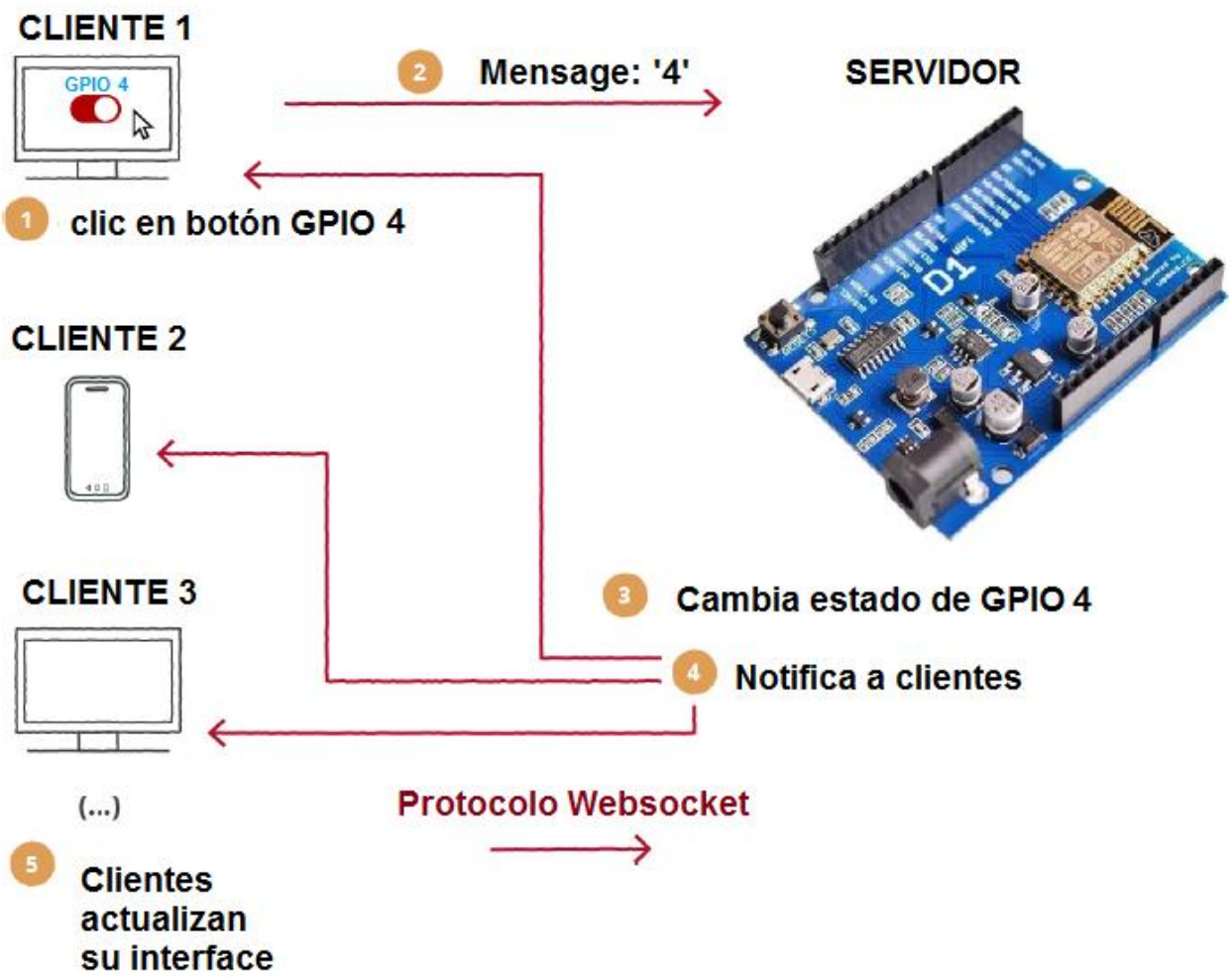
SERVIDOR WEB DE MÚLTIPLES SALIDAS CON WEBSOCKETS

El servidor tendrá una interface gráfica de control con 4 interruptores deslizables como se muestra a continuación:

LISTA DE MATERIALES:

1 Wemos D1 R1
1 cable USB - micro
1 Protoboard
3 LEDs
3 resistores de 220 Ohms
1 alambre dupont color NEGRO
3 alambres dupont cualquier color diferente a NEGRO.

En la siguiente figura se ilustra el funcionamiento de control mediante WebSockets:



ARCHIVO main.cpp

```
#include <Arduino.h>
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>
#include <LittleFS.h>
#include <Arduino_JSON.h>

const char* ssid = "MEGACABLE-DE3R5";
const char* password = "f452oTnk";
// Crea el objeto AsyncWebServer en el puerto 80
AsyncWebServer server(80);
/* La biblioteca ESPAsyncWebServer incluye un complemento WebSocket que facilita el
manejar las conexiones WebSocket. Crear un objeto AsyncWebSocket llamado ws para
manejar las conexiones en la ruta / ws. */
AsyncWebSocket ws("/ws");
// Establece el número de salidas
#define NUM_SALIDAS 4
// Asigna los números de salidas
// En Wemos D1: 5 está en /D3 y 13 está en /D7
int salidaGPIOs[NUM_SALIDAS] = {4, 5, 12, 13};
/* El código está preparado para controlar los GPIO 4, 5, 12 y 13. Puede modificarse
la cantidad de NUM_SALIDAS y variables salidaGPIOs para cambiar el número de GPIO y
cuáles los que se requiere controlar.
La variable NUM_SALIDAS define la cantidad de GPIOs. El salidaGPIOs es una matriz
con los números GPIO que desea controlar.

NOTA: Si cambia la cantidad de GPIOs y los GPIO que desea controlar, también debe
cambiarse los identificadores de los elementos HTML en el documento index.html. */

// Inicializa LittleFS
void initFS() {
 if (!LittleFS.begin()) {
 Serial.println("Un error ha ocurrido mientras se montaba LittleFS");
}
 Serial.println("LittleFS montado exitosamente");
}
// Inicializa WiFi
void initWiFi() {
 WiFi.mode(WIFI_STA);
 WiFi.begin(ssid, password);
 Serial.print("Conectando a la red  WiFi ..");
 while (WiFi.status() != WL_CONNECTED) {
    Serial.print('.');
 delay(1000);
 }
 Serial.println(WiFi.localIP());
```

```
}

/* La función ObtenerEstadosDeSalida () comprueba el estado de todos sus GPIO
y devuelve una variable de cadena JSON con esa información. */
String ObtenerEstadosDeSalida(){
 JSONVar miObjetoVectorDePuertosJSON;
 for (int i =0; i<NUM_SALIDAS; i++){
 miObjetoVectorDePuertosJSON["gpios"][i]["salida"] = String(salidaGPIOs[i]);
 miObjetoVectorDePuertosJSON["gpios"][i]["estado"] = String(digitalRead(salidaGPIOs[i]));
 }
    /* La variable json VectorDePuertos contiene un arreglo de GPIOs
    con la siguiente estructura:
       gpios[0] = salida =  estado = 0
       gpios[1] = salida = 5 estado = 0
       gpios[2] = salida = 12 estado = 0
       gpios[3] = salida = 13 estado = 0
    */
 String CadenaJSON = JSON.stringify(miObjetoVectorDePuertosJSON);
  /* la función JSON.stringfy convierte el vector en formato
  texto JSON */
  Serial.print(CadenaJSON); // se envía la cadenaJSON al monitor de arduino
 return CadenaJSON;
}
void notifyClients(String estado) {

  /* La función notifyClients () notifica a todos los clientes con un mensaje
   que contiene lo que sea que se pase como argumento. En este caso, queremos
   notificar a todos los clientes de el estado actual de todos los GPIO
   siempre que haya un cambio. */

 ws.textAll(estado);
}
void handleWebSocketMessage(void *arg, uint8_t *data, size_t len) {
 AwsFrameInfo *info = (AwsFrameInfo*)arg; //La totalidad del mensaje está en un frame

 /*La función handleWebSocketMessage () es una función de devolución
 de llamada que se ejecutará siempre que recibamos nuevos datos de los
 clientes a través del protocolo WebSocket. Como se explica previamente,
 el cliente enviará el mensaje "estados" para solicitar el GPIO actual de
 los estados o un mensaje que contiene el número GPIO para cambiar el estado. */

 if (info->final && info->index == 0 && info->len == len && info->opcode == WS_TEXT) {
 data[len] = 0;
 if (strcmp((char*)data, "estados") == 0) {
   /* Si no (0) se han recibido los estados, obtenerlos */
 notifyClients(ObtenerEstadosDeSalida());
 }
 else{
 int gpio = atoi((char*)data); // atoi convierte a entero el texto
```

```
 digitalWrite(gpio, !digitalRead(gpio));
 /* grabamos el número de gpio en la variable gpio
 observa que se graba el valor negado ! de la lectura digital
 porque es un interruptor deslizable de dos estados */
 notifyClients(ObtenerEstadosDeSalida());

/* Si recibimos el mensaje "estados", enviamos un mensaje a todos los
clientes con el estado de todos los GPIO que utilizan la función notifyClients.
Llamar a la función ObtenerEstadosDeSalida() devuelve una cadena JSON con
los estados GPIO.
*/
 }
 }
}
/*
Ahora necesitamos configurar un detector de eventos para manejar los diferentes
pasos asincrónicos del protocolo WebSocket. Este controlador de eventos
se puede implementar definiendo onEvent () de la siguiente manera:
*/
void onEvent(AsyncWebSocket *server, AsyncWebSocketClient *client,AwsEventType
TipoDeEvento,
 void *arg, uint8_t *data, size_t len) {

 switch (TipoDeEvento)
 { /* El argumento TipoDeEvento representa el evento que ocurre.
 Puede tomar los siguientes valores: */
 case WS_EVT_CONNECT: // cuando un cliente ha iniciado sesión
 Serial.printf("Cliente WebSocket #%u conectado de %s\n", client->id(), client-
>remoteIP().toString().c_str());
 break;
 case WS_EVT_DISCONNECT: // cuando un cliente se desconecta
 Serial.printf("Cliente WebSocket #%u desconectado\n", client->id());
 break;
 case WS_EVT_DATA: // cuando se recibe un paquete de datos del cliente
 handleWebSocketMessage(arg, data, len);
 break;
 case WS_EVT_PONG: // en respuesta a una solicitud de ping
 /* Ping es un comando o una herramienta de diagnóstico que permite hacer una
 verificación   del estado de una determinada conexión o host local.
Es un acrónimo para Packet Internet Groper, lo que literalmente significa
 “buscador de paquetes en redes”. Se trata de un comando que permite verificar
 el estado de  una conexión para determinar si una dirección IP específica
 o host es accesible desde la red o no. */
 case WS_EVT_ERROR: // cuando se recibe un error del cliente.
 break;
 }
}
void initWebSocket() {
/*
```

```
Finalmente, la función initWebSocket () inicializa el protocolo WebSocket.
*/
 ws.onEvent(onEvent);
 server.addHandler(&ws);
}
void setup(){
 // Puerto serie para propósitos de depuración
 Serial.begin(115200);
 // Prepara GPIOs como salidas
 for (int i =0; i<NUM_SALIDAS; i++){
 pinMode(salidaGPIOs[i], OUTPUT);
 }
 initFS();  // iniciliza File System
 initWiFi(); // inicializa Wi Fi
 initWebSocket(); // Inicializa WebSocket

 /* Las siguientes líneas manejan lo que sucede cuando recibe una
 solicitud en la raíz (/) URL (dirección IP de ESP8266). */
 server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
 request->send(LittleFS, "/index.html", "text/html",false);
/* Se usa LittleFS porque es ESP8266
   El archivo principal en la raíz de littleFS es /index.html
   Es texto/html
   false indica que NO es descarga */
 });
 server.serveStatic("/", LittleFS, "/"); /*  Cuando el archivo HTML
se cargue en el navegador, solicitará el CSS, archivo JavaScript y favicon.
Estos son archivos estáticos guardados en el mismo directorio
(SPIFFS o LittleFS). Entonces, podemos simplemente agregar la siguiente
línea para servir archivos en un directorio cuando sea solicitado por
la URL raíz. Servirá los archivos CSS y favicon automáticamente
(también en ocasiones agregamos una imagen) */

 server.begin(); // inicia el servidor
}
void loop() {
 ws.cleanupClients(); /* Los navegadores a veces no cierran correctamente
 la conexión WebSocket, incluso cuando se llama a la función close () en
 JavaScript. Esto eventualmente agotará los recursos del servidor web y hará
 que el servidor se bloquee. Llamar periódicamente a la función
 cleanupClients () desde el bucle principal () limita el número de clientes
  cerrando el cliente más antiguo cuando se ha superado el número máximo
  de clientes. Esto se puede llamar en cada ciclo, sin embargo, si desea
  usar menos energía, entonces es suficiente llamar con tan poca frecuencia
  como una vez por segundo. */
}
```

ARCHIVO index.html

```
<!DOCTYPE html>
<html>
<head>
 <title>Salidas con Websockets</title>
 <meta name="viewport" content="width=device-width, initial-scale=1">
 <meta charset="UTF-8"/>
 <link rel="stylesheet" type="text/css" href="style.css">
 <link rel="icon" type="image/png" href="favicon.png">
 <link rel="stylesheet" href="https://use.fontawesome.com/releases/v5.7.2/css/al
l.css" integrity="sha384-
fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr" crossorigin="an
onymous">
</head>
<body>
 <div class="marquesina">
 <h1>SERVIDOR WEB MULTIPLES SALIDAS MEDIANTE WEBSOCKETS</h1>
 </div>
 <div class="contenido">
 <div class="cuadricula-tarjeta">
 <div class="tarjeta">
 <p class="titulo-tarjeta"><i class="fas fa-lightbulb"></i> GPIO 4</p>
 <label class="switch">
 <input type="checkbox" onchange="toggleCheckbox(this)" id="4">
 <span class="deslizador"></span>
 </label>
 <p class="estado">Estado: <span id="4s"></span></p>
 </div>
 <div class="tarjeta">
 <p class="titulo-tarjeta"><i class="fas fa-lightbulb"></i> GPIO 5 (Pin 3 Wemos D1)</p>
 <label class="switch">
 <input type="checkbox" onchange="toggleCheckbox(this)" id="5">
 <span class="deslizador"></span>
 </label>
 <p class="estado">Estado: <span id="5s"></span></p>
 </div>
 <div class="tarjeta">
 <p class="titulo-tarjeta"><i class="fas fa-lightbulb"></i> GPIO 12</p>
 <label class="switch">
 <input type="checkbox" onchange="toggleCheckbox(this)" id="12">
 <span class="deslizador"></span>
 </label>
 <p class="estado">Estado: <span id="12s"></span></p>
 </div>
 <div class="tarjeta">
 <p class="titulo-tarjeta"><i class="fas fa-lightbulb"></i> GPIO 13 (pin 7 Wemos D1)</p>
 <label class="switch">
 <input type="checkbox" onchange="toggleCheckbox(this)" id="13">
```

```
 <span class="deslizador"></span>
 </label>
 <p class="estado">Estado: <span id="13s"></span></p>
 </div>
 </div>
 </div>
<script src="script.js"></script>
</body>
</html>
```

ARCHIVO style.css

```
html { /* observa que los elementos html no tienen punto antes */
 font-family: Arial, Helvetica, sans-serif;
 text-align: center;
}
h1 { /* observa que los elementos html no tienen punto antes */
 font-size: 1.8rem;
 color: white;
}
.marquesina { /* es la barra de arriba */
 overflow: hidden;
 background-color: #0A1128;
}
body { /* observa que los elementos html no tienen punto antes */
 margin: 0;
}
.contenido { /* es el contenido de cada tarjeta */
 padding: 50px;
}
.cuadricula-tarjeta {  /* dimensiones y propiedades de la tarjeta */
 max-width: 600px;
 margin: 0 auto;
  display: grid;
 gap: 2rem;
 grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
}
.tarjeta {
 background-color: white;
 box-shadow: 2px 2px 12px 1px rgba(140,140,140,.5);
}
.titulo-tarjeta { /*propiedades de cada tarjeta */
 font-size: 1.2rem;
 font-weight: bold;
 color: #034078
}
.estado { /* propiedades del mensaje de estado de los GPIO */
 font-size: 1.2rem;
```

```
 color: #1282A2;
}
.switch { /* propiedades de cada switch */
 position: relative;
 display: inline-block;
 width: 120px;
 height: 68px
}
.switch input { /* cuando "no se ve" el switch */
 display: none
}
.deslizador {
 position: absolute;
 top: 0; left: 0; right: 0; bottom: 0;
 background-color: #ccc;
 border-radius: 50px
}
.deslizador:before {
 position: absolute;
 content: "";
 height: 52px;
 width: 52px;
 left: 8px;
 bottom: 8px;
 background-color: #fff;
 -webkit-transition: .4s;
 transition: .4s;
 border-radius: 50px;
}
input:checked+.deslizador {  /* color al deslizar ENCENDIDO */
 background-color: #b30000;
}
input:checked+.deslizador:before {
 -webkit-transform: translateX(52px);
 -ms-transform: translateX(52px);
 transform: translateX(52px);
}
```

ARCHIVO script.js

```
// La puerta de enlace es el punto de entrada a la interfaz de WebSocket.
var PuertaDeEnlace = `ws://${window.location.hostname}/ws`;
/*
window.location.hostname obtiene la dirección de la página actual (la
dirección IP del servidor web). */
var miWebSocket; // Crea una nueva variable global llamada miWebSsocket
function onLoad(event)
 {
```

```
  /*  Agrega un detector de eventos que llamará la
     función onload cuando se cargue la página web. */
initWebSocket(); // inicializa WebSocket
}
function initWebSocket()
{ /* La función initWebSocket () inicializa una conexión WebSocket
    en la puerta de enlace definido anteriormente PuertaDeEnlace.
    También asignamos varias funciones de devolución de llamada
    que se activarán cuando la conexión WebSocket se abra, se cierre
    o cuando se recibe un mensaje.
  */
 console.log('Intentando abrir una conexión WebSocket...');
 miWebSocket = new WebSocket(PuertaDeEnlace);
 miWebSocket.onopen = onOpen;
 miWebSocket.onclose = onClose;
 miWebSocket.onmessage = onMessage;
}

function onOpen(event)
 {
 console.log('Conexión abierta'); /* Cuando se abre la conexión, presenta
 un mensaje en la consola para propósitos de depuración y envía un mensaje
 que diga "estados",  para que el servidor sepa que necesita enviar los
 estados GPIO actuales. */
 miWebSocket.send("estados");
}

function onClose(event) { /* Si por alguna razón la conexión del
  WebSocket está cerrada, llama a initWebSocket () volverá a funcionar
  después de 2000 milisegundos (2 segundos). */
 console.log('Conexión cerrada');
 setTimeout(initWebSocket, 2000);
}
function onMessage(event)
{  /* Finalmente, necesitamos manejar lo que sucede cuando el cliente
    recibe un nuevo mensaje (evento onMessage). El servidor (Wemos D1 R1)
    enviará una variable JSON con los estados GPIO actuales en el siguiente
    formato:

    {
"gpios":[
{
"salida":"3",
"estado":"0"
},
{
"salida":"4",
"estado":"0"
},
```

```
{
"salida":"12",
"estado":"0"
},
{
"salida":"13",
"estado":"0"
}
]
}

Esto puede observarse en la consola si al momento de estar funcionando el servidor
tecleamos Ctrol-Shift-J
    */

var miObjetoJSON = JSON.parse(event.data);
/* Puede obtener la respuesta del servidor como una cadena de JavaScript
 utilizando la propiedad event.data (porque recibe los datos de un evento).
  La respuesta viene en formato JSON, entonces,podemos guardar la respuesta
  como un objeto JSON usando el método JSON.parse () como este:
*/
 console.log(miObjetoJSON); /* Para fines de depuración, puede mostrar
 el valor de la variable JSON miObjetoJSON en la consola usando
 console.log ().
 */
 for (i in miObjetoJSON.gpios){ /*
    Ahora, necesitamos un bucle for para pasar por todas las salidas
     y los estados correspondientes.
    */
 var salida = miObjetoJSON.gpios[i].salida;
 var estado = miObjetoJSON.gpios[i].estado;
 console.log(salida); // se presenta salida en consola
 console.log(estado); // se presenta estado en consola
/* analizando las vueltas del ciclo for:
En vuelta 0:
• miObjetoJSON.gpios [0].output devuelve 4
• miObjetoJSON.gpios [0].estado devuelve 0 (al inicio)
El bucle for pasa por todos los objetos dentro de la matriz gpios,
obtiene el GPIO y los estados correspondientes y los guarda en la
salida y el estado de variables JavaScript
En vuelta 1:
• miObjetoJSON.gpios [1].output devuelve 5
• miObjetoJSON.gpios [1].estado devuelve 0 (al inicio)
En vuelta 2:
• miObjetoJSON.gpios [2].output devuelve 12
• miObjetoJSON.gpios [2].estado devuelve 0 (al inicio)
En vuelta 3:
• miObjetoJSON.gpios [3].output devuelve 13
• miObjetoJSON.gpios [3].estado devuelve 0 (al inicio)
```

```
Los estados van cambiando de valor a 1 o regresan a 0
a medida que se mueven los deslizadores en la interface
gráfica de control
*/

 if (estado == "1"){ /* Se evalúa para id = "3", "4", "12", "13"
 y actualiza el estado correspondiente a ENCENDIDO o APAGADO y se
 establezca si el control deslizante queda en una posición
 activada o no  */
 document.getElementById(salida).checked = true;
 // Al "checkar" resultó 1 ?
 document.getElementById(salida+"s").innerHTML = "ENCENDIDO";
 /* También necesitamos actualizar el texto del estado a ENCENDIDO.
 Obtener el elemento con id = "4s", "5s", "12s", "13s" (cada salida + "s")
 y actualizar el texto  a ENCENDIDO.  */
 }
 else{
 document.getElementById(salida).checked = false;
 // Al "checkar" resultó 0
 document.getElementById(salida+"s").innerHTML = "APAGADO";
 /* También necesitamos actualizar el texto del estado a APAGADO.
 Obtener el elemento con id = "4s", "5s", "12s", "13s" (cada salida + "s")
 y actualizar el texto  a APAGADO.  */
 }
 }
 console.log(event.data); //visualizar con la consola
 }
// Envía peticiones para controlar GPIOs
function toggleCheckbox (element)
 { /* La función toggleCheckBox () envía un mensaje usando la conexión
    WebSocket cada vez que se activa un interruptor en la página web.
    El mensaje contiene el número GPIO que queremos controlar */
 console.log(element.id); /*  (element.id corresponde a la identificación
    del interruptor deslizante que corresponde al número GPIO) */
 miWebSocket.send(element.id); // envía el elemento.id
 /*Además, también actualizamos el estado actual de GPIO en la página web: */
 if (element.checked){
 document.getElementById(element.id+"s").innerHTML = "ENCENDIDO";
 }
 else {
 document.getElementById(element.id+"s").innerHTML = "APAGADO";
 }
}
/* Entonces, la Wemos D1 debería manejar lo que sucede cuando recibe estos
mensajes: active o desactive los GPIO correspondientes y notifique a todos
los clientes. */
window.addEventListener('load', onLoad);
```

ARCHIVO platformio.ini

```
[env:d1]
platform = espressif8266
board = d1
framework = arduino
monitor_speed = 115200
lib_deps =
        ESP Async WebServer
        arduino-libraries/Arduino_JSON@^0.1.0
        ottowinter/ESPAsyncTCP-esphome@^1.2.3
        ottowinter/ESPAsyncWebServer-esphome@^1.2.7
board_build.filesystem = littlefs
```

CONEXIONES:

Conectar cuatro LEDs de cualquier color en serie con resistores de 220 Ohms y conexión a tierra. La parte del ánodo (positiva) debe conectarse a los siguientes GPIOs de la tarjeta Wemos D1 R1

/D4 para GPIO 4
/D3 para GPIO 5
/D12 o /D6 para GPIO12
/D7 para GPIO13

PRUEBAS:

Al probar el servidor se solicita que otros clientes se conecten mediante su navegador a la misma IP y hagan cambios de posición en los interruptores deslizables. Al realizar los cambios, todos los clientes deberán observar en su interface gráfica de control una actualización en los estados de cada interruptor deslizable.

Al activar la consola con Ctrol-Shift-J pueden observarse los cambios de manera actualizada: