



## Computación orientada a la web: Documentación de los temas 6, 7 y 8

ALBERT ESPÍN ROMÁN

03.05.2018 | PROFESOR: SIMONE BALOCCO

## 1. Apartado 1

En este apartado se han adaptado los efectos y las funcionalidades implementadas en la entrega anterior para el cliente con Prototype y Scriptaculous: ahora en su lugar se usa JQuery y JQuery UI. Para respetar los principios de programación no obstructiva no se vinculan los eventos a los elementos HTML hasta que se carga la página: para ello se usa la función “\$(document).ready()”, en sustitución de “document.observe(“dom:loaded”)” de Prototype. JQuery ha permitido evitar el uso de “document.getElementById(‘id’)” con el uso de “\$(‘#id’)”. También se puede acceder fácilmente a los atributos de los elementos HTML con la función “attr(‘nombre-atributo’)”, que también se pueden modificar con la versión de esta misma función que recibe el valor a configurar como parámetro adicional.

Los eventos se asocian a los elementos con la misma política que con Prototype, pero con una sintaxis algo distinta. Por ejemplo, un campo del formulario es validado cuando deja de tener el foco (evento “blur”: ahora se requiere la construcción ‘field.on(“blur”, checkField)’, siendo “field” un campo y “checkField” su función de validación). Para cambiar el aspecto de los elementos dado un evento específico, como por ejemplo poner rojo un elemento que se determina incorrecto, se cambia la “class” del elemento HTML afectado (usando “removeClass” y “addClass” de JQuery) para que su visualización CSS varíe (en el caso mencionado, para que se muestre en rojo, cambiando los atributos “color” y “border-color” a “red”). Para maximizar la atención del usuario, se acompaña este cambio de varios efectos de JQuery UI. En particular, todo campo de textos que es considerado incorrecto por el validador vibra gracias a la función ‘effect(“shake”)’ (pasando opciones personalizadas de “duration”, “times” y “distance” menores que las que tiene por defecto, para que sea suave, no exagerado); también parpadea, gracias a ‘effect(“pulsate”)’ (se reduce el número de pulsos por parámetro, “times”, así como la duración, “duration”, y la opacidad inicial, “from” para conseguir nuevamente un efecto más sutil, no forzado). Adicionalmente, el mensaje de error aparece con la función “fadeIn(500)”.

Se ha experimentado con otras de las funcionalidades que Scriptaculous ya ponía al alcance del programador, ahora usando JQuery UI, se ha hecho que la lista creada en la página de estadísticas del sitio web pasara a ser reordenable, gracias a ser pasada como parámetro a la función “sortable()”. En esta lista también se han aplicado búsquedas múltiples en el DOM con JQuery, usando los filtros “li:even” y “li:odd” para colorear de forma diferente elementos pares e impares. En la página de las reservas, además, se ha reimplementado drag & drop usando las funciones “draggable(...)” y “droppable(...)”, sobre una imagen de un billete de avión y una de una cesta de la compra, respectivamente. De este modo, se puede arrastrar el billete hasta la cesta y que al soltarlo allí se llame al evento “drop”, vinculado a una función de Javascript que cambia la imagen de la cesta vacía por una imagen con un billete, contribuyendo a dar realismo a la ilusión de depositar el billete. En este punto, además, se incrementa el número de billetes reservados en el campo numérico adyacente, que se puede usar de forma independiente para configurar esta cifra. Para conseguir que si se suelta el billete en una posición fuera de su “target” retorne a su ubicación inicial se utiliza la opción “revert” (asociada a una función propia que restaura las coordenadas) como parámetro para “draggable(...)”, y para que al arrastrar la imagen del billete esta sea parcialmente translúcida se usa “opacity” con valor inferior a 1.

Las funcionalidades de comunicación cliente-servidor (específicamente la del autocompletado de los aeropuertos en la página de reservar vuelos) que antes se iniciaba con “Ajax.Request(...)” de Prototype ahora se lleva a cabo con JQuery. Se probó la función “\$.ajax(...)”, pero posteriormente se

usó “\$.post(...)”, que es similar pero tiene implícito el uso de POST como método de envío de datos. Se vincula la respuesta a una función de callback que analiza el “statusText” para determinar si se ha producido un error, en cuyo caso se muestra una alerta informativa; si todo ha ido bien (el string de status es “success”), se procesa la respuesta de las sugerencias de aeropuertos del mismo modo que en la entrega anterior.

Figura 1: Formulario de reserva de vuelos que detecta los campos no válidos y muestra un mensaje informando de ello, además de colorearlos en rojo usando JQuery y JQuery UI para sustituir a Prototype y Scriptaculous en las funcionalidades y efectos ya desarrollados anteriormente.

- Criteria to choose or discard an airplane seat:

  - Seats at the **tail end of the plane** often have no middle seats, which gives you more room to spread out.
  - Seats **just before the exit row** and **at the end of a section** may not recline.
  - Seats **next to the toilets** may be smelly and have lots of people trooping up and down to them, or queuing outside of them.
  - Seats **next to the galleys** may be noisy especially when flight attendants prepare and roll-out the meals, and surprisingly smelly from steam-heated food.
  - **Certain rows** may have the electronics for the seat-back entertainment under the seat in front, stealing leg room. Check the sites listed below to identify them.
  - The **first** seats may not have a place to put your bag, so you have to put it in another place.

Figura 2: Lista con elementos reordenables gracias al uso del efecto de la función “sortable()” de JQuery UI.

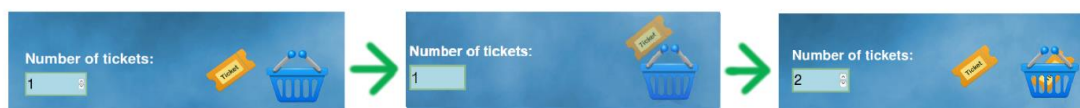


Figura 3: Implementación de drag & drop con JQuery UI para poder arrastrar billetes a una cesta de la compra y así incrementar el número de reservas (también se puede hacer desde el campo numérico).

## 2. Apartado 2

En este apartado se implementa una transferencia de datos, tanto de cliente a servidor como en sentido inverso, de servidor a cliente, usando JSON como estructura contenedora del mensaje. En

cuanto a la transferencia de datos de servidor a cliente, se utiliza para enviar al cliente datos procedentes de la base de datos Simpsons para mostrar una tabla de un ficticio “hall de la fama”, interpretando su identificador de la base de datos como su número de vuelos encargados con la página. Para comenzar, se realiza una consulta del servidor a la base de datos Simpsons en un nuevo archivo PHP, “get\_json\_stats.php”, obteniendo el nombre, email e identificador (a tratar como número de vuelos). Esto se almacena temporalmente como un array de arrays, siendo cada array interno un conjunto con los 3 datos correspondientes de un personaje de los Simpsons. Ello se codifica como string de JSON con “json\_encode(...)”. En el cliente, en la función callback de Ajax con Javascript, se convierte a objeto JSON el string recibido, usando “JSON.parse(...)”. Esto ocurre en “stats.js”, el archivo de Javascript para la página de muestra de estadísticas, que posteriormente construye una tabla dinámicamente con los campos antes mencionados, y añade un “caption” como título de la tabla, así como los títulos de las columnas, con “th”. La petición AJAX del cliente al servidor se solicita nada más cargarse la página, y no se añade hasta que el servidor devuelve el resultado.

En cuanto a la transferencia del cliente al servidor, se realiza algo mucho más sencillo: se pasa como objeto JSON el string a medio escribir del campo del nombre de un aeropuerto para realizar el proceso de autocompletar de la entrega anterior, que en esta se adaptó a JQuery como ya se mencionó en el apartado anterior. La información consiste únicamente en una clave con su valor (el string a analizar en el servidor), que el servidor procesa si es válido (no nulo).



Name	Contact email	Booked flights
Lisa	lisa@fox.com	888
Milhouse	milhouse@fox.com	456
Ralph	ralph@fox.com	404
Bart	bart@fox.com	123

Figura 4: Tabla creada en el cliente a partir de información recibida por una transferencia JSON con AJAX desde el servidor.

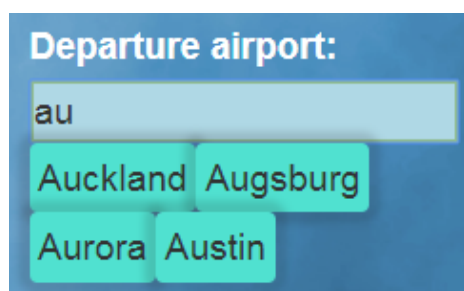


Figura 5: El nombre de un aeropuerto a medio escribir se envía como una estructura JSON sencilla de cliente a servidor para que sea procesado y se obtengan las sugerencias.

### 3. Apartado 3

En este apartado se implementa una transferencia de datos con XML del servidor al cliente con el fin de que el servidor pase al cliente un conjunto de datos que le permitan construir una tabla por Javascript (diferente a la del apartado de JSON, concretamente correspondiente a películas ordenadas por su puntuación en la base datos). Para comenzar, el servidor, en la página

“get\_xml\_stats.php”, crea un documento XML con “new DOMDocument(...)”. A este objeto se le da un elemento raíz “movies”, usando “createElement(...)”, que se añade con “appendChild(...)”. Posteriormente se iteran las filas obtenidas por una consulta a la base de datos de películas, y para cada una se crea un elemento XML “movie”, con tres atributos: nombre, año y puntuación. Se añade cada elemento “movie” al elemento raíz “movies”. Posteriormente se añade la cabecera XML con “saveXML(...)”, y se envía al cliente que la había solicitado por Ajax (en este caso con la función de JQuery “\$.ajax(...)”, especificando en el parámetro “dataType” que se quiere obtener un XML), que vuelve a ser, como en el apartado anterior, la página de estadísticas. En este caso, se usa la función “find(“movie”)” para iterar todas las películas con “each”. Para cada una se añade un elemento a una tabla, de forma similar a la especificada en el apartado anterior.

Name	Year	Rating
Shawshank Redemption, The	1994	9
Godfather, The	1972	9
Star Wars	1977	8.8
Memento	2000	8.7
Pulp Fiction	1994	8.7
Fight Club	1999	8.5
Matrix, The	1999	8.5
Kill Bill: Vol. 1	2003	8.4
Garden State	2004	8.3
Braveheart	1995	8.3

Figura 6: Tabla creada en el cliente a partir de información recibida por una transferencia XML con AJAX desde el servidor.

```
function buildTableFromXML(xml) {
    var parentDiv = $("#ajax-stats");
    parentDiv.append("<br><br>");
    var table = $("<table></table>");
    table.attr("id", "movie-table");
    table.addClass("custom-table table table-responsive table-bordered table-hover");
    parentDiv.append(table);

    var caption = $("<caption>Top rated movies</caption>");
    table.append(caption);

    var headRow = $("<tr><th>Name</th><th>Year</th><th>Rating</th></tr>");
    table.append(headRow);

    $(xml).find("movie").each(function(){
        var row = $("<tr></tr>");
        table.append(row);

        var name = $(this).attr("name");
        var rowElement = $("<td>" + name + "</td>");
        row.append(rowElement);

        var year = $(this).attr("year");
        rowElement = $("<td>" + year + "</td>");
        row.append(rowElement);

        var rank = $(this).attr("rank");
        rowElement = $("<td>" + rank + "</td>");
        row.append(rowElement);
    });
}
```

Figura 7: Código que permite crear una tabla en el cliente a partir de información recibida por una transferencia XML con AJAX desde el servidor.