

# PRÁCTICA MONGOOSE

ALUMNOS:

BASTIDA PRADO JAIME ARMANDO

ORTÍZ RODRÍGUEZ SALVADOR ALEJANDRO

SÁNCHEZ CUEVAS ESTEBAN

PROFESOR: UKRANIO CORONILLA CONTRERAS

GRUPO: 4CM4

Mayo 2020

# Índice

<b>1. INTRODUCCIÓN</b>	<b>3</b>
<b>2. DESARROLLO DE LA PRÁCTICA</b>	<b>3</b>
2.1. EJEMPLO . . . . .	3
<b>3. EJERCICIO 1</b>	<b>4</b>
3.1. Desarrollo . . . . .	4
3.2. Haciendo las pruebas . . . . .	7
<b>4. CONCLUSIONES</b>	<b>9</b>

# 1. INTRODUCCIÓN

Para ésta práctica se buscó aprender a usar Mongoose para proporcionar un servidor web embebido a nuestra aplicación de Elecciones Presidenciales y así ofrecer una interfaz amigable a los usuarios.

Primero se probó el ejemplo dado por el profesor, de un simple servidor HTTP que sirve páginas web solicitadas por algún cliente como un navegador e.g. (Google, Firefox, etc).

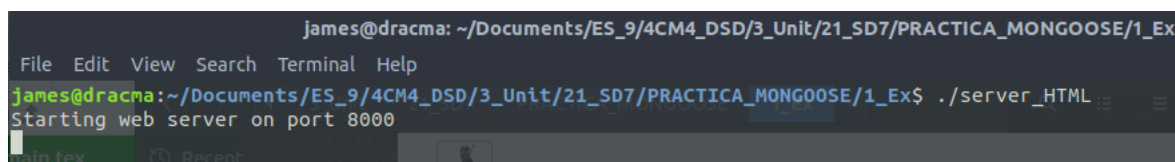
Después, se analizó el código de un servicio REST sencillo, y se procedió a elaborar el Ejercicio 1.

## 2. DESARROLLO DE LA PRÁCTICA

### 2.1. EJEMPLO

Procedimos a implementar el código del servidor HTTP de ejemplo, y luego lo ejecutamos e hicimos la siguiente prueba.

Corriendo el servidor:



```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/PRACTICA_MONGOOSE/1_Ex
File Edit View Search Terminal Help
james@dracma:~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/PRACTICA_MONGOOSE/1_Ex$ ./server_HTML
Starting web server on port 8000
```

Figura 1: Servidor corriendo

Abrimos un navegador web para utilizarlo como cliente y probamos que el servidor se encontrase ejecutando:

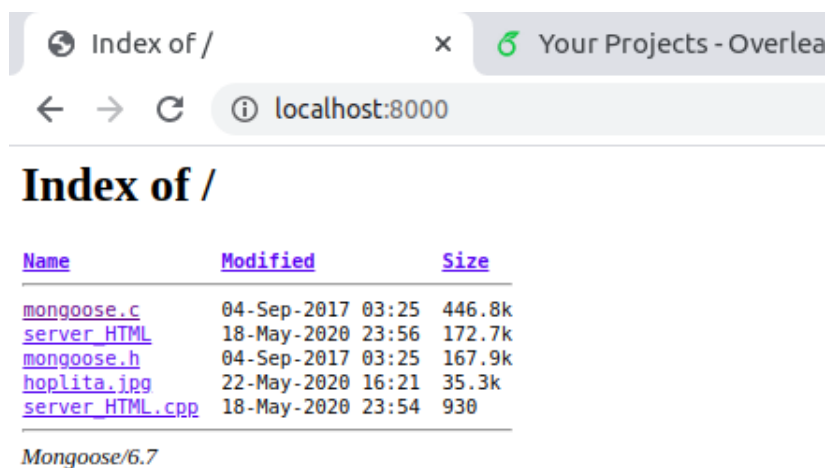


Figura 2: Navegador web

Pedimos que nos sea mostrado el recurso dentro del directorio del servidor "hoplita.jpg":



Figura 3: Ejemplo de recurso servidor por el servidor HTTP

### 3. EJERCICIO 1

Reutilizando el código del servicio REST sencillo y haciendo distintas pruebas con los 4 ejemplos "connected\_device" que encontramos en la página de github de la documentación de Mongoose, procedimos a elaborar la implementación solicitada.

#### 3.1. Desarrollo

Para el desarrollo de esta práctica creamos un sitio web utilizando jQuery para hacer uso de Ajax y para hacer la página dinámica, y para la vista utilizamos Materialize para poder crear una página agradable visualmente y responsiva.

En el archivo "index.html" tenemos un formulario donde hay un input de nombre "ip\_broadcast", que es donde el usuario ingresará la dirección IP de Broadcast para hacer la búsqueda de servidores:

```
<form class="col s12" id="formBuscar" autocomplete="off">
  <div class="row">
    <div class="col s12 input-field">
      <i class="material-icons prefix">wifi_tethering</i>
      <label for="ip_broadcast">IP Broadcast: </label>
      <input type="text" id="ip_broadcast" name="ip_broadcast">
    </div>
  </div>
</form>
```

Figura 4: Parte del formulario donde el usuario ingresa la IP

En el archivo "index.js" cada que el usuario hace clic en el botón de búsqueda, se toma el valor que tiene el campo input con la IP de Broadcast y es enviada hacia el servidor web, quien nos regresará un objeto JSON que contiene las direcciones IP y los tiempos de respuesta de cada servidor. Entonces se itera sobre cada dirección y se agregan dichos datos a la página web dentro de un div que tiene el id "servers\_info\_container":

```
let servers_info_container = document.getElementById("servers_info_container");
servers_info_container.innerHTML = "";
$.ajax(
{
  type: "POST",
  url: "/search",
  data:{ip_broadcast: $("#ip_broadcast").val()},
  success: function(respAx)
  {
    let respuesta = JSON.parse(respAx);
    let i;

    for(i = 0; i < respuesta.ips.length; i++)
    {
      // console.log(respuesta.ips[i]);
      let new_server = document.createElement("h6");
      let new_server_text = document.createTextNode("IP: " + respuesta.ips[i]);
      new_server.appendChild(new_server_text);
      servers_info_container.appendChild(new_server);
    }
    $("#servers_info_container").show();
  }
});
return false;
```

Figura 5: Código dentro de index.js

En el archivo "server.cpp" programamos el servidor web, e implementamos la función "ev\_handler()" de la siguiente manera, checando que si el cliente busca que le sea servido un recurso o si está haciendo la búsqueda de servidores, en cuyo caso se llama a la función "handle\_search\_services()":

```
99 static void ev_handler(struct mg_connection *nc, int ev, void *ev_data)
100 {
101     struct http_message *hm = (struct http_message *) ev_data;
102
103     // CHECANDO QUE TIPO DE EVENTO OCURRIÓ
104     switch(ev)
105     {
106         case MG_EV_HTTP_REQUEST:
107             // SI SE TRATA DE UN EVENTO HTTP, DEBEMOS CHECAR SI ES UNA PETICIÓN REST HACIA EL SERVIDOR
108             // O SI SE ESTÁ PIDIENDO DEVOLVER ALGÚN ARCHIVO DENTRO DEL DIRECTORIO DEL SERVIDOR
109             if(mg_vcmp(&hm->uri, "/search") == 0)
110                 handle_search_services(nc, hm); // El usuario desea buscar los servicios
111             else
112                 mg_serve_http(nc, (struct http_message *) ev_data, s_http_server_opts);
113             break;
114             default:
115                 break;
116         }
117     }
```

Figura 6: Código dentro de la función ev\_handler()

La función "handle\_search\_services()" se encarga de crear un SocketDatagrama() y lo configura con la función "setBroadcast()", para que pueda hacer envío de un mensaje broadcast a la IP especificada:

```
static void handle_search_services(struct mg_connection *nc, struct http_message *hm)
{
    SocketDatagrama socket_datagrama(0);           // Abrimos un socket para enviar un mensaje broadcast
    socket_datagrama.setBroadcast();               // Habilitamos el envío de mensajes broadcast
    char ip_broadcast[17];                        // Var donde almacenaremos la IP enviada por el usuario por medio de HTTP
}
```

Figura 7: Creación del socket para broadcast

Después de eso se encarga de ir recibiendo las respuestas al broadcast y almacenando en formato JSON las direcciones IP y los tiempos de respuesta de cada servidor:

```
printf("Esperando respuesta(s) al broadcast...\n");
start = high_resolution_clock::now();
while(socket_datagrama.recibeTimeout(paquete_recepcion, 2, 0) > 0)
{
    // TOMAMOS EL TIEMPO/MOMENTO EN QUE EL SERVIDOR RESPONDIÓ
    auto stop = high_resolution_clock::now();
    // SE ALMACENAN EL NÚMERO DE MILLISEGUNDOS QUE TARDÓ EN CONTESTAR
    elapsed_time_ms = duration<double, milli>(stop - start).count();

    printf("-----\n");
    printf("IP: %s\n", paquete_recepcion.obtieneDireccion());
    printf("Puerto: %d\n", paquete_recepcion.obtienePuerto());
    resultado = *(int *) paquete_recepcion.obtieneDatos();
    printf("Resultado: %d\n", resultado);

    // PARA LA PRIMER RESPUESTA EL FORMATO ES LIGERAMENTE DIFERENTE, PUES NO LLEVA UNA COMA
    if(firstTime)
    {
        sprintf(ips + strlen(ips), "\"%s\"", paquete_recepcion.obtieneDireccion());
        sprintf(tiempos_respuesta + strlen(tiempos_respuesta), "%lf", elapsed_time_ms);
        firstTime = false;
    }
    else
    {
        sprintf(ips + strlen(ips), ", \"%s\"", paquete_recepcion.obtieneDireccion());
        sprintf(tiempos_respuesta + strlen(tiempos_respuesta), ", %lf", elapsed_time_ms);
    }
    // VOLVEMOS A TOMAR EL TIEMPO PARA CALCULAR LO QUE TARDA EL SIGUIENTE SERVIDOR EN RESPONDER
    start = high_resolution_clock::now();
}
}
```

Figura 8: Recibiendo respuestas al mensaje broadcast

Finalmente se envía el objeto JSON como respuesta al cliente:

```
87 // Use chunked encoding in order to avoid calculating Content-Length
88 mg_printf(nc, "%s", "HTTP/1.1 200 OK\r\nTransfer-Encoding: chunked\r\n\r\n");
89 // Output JSON object
90 mg_printf_http_chunk(nc, "{ \"ips\": %s, \"tiempos\": %s }", ips, tiempos_respuesta);
91 // Send empty chunk, the end of response
92 mg_send_http_chunk(nc, "", 0);
93 }
```

Figura 9: Enviando JSON

### 3.2. Haciendo las pruebas

Hicimos las pruebas corriendo 3 servidores del capítulo 9 del manual, de manera remota cada uno usando hamachi:  
Servidor:

```
james@dracma: ~/Documents/ES_9/4CM4_DSD/3_Unit/21_SD7/PRACTICA_MONGOOSE/1_Ej$ ./server
File Edit View Search Terminal Help
Starting web server on port 8000
```

Figura 10: Servidor corriendo

Entramos al navegador web y buscamos "localhost:8000/":

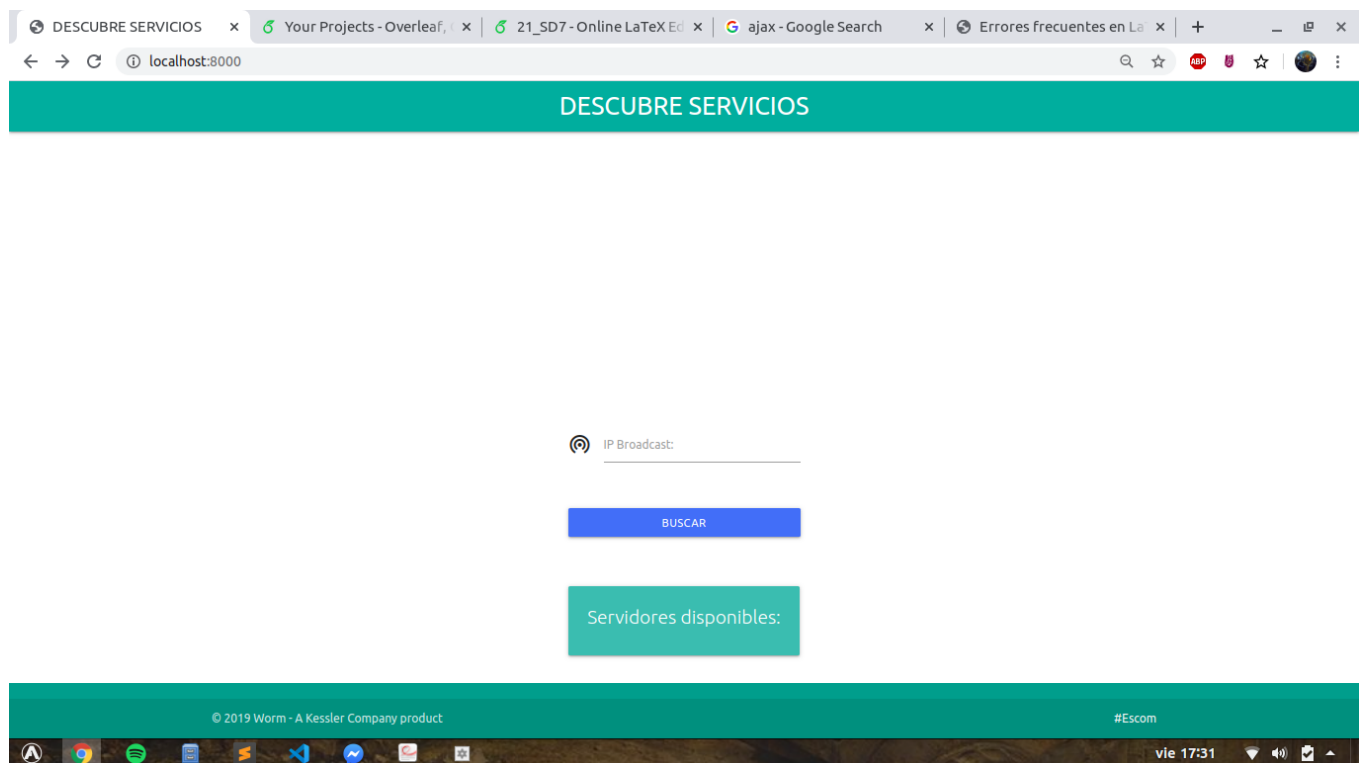


Figura 11: Página web

Al ingresar la IP de Broadcast 25.255.255.255 como ejemplo, obtenemos la respuesta a todos los servidores corriendo en la red VPN, que en este caso son ejecutando 3, justo como en el ejemplo de la propia práctica (Diseño Web) pero ahora con la página web:

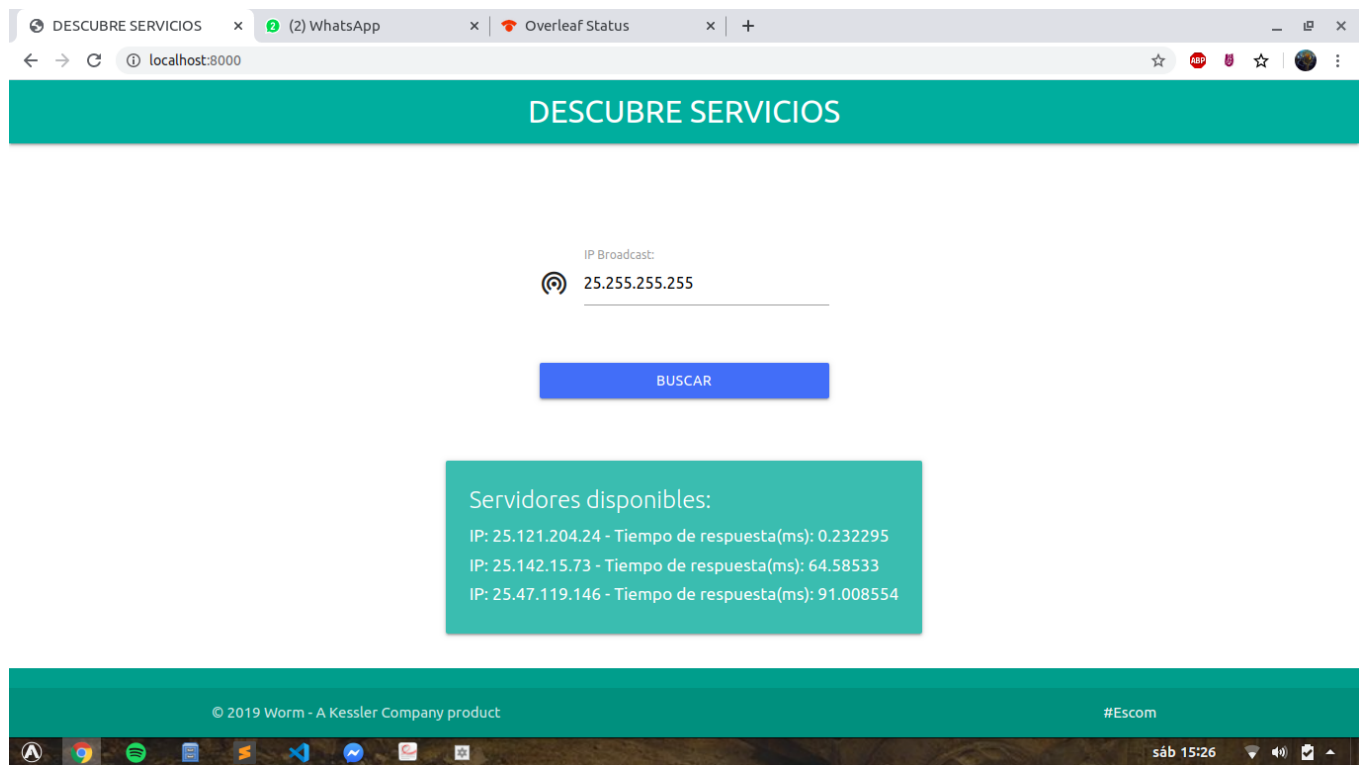


Figura 12: Resultado de búsqueda



## 4. CONCLUSIONES

El desarrollo de esta práctica se dió de forma bastante didáctica gracias a toda la documentación proporcionada por los creadores de Mongoose y la variedad de ejemplos en su GitHub, al ir checándolos pudimos ir aprendiendo como es que se puede implementar un servidor web emebebido en nuestro sistema distribuido de Elecciones Presidenciales.

Sin duda, Mongoose es una gran y práctica herramienta para cuando se desea implementar un servidor web, gracias a que nos permite programar en lenguaje C o C++ lo cual le da a nuestras aplicaciones un mejor y más eficiente tiempo de respuesta, claro sabiéndolo programar.