

ICC343 - Programación Avanzada 2021-2

Tutorial 7 - HTTPRequest y JSON en PHP

1 (25%) Proceso PHP usando JSON

El primer problema se trata de replicar un programa PHP, llamado *conectados.php* que realiza un proceso de agregar o eliminar personas conectadas a una plataforma. El programa PHP responde a la información dada en el URL, las variables que el URL contiene son las siguientes:

accion. La variable *accion* puede tener uno de los siguientes valores:

- | | |
|---------------|---|
| 'agrega'. | Lo que significa que el nombre asociado debe agregarse a la lista de conectados. Cada conectado tiene un color asociado, de este modo, al agregar la persona como conectado también se agrega su color, información que también debe estar en la URL. |
| 'desconecta'. | Lo que significa que el nombre asociado debe ser eliminado de la lista de conectados del servidor. |
| 'todos'. | Lo que significa que se está pidiendo un listado de todos los conectados |

nombre. La variable *nombre* debería contener el nombre de la persona a conectarse o desconectarse

color. La variable *color* traerá el color hexadecimal (RGB) que identificará a la persona conectada que se conectará

El código del programa *conectados.php* se muestra y explica por partes (note la continuidad de los números de línea de cada fragmento explicado. Comenzamos.

```
1  <?php
2  $accion = $_GET['accion'];
3  $color="";
4  $nombre="";
5
6  if(isset($_GET['nombre'])){
7      $nombre= $_GET['nombre'];
8  }
9  if(isset($_GET['color'])) {
10     $color = $_GET['color'];
11 }
12
```

Este fragmento recupera tres variables desde el URL, o desde un formulario HTML con método GET. Recupera siempre la variable **accion**, y en el caso de las variables **nombre** y **color**, las

recupera sólo si forman parte del URL. La función `isset()` pregunta por la existencia de la variable, y cuando existe nombre y color se asignan a variables php. Sino quedan vacías.

```
13  if($accion=='agrega' && $color!='' && $nombre!='') {
14      $con = arreglo_conectados();
15      $neo['nombre']=$nombre;
16      $neo['color']=$color;
17      array_push($con,$neo);
18      $resul = graba_conectados($con);
19      if($resul) {
20          $ret['ok']='yes';
21          $ret['error']='';
22      }
23      else {
24          $ret['ok']='no';
25          $ret['error']='error desconocido al agregar';
26      }
27      echo json_encode($ret);
28      return;
29  }
```

En este fragmento se implementa el caso que **acción** sea **agrega**. Esta acción se realiza sólo cuando efectivamente venga un nombre y un color, por eso la pregunta de la línea 13 se asegura que las correspondientes variables no están vacías (\$nombre y \$color).

En la línea 14 se invoca la función `arreglo_conectados()`, que recupera en un listado de arreglos asociativos de todos los conectados. Luego con las variables \$nombre y \$color se ponen en un arreglo asociativo que es agregado como elemento del arreglo mediante la función `graba_conectados()` - línea 18 -.

Desde la línea 19 a 26 se implementan las dos opciones del resultado de grabar la nueva lista de conectados. En realidad, no hay razones de software para que esto salga mal, pero por razones de hardware sí, como desconexión de la red, falta de disco, sobrepasar el límite de memoria RAM entre otros. En ambos casos se construye un arreglo asociativo, ya sea reportando que la grabación fue un éxito (caso if, línea 19), o que la grabación tuvo problemas, caso else - línea 23 -.

En la línea 27 el arreglo asociativo de respuesta (\$ret) es codificado en json y es escrito (echo). Si la invocación fuese un formulario este escrito sería la página de respuesta, pero este tutorial es del objeto XMLHttpRequest de JS, es decir, será un PHP que la responde a un programa javascript. El `return` de la línea 28 hace que el programa termine su ejecución de manera normal.


```
30 else if($accion=='agrega' && ($color==' or $nombre==')) {  
31     $ret['ok']='no';  
32     $ret['error']='faltan parámetros';  
33     echo json_encode($ret);  
34     return;  
35 }
```

En este fragmento (línea 30 a 35), simplemente se genera el mensaje de respuesta si fuese el caso que se ha optado por la acción **agrega**, pero que no vienen todos los parámetros esperados en la línea del URL (\$nombre y \$color). Igual que antes, el arreglo asociativo de respuesta se codifica en formato json.

A esta altura usted todavía no puede ejecutar el programa, pero para que entienda lo que se está explicando ejecutaré el programa con el siguiente URL:

localhost/conectados.php?accion=agrega

El navegador mostrará



```
{ "ok": "no", "error": "faltan par\u00e1metros" }
```

(la “á” sale con su código unicode)

Continuamos.

```

36  else if($accion=="desconecta") {
37      $con = arreglo_conectados();
38      for($k=0 ; $k<sizeof($con) ; $k++) {
39          if($nombre==$con[$k]['nombre']) {
40              unset($con[$k]);
41              break;
42          }
43      }
44      $resul = graba_conectados($con);
45      if($resul) {
46          $ret['ok']='yes';
47          $ret['error']='';
48      }
49      else {
50          $ret['ok']='no';
51          $ret['error']='error desconocido al desconectar';
52      }
53      echo json_encode($ret);
54      return;
55  }

```

En este fragmento de código se implementa la opción de desconexión. Igual que antes se cargan los conectados en un listado de arreglos asociativos. En el for (línea 38 a 43) se recorre el listado buscando el nombre a eliminar, cuando se encuentra (if de línea 39) se elimina del arreglo ese elemento (unset es una función php que extrae el elemento de un arreglo de cualquier tipo. Las líneas 45 a 52, igual que antes (línea 19 a 26), para preparar la estructura de la respuesta. Las líneas 53 y 54, empaquetan la respuesta como json y finaliza el programa.

```

56  else if ($accion="todos") {
57      $con = arreglo_conectados();
58      $ret['ok']='yes';
59      $ret['error']='';
60      $ret['conectados']=$con;
61      echo json_encode($ret);
62      return;
63  }
64

```

De la línea 56 a 63 se implementa la respuesta de todos. En este caso se carga el arreglo con todos los conectados como un componente de la respuesta (componente "conectados" del arreglo asociativo, lo que se empaqueta json y termina la respuesta si la acción fue **todos**.

El programa termina con las funciones usadas anteriormente, es decir, graba_conectados() y arreglo_conectados() implementadas de la siguiente manera

Función graba_conectados.

```
64
65 function graba_conectados($con) {
66     $archi = fopen("conectados.txt","w");
67     if(!$archi)
68         return false;
69     foreach($con as $conectado){
70         fwrite($archi,$conectado['nombre'].",".$conectado['color'].PHP_EOL);
71     }
72     fclose($archi);
73     return true;
74 }
75
```

Función arreglo_conectados

```
75
76 function arreglo_conectados() {
77     if(!file_exists("conectados.txt")) {
78         return [];
79     }
80     $archi = fopen("conectados.txt","r");
81     $ret = [];
82     $k=0;
83     while( !feof($archi)) {
84         $campo = explode(",",str_replace(PHP_EOL,'',fgets($archi)));
85         if(sizeof($campo)==2) {
86             $ret[$k]['nombre']=$campo[0];
87             $ret[$k]['color']=$campo[1];
88             $k++;
89         }
90     }
91     fclose($archi);
92     return $ret;
93 }
```

Este ejemplo, para que se ejecute, requiere que el administrador web tenga los privilegios sobre el archivo conectados.php. El programa puede que no tenga ningún problema de privilegios en Windows (DCI no da soporte Windows en sus clases), pero en el caso de Linux (Ubuntu, Xubuntu) lo más práctico es crear el archivo conectados.txt, en el mismo directorio del archivo conectados.php y permitir que el usuario webserver (a veces apache a veces www-data) tenga acceso de escritura a dicho archivo. (El comando chown cambia el propietario del archivo).

Para que tenga el porcentaje de este problema debe ejecutar con las siguientes URL

```
suservidor/conectados.php?accion=todos
```

La respuesta debería ser:

```
{"ok":"yes","error":"","conectados":[]}
```

Luego agregue tres nombre ejecutando el URL 3 veces con 3 nombres y 3 colores, por ejemplo:

```
suservidor/conectados.php?accion=agrega&nombre=pedro&color=90f280
```

Use dos nombres más y dos colores diferentes más y tome una fotografía del resultado de ejecutar la acción ***todos***, nuevamente.

Reporte, en el canal de Slack del curso, usando las etiquetas provistas (Tu-7-1X), igual que en los tutoriales anteriores, es decir, con fondo de pantalla y reloj EN EL MOMENTO de terminar el problema, los siguientes ítems:

Tu-7-1A	Código fuente del php copiado
Tu-7-1B	Invocación y respuesta inicial de llamada con 'todos'
Tu-7-1C	3 invocaciones y respuestas de agrega con datos de su elección
Tu-7-1D	Nueva invocación y respuesta de llamada con 'todos'

2 (15%) Agregaciones usando formularios HTML

Para poder realizar esta parte del tutorial se requiere que haya completado y funcionado la fase anterior. En este problema SIMPLEMENTE, invocaremos el programa php anterior por medio de un archivo html. El archivo a implementar que se pide debe llamarse `agregaConectado.html` y su contenido debe ser el siguiente:

```
1  <html>
2  <head>
3  <title>Ejemplo llamada conectados</title>
4  </head>
5  <body>
6      <form action="conectados.php" method="GET">
7          Nombre: <input type="text" name="nombre"><br>
8          Color: <input type="text" name="color"><br>
9          <input type="hidden" name="accion" value="agrega">
10         <input type="submit" value="Agregar Conectado">
11     </form>
12 </body>
13 </html>
```

En este fragmento de código hay un formulario HTML que invoca el anterior programa `conectados.php` (atributo **action** en la línea 6). Como ya se dijo en método debe ser GET para que la llamada sea visible en el URL (esto no es muy seguro pero para el tutorial lo hacemos así. Como alternativa tenemos el método POST, pero ello requiere cambiar el programa `conectados.php`, y en lugar de usar el arreglo `$_GET`, se debe usar el arreglo `$_POST`).

El otro elemento destacable de este fragmento son los campos de entrada (INPUT) **nombre** y **color**. El nombre DEBE coincidir con los nombres de las variables capturadas en el programa **conectados.php**. Este programa, usa además, una tercera variable que es la "accion". La acción no tiene sentido que sea visible en este ejemplo que, supuestamente, es para agregar, pero la variable es requerida, entonces generamos un campo invisible (hidden) en la línea 9, que tenga el nombre de la variable requerida (accion) con el valor requerido (agrega).

Copie el programa en el mismo directorio de `conectados.php` y úselo para agregar tres nombres más y tres colores diferentes más vía este formulario.

Reporte lo siguiente con fondo de pantalla y reloj EN EL MOMENTO de terminar el problema:

Tu-7-2A	Código fuente nuevo HTML
Tu-7-2B	3 Fotos del formulario con 3 nuevos ingresos.
Tu-7-2C	3 invocaciones y respuestas para los datos agregados
Tu-7-2D	Nueva invocación y respuesta json de llamada con 'todos' (para verificar inserción)

3.(30%) El Objeto XMLHttpRequest de PHP

En este problema usamos el objeto del tipo XMLHttpRequest, que es una clase de Javascript que implementa invocaciones asincrónicas, es decir, llamadas que pueden ejecutarse en paralelo, lo que es conocido como AJAX. En el caso de este ejemplo uso una función de creación propia (*ejecutaExterno*) que no necesita cambiarse de una aplicación a otra porque hace lo básico de una invocación a un URL externo, que puede contener variables en el URL, y tiene como parámetro el nombre de una función que se ejecute después de finalizado el proceso externo. Copie el siguiente programa con el nombre `conectadosActuales.html`

```
1 <html>
2 <head>
3 <script>
4 //Base AJAX. Generalización de Carlos Cares
5 function ejecutaExterno(id_doc_element, url_php, maximaEspera,sigueFunction) {
6     //url_php = encodeURIComponent(url_php);
7     var xhttp = new XMLHttpRequest();
8     var out= document.getElementById(id_doc_element);
9     out.innerHTML = "";
10    xhttp.timeout = maximaEspera;
11    xhttp.ontimeout = function(e) {
12        out.innerHTML = '{"error":"Proceso excede tiempo"}';
13    };
14    xhttp.onreadystatechange = function() {
15        if (this.readyState == 4 && this.status == 200) {
16            out.innerHTML = this.responseText;
17            sigueFunction();
18        }
19        else if (this.readyState == 4) {
20            var err =
21            out.innerHTML = '{"error":"inaccesible","status":"' + this.status + '", "url":"' + url_php + '"}';
22            sigueFunction();
23        }
24    };
25    xhttp.open("GET", url_php, true);
26    xhttp.send();
27 }
28
```



```

28
29  function actualizaConectados() {
30      var contenido = document.getElementById('conectados').innerHTML;
31      var llega = JSON.parse(contenido);
32      if(llega.error!='') {
33          alert(llega.error);
34      }
35      else if (llega.ok=='yes') {
36          var con = llega.conectados;
37          alert('llegaron '+con.length+' conectados');
38      }
39      return;
40  }
41
42  </script>
43  </head>
44  <body>
45      <h3>Conectados</h3>
46      <div id="conectados" style="display:"></div>
47      <div id="salidavisible"></div>
48      <script>
49          var url="conectados.php?accion=todos&hora="+new Date().getTime();
50          ejecutaExterno("conectados",url,2000,actualizaConectados);
51      </script>
52  </body>
53  </html>

```

Lo habitual es que las funciones javascript queden en el encabezado, por ese motivo lo primero que tenemos son estas dos funciones, *ejecutaExterno* y *actualizaConectados*. La primera ya fue explicada en términos generales y es básicamente genérica. En la línea 50 tenemos una invocación de la función *ejecutaExterno*, el primer parámetro es el nombre de un div donde quedará el resultado de la ejecución, el segundo parámetro es la url, en este caso he agregado un parámetro adicional que se llama hora. Este parámetro es un truco para evitar que el resultado quede en caché, porque, dado que la llamada es interna, y el url es el mismo, muchos navegadores podrían no enviar el URL al servidor, sino que, simplemente, traer lo que ya ejecutaron anteriormente desde su memoria interna (caché) lo que obviamente no provocará el resultado deseado. Con este “truco” el URL será diferente cada vez, “engañando” al administrador de memoria caché del navegador. El programa *conectados.php* no toma en cuenta este parámetro.

El tercer parámetro de la llamada *ejecutaExterno* es el tiempo de espera máximo al servidor en milisegundos, en este caso 2000 son dos segundos. El último parámetro es la función que ejecutará javascript después que este acceso a servidor termine. En este caso se ejecutará la función *actualizaConectados*.

La llamada a la función *actualizaConectados* lee el contenido del div “conectados” donde se ha dejado el resultado del proceso. Como sabemos que está en formato JSON, usamos el parser de JSON de javascript, para procesar lo que fue generado como JSON desde PHP. En este

caso el parser de JSON crea un objeto con el contenido JSON, es decir, los atributos son atributos del objeto y los valores son los valores expresados en JSON, si alguno de estos valores es un listado (array) entonces queda como un array, y si el contenido original era un arreglo asociativo en PHP, entonces en Javascript queda como un arreglo de objetos con los atributos correspondientes como si fuesen objetos almacenados en cada posición del arreglo.

En el caso de este paso del tutorial simplemente pedimos ejecutar el programa indicado. El resultado debería ser simplemente una ventana de alerta con la cantidad de conectados informadas por un mensaje de alerta -línea 37-. Si el proceso tuvo un error, ya sea generado por el propio programa PHP o por la función *ejecutaExterno* que no tuvo acceso al programa, el proceso no varía, porque en ambos casos se lee el atributo "error" del JSON recibido. En el caso de este programa dicha advertencia también se reporta como alerta -línea 33-.

Se han generado dos divs en las líneas 46 y 47. El div de la línea 46 es para almacenar el resultado, por ahora está visible; el div "salidavisible" contendrá (en el próximo paso) la salida visible de los conectados.

Ejecute el programa `conectadosActuales.html`, tome una foto de la salida, luego agregue con el programa anterior dos nombres más, y vuelva a ejecutarlo. EL mensaje de salida debería dar cuenta de los incrementos.

Reporte lo siguiente con fondo de pantalla y reloj EN EL MOMENTO de terminar el problema:

Tu-7-3A	Código fuente nuevo HTML
Tu-7-3B	3 Fotos del formulario con 3 nuevos ingresos.
Tu-7-3C	3 invocaciones y respuestas para los datos agregados
Tu-7-3D	Nueva invocación y respuesta json de llamada con 'todos' (para verificar inserción)

4.(30%) Salida gráfica del proceso

En este problema modificamos el programa conectadosActuales para que se reporte en una tabla los conectados actuales. La tabla tiene dos columnas, la primera para el nombre, y la segunda el color asociado al nombre de la persona conectada. El programa se debe llamar conectadosActuales2.html y queda de la siguiente forma.

La función ejecutaExterno no cambia así que comenzamos lo diferente desde la línea 29 con la función actualizaConectados. Queda así.

```
28
29  function actualizaConectados() {
30      var contenido = document.getElementById('conectados').innerHTML;
31      var llega = JSON.parse(contenido);
32      if(llega.error!='') {
33          alert(llega.error);
34      }
35      else if (llega.ok=='yes') {
36          muestraConectados(llega.conectados);
37      }
38      return;
39  }
40
```

Lo que hacemos es invocar una nueva función, pero ya con el arreglo de conectados pasado como parámetro (línea 36) que es la función que sigue a continuación.

```

41  function muestraConectados(conectado) {
42      var tabla = document.createElement("TABLE");
43      var fila = document.createElement("TR");
44      var celda = document.createElement("TD");
45      celda.innerHTML="Conectado";
46      fila.appendChild(celda);
47      celda = document.createElement("TD");
48      celda.innerHTML="Color";
49      celda.width = "20%";
50      fila.appendChild(celda);
51      tabla.appendChild(fila);
52      for(var k=0; k < conectado.length; k++) {
53          fila = document.createElement("TR");
54          celda = document.createElement("TD");
55          celda.innerHTML = conectado[k].nombre;
56          fila.appendChild(celda);
57          celda= document.createElement("TD");
58          celda.style.backgroundColor = '#' +conectado[k].color;
59          fila.appendChild(celda);
60          tabla.appendChild(fila);
61      }
62      var div = document.getElementById("salidavisible");
63      div.innerHTML = "";
64      div.appendChild(tabla);
65  }

```

Esta función crea el encabezado de la tabla (líneas 45 a 51) y crea el cuerpo de la tabla (líneas 52 a 61), iterando sobre el arreglo de conectados. Fíjese como se accede al nombre (línea 55) y al color (línea 58). Las funciones appendChild permiten agregar un nuevo hijo en la estructura del documento. Así que la tabla creada se agrega al div destinado para este despliegue (línea 64).

Lo que queremos en este programa, además de desplegar los conectados en una tabla, es que, dicha tabla, se refresque automáticamente, de modo de saber quién se conecta o desconecta sin necesidad de actualizar la página. La siguiente función hace esta labor.

```

66
67  function actualizaConectadosRecurrentemente(tiempo) {
68      var url="conectados.php?accion=todos&hora="+ (new Date()).getTime();
69      ejecutaExterno("conectados",url,2000,actualizaConectados);
70      setTimeout(actualizaConectadosRecurrentemente,tiempo,tiempo);
71  }
72
73

```

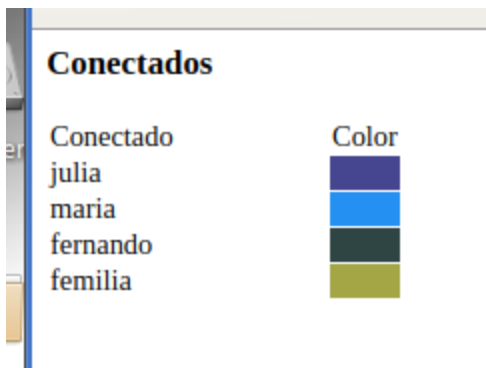
Las líneas 68 y 69 ya las usabamos en la versión anterior para llamar a la ejecución externa del php conectados.php. Pero ahora como la invocación será recurrente, implementamos esta función. En la tercera línea (línea 70) usamos una función ya explicada en otros tutoriales que permite una invocación asíncrona diferida en el tiempo, en este caso la función se llama a sí misma, en cantidad de “tiempo” milisegundos después. Como esta función usa parámetros, entonces, pasamos los parámetros que requiere de manera adicional, por eso el parámetro tiempo está dos veces, para que en la segunda invocación, también se utilice el mismo tiempo de retardo.

El resto del programa queda como se indica.

```
72
73
74 </script>
75 </head>
76 <body>
77     <h3>Conectados</h3>
78     <div id="conectados" style="display:none"></div>
79     <div id="salidavisible"></div>
80     <script>
81         actualizaConectadosRecurrentemente(5000);
82     </script>
83 </body>
84 </html>
```

Lo diferente de la versión anterior es que el div de recepción de la información del programa php externo está oculto y, la segunda diferencia, es que se invoca la función *actualizaConectadosRecurrentemente* para que se ejecute cada 5 segundos.

La salida de este programa debería verse como se indica.



Conectado	Color
julia	
maria	
fernando	
familia	

En OTRA pestaña ejecute el programa *agregaConectado.html*. Lo que debería suceder es que SIN NECESIDAD de refrescar, la pestaña actualizará la tabla con el nuevo conectado. Deje evidencia de su salida tomando fotografía antes y después de agregar dos nombres más a los ya agregados anteriormente. Para asegurarse que la invocación se está repitiendo elija un

lugar donde hacer un console.log de tal modo que pueda ver por la consola de javascript el hecho si están o no ocurriendo invocaciones recurrentemente.

Reporte lo siguiente con fondo de pantalla y reloj EN EL MOMENTO de terminar el problema:

Tu-7-4A	Código fuente completo (ambos archivos)
Tu-7-4B	Salida gráfica de nombres y conectados
Tu-7-4C	2 nuevas agregaciones y respuestas para los datos agregados
Tu-7-4D	Foto de la consola con las llamadas recurrentes.
Tu-7-4E	Nueva salida gráfica que incluye los agregados.