2018

Computación orientada a la web: Documentación de los temas 4 y 5

ALBERT ESPÍN ROMÁN

13.4.2018 | PROFESOR: SIMONE BALOCCO

1. Apartado 1

En este apartado se ha implementado un sistema de validación de los datos introducidos por el usuario en el formulario de reserva de un vuelo en el lado del cliente, usando Javascript. Para ello, el primer paso ha sido crear un archivo ".js" donde se escribirían las funciones, y que sería enlazado por el documento HTML mediante tags "script", para conseguir una separación entre el contenido de la página (HTML), el estilo (CSS, ya ubicado anteriormente en archivos ".css") y el comportamiento que aporta Javascript, y que permite mejorar la usabilidad de la página dando feedback instantáneo al usuario como respuesta a sus acciones.

Se han seguido los principios de programación no obstructiva explicados en clase de teoría para evitar que un problema en Javascript pueda bloquear la carga de un documento HTML, y conseguir agilizar el proceso. Para ello, todas las funcionalidades básicas de asociación de elementos HMTL a eventos que se deben lanzar ante acciones del usuario sólo se ejecutan una vez que la página se ha cargado. Primeramente esto se hizo introduciendo el código mencionado dentro de la función handler del evento "window.onload" de Javascript, pero posteriormente se sustituyó por su equivalente de Prototype, "document.observe("dom:loaded")". Prototype también ha permitido reducir la longitud de las líneas de código gracias a sustituciones como "$('id')" en lugar de "document.getElementById('id')".

Un campo del formulario es validado cuando deja de tener el foco (evento "onblur"), y adicionalmente para las fechas también cuando cambia su valor (evento "onchange"), momento en el que se llama una función de validación que depende del campo y que incluye en muchos casos el uso de expresiones regulares (REGEX) análogas a las utilizadas en el servidor, pero también otras comprobaciones lógicas, como asegurarse que la fecha de nacimiento que ha introducido el usuario corresponde a una persona adulta. Si se concluye que el campo tiene un valor no válido, se hace visible una alerta con un texto diferente para cada campo. Además, se cambia la "class" del elemento HTML afectado (usando "removeClassName" y "addClassName") para que su visualización CSS varíe, en particular, 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 Scriptaculous. En particular, el campo afectado vibra gracias a la función "shake()" (pasando opciones personalizadas de "duration" y "distance" menores que las que tiene por defecto, para que sea suave, no exagerado); también parpadea, gracias a "pulsate()" (se reduce el número de pulsos por parámetros así como la duración, para conseguir nuevamente un efecto más sutil, no forzado). Adicionalmente, el mensaje de error aparece con "fade()", pero como por defecto esta función hace desaparecer elementos en lugar de causar su aparición, se requiere que el valor de "to" pasado como opción por parámetro sea mayor que el de "from". El aspecto de los campos erróneos vuelve a ser el normal invirtiendo el proceso descrito al lanzarse el evento "onfocus", o cuando se valida positivamente el campo.

Para experimentar con otras de las funcionalidades que Scriptaculous pone al alcance del programador, 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.create()". Volviendo a la página de las reservas, además, se ha implementado drag & drop usando las funciones "new Draggable(...)" y "Droppables.add(...)", 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 "onDrop", 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" de "Draggable", y para que al arrastrar la imagen del billete lo que se arrastre en realidad sea un duplicado de ésta y por tanto siga viéndose otro billete donde estaba el primero se usa la opción "ghosting": esto contribuye a la idea de que múltiples billetes pueden ser llevados a la cesta, uno tras otro.

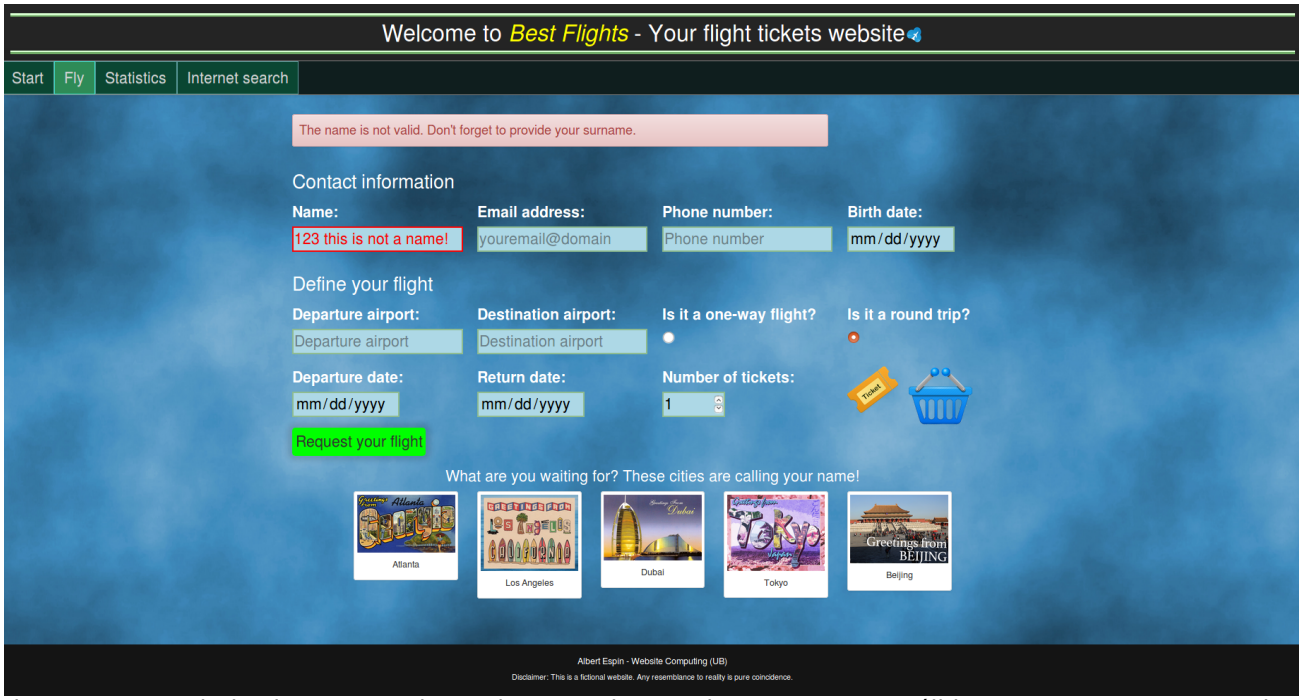

Figura 1: Formulario de reserva de vuelos que detecta los campos no válidos y muestra un mensaje informando de ello, además de colorearlos en rojo.

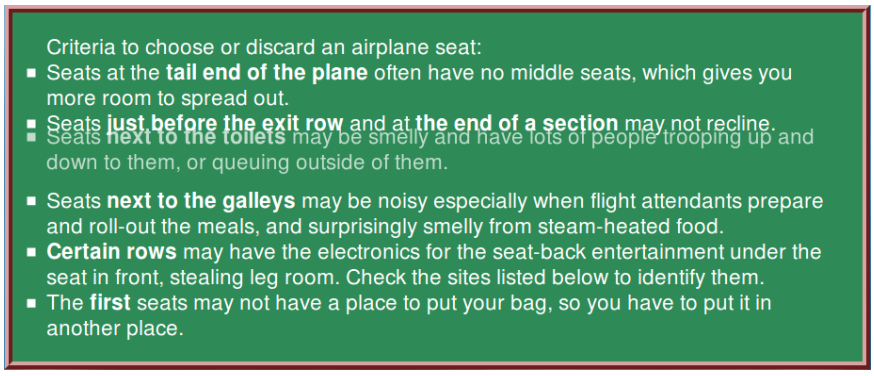

Figura 2: Lista con elementos reordenables gracias al uso del efecto Sortable de Scriptaculous.

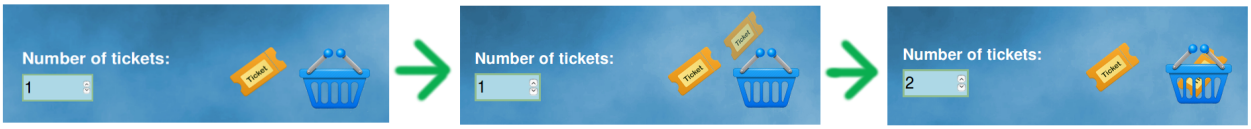

Figura 3: Implementación de drag & drop para poder arrastrar billetes a una cesta de la compra y así incrementar el número de reservas (también se puede hacer desde el campo numérico).

2. Apartado 2

En este apartado se ha desarrollado una funcionalidad de autocompletar para los campos de aeropuertos (de salida y destino), para que el usuario pueda seleccionar el nombre completo que tenía en mente con sólo escribir las primeras letras, ya que se le muestra una lista de sugerencias que puede seleccionar para trasladar a su campo de texto. Para lograr este efecto ha sido imprescidible el uso de la tecnología AJAX para comunicar el cliente con el servidor de forma asíncrona; el cliente pide una información al servidor (los aeropuertos que empiezan por el string introducido hasta el momento, al lanzarse el evento “onkeyup” del campo de texto), el servidor realiza una consulta a la base de datos y sobre ella usa lógica de strings para obtener la respuesta, sin paralizar el cliente, que, cuando la respuesta esta lista, la obtiene y procesa para actualizar la visualización de la página.

Se ha experimentado con las diferentes opciones propuestas en clase de teoría para crear la petición de AJAX. Inicialmente se usó la versión estándar de Javascript, creando un objeto XMLHTTP request, abriendo una conexión con el servidor con “open()” y enviándola con “send()”, vinculando el evento “onreadystatechange” del objeto a una función que procesara los datos recibidos para mostrar la vista actualizada, siempre y cuando hubiera habido éxito (“readyState” con valor exitoso 4 y “status” de HTTP con valor 300). Se usó inicialmente el método GET, adjuntando a la URL del servidor el parámetro string (el nombre del aeropuerto sin completar). Posteriormente, para mayor seguridad, se optó por usar POST, enviando un diccionario con una pareja nombre-valor para el string mencionado, a pasar como parámetro de la función “send()”.

Posteriormente, se optó por usar el “Ajax.Request()” de Prototype, que permite reducir el número de líneas necesarias para configurar la petición que se hace al servidor; se probó GET y se acabo sustituyendo por POST para conseguir mayor privacidad de los datos. Se vincularon los eventos “onError” y “onException” a una función que muestra información sobre un hipotético error en caso de que tenga lugar, para ofrecer un feedback al usuario. Se vinculó, asimismo, el evento “onSuccess” a la función que procesa la respuesta.

La función procesadora de la respuesta del servidor extrae los nombres de los aeropuertos del texto de respuesta “responseText” del objeto de request Ajax. Por cada uno de estos nombres, se crea dinámicamente un botón, usando “document.createElement()”, posteriormente añadido a un “div” asociado al campo de texto del aeropuerto, mediante la función “appendChild()”. Antes, sin embargo, se configura detalladamente el botón para que muestre el texto del nombre del aeropuerto de su iteración, y se lo vincula dinámicamente a un evento “onclick” para trasladar este texto al campo de escritura del usuario en caso de pulsar el botón, haciendo realidad la posibilidad de autocompletar el texto.

```php
if (isset($_POST["string"])) {

    $query = strtolower($_POST["string"]);
    $queryLength = strlen($query);

    if ($query !== "") {

        $allAirportNames = getAllAirportNames();

        $hint = "";

        foreach($allAirportNames as $name) {

            if (stristr($query, substr($name, 0, $queryLength))) {

                if ($hint == "") {

                    $hint = $name;
                }

                else {

                    $hint .= ", $name";
                }
            }
        }
    }

    echo $hint;
}
```

Figura 4: Bloque de código principal del servidor (“get_airport_hint.php”) para obtener las sugerencias de aeropuertos a partir del string enviado por el cliente mediante AJAX.

```php
$citiesDB = new PDO("mysql:host=$servername;dbname=world;charset=utf8", $username, $password);
$citiesDB->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
$query = "SELECT DISTINCT name FROM cities ORDER by name ASC";
$rows = $citiesDB->query($query);
foreach ($rows as $row) {
    $allAirportNames[] = $row["name"];
}
return $allAirportNames;
```

Figura 5: Extracto de la función “getAllAirportNames()” del servidor que obtiene los nombres de las ciudades para los aeropuertos.

```javascript
function showAjaxAirportHints(airportTextField, hintDiv) {

    return function(ajax) {

        var string = ajax.responseText;

        if (string == "") {

            $(hintDiv).innerHTML = "No suggestions";
        }

        else {

            var airportNames = string.split(", ");

            var buttonCount = 0;

            airportNames.forEach(function(name) {

                if (isValidAirportName(name)) {

                    var button = document.createElement("button");
                    button.id = $(hintDiv).id + "-button-" + buttonCount;
                    button.addClassName("hint-button");
                    button.innerHTML = name;

                    button.onclick = function() {

                        resetTextError($(airportTextField));

                        $(airportTextField).value = this.innerHTML;

                        clearHintsForAirportText($(airportTextField));

                        return false;
                    };

                    $(hintDiv).appendChild(button);

                    buttonCount++;
                }
            });
        }
    }
}
```

Figura 6: Función de Javascript que se llama en el cliente al recibirse los datos del servidor (evento onSuccess), donde se crean dinámicamente botones con las sugerencias de aeropuertos que el usuario puede pulsar para rellenar el campo de texto (mediante un evento onclick asignado dinámicamente).

Figura 7: Funcionalidad de autocompletar para los campos de texto de los aeropuertos.