Datatables

Control de versiones del documento

Versión	Fecha	Autor	Motivo del cambio
1.0	05/08/2015	Antonio Morcillo Martínez	Creación del documento

Datatables

Este documento tiene como finalidad describir los pasos necesarios para crear tablas HTML de datos paginables, ordenables y con capacidades de búsqueda de forma dinámica.
Para ello se empleará un *plugin* de *Jquery* conocido como *Datatables* (www.datatables.net) que se encargará de realizar la inmensa mayoría de operaciones.
Nos centraremos en el caso en el que los datos que se muestran en las tablas son obtenidos de un servidor mediante una llamada *AJAX.*
Este documento asume que nos encontramos dentro de los portales temáticos de la CARM bajo *Liferay 6.2* por lo que en algunos puntos se mencionará funcionalidad aportada por *Liferay*, por los temas o por el servidor *RESTFulCARM* todos ellos parte de este proyecto. No obstante el funcionamiento básico y principales conceptos son extrapolares a cualquier otro sistema web

Creación de un *datatable*

Para la creación de una tabla necesitamos de tres elementos esencialmente:

- *HTML*: El esqueleto estático básico de una tabla *HTML* a partir de la cual se construirá la tabla dinámica.
- *Javascript*: Código de inicialización de la tabla y definición de columnas. Adicionalmente puede existir código para operaciones adicionales
- Datos: El conjunto de datos que poblará la tabla de forma dinámica

Recursos

En el momento de escribir este documento, *Datatables* se encuentra en la versión 1.10.7 y los recursos necesarios para su funcionamiento pueden obtenerse de su *CDN* en las siguientes *URLs*.

Lo ideal es que estos recursos se guarden y sea nuestro servidor web quién los haga disponibles a los usuarios.

Javascript

```
http://cdn.datatables.net/1.10.7/js/jquery.dataTables.min.js
```

CSS

```
http://cdn.datatables.net/1.10.7/css/jquery.dataTables.min.css
```

En el caso de los portales temáticos de la CARM, el tema *CARM-theme* ya incluye una versión de *jQuery* por lo que no es necesario incluir desde los contenidos ninguna referencia extra a los recursos de *Datatables*

Origen de datos

Nos vamos a centrar en el caso en el que los datos son proporcionados por un servidor a través de una llamada *AJAX* por parte del navegador del usuario en tiempo de procesamiento de la página.
La forma de generar el conjunto de datos por parte del servidor es absolutamente indiferente siempre y cuando se respete el formato de respuesta pudiendo ser estos, entre otros:

- Consulta a base de datos
- Acceso a servicios web externos
- Acceso a ficheros estáticos

Para evitar problemas de *Cross-site scripting (XSS)* el navegador sólo nos permitirá hacer llamadas *AJAX* al mismo dominio del servidor que sirvió el *HTML* de la página

Formato de la respuesta

Para *Datatables* con fuente de datos accesible vía *AJAX* el formato de la respuesta será *JSON*; y la forma de servirlo será, en el caso más sencillo bien un *array* de *arrays* o bien un *array* de objetos

Array de arrays

```
[
    [
        "Francisco García",
        "21/10/1965"
    ],
    [
        "María Pérez",
        "13/4/1981"
    ],
    [
        "Andrés López",
        "04/6/1974"
    ]
]
```

Array de objetos

```
 [
    {
        "nombre": "Francisco García",
        "fecha_nacimiento": "21/10/1965"
    },
    {
        "nombre": "María Pérez",
        "fecha_nacimiento": "13/4/1981"
    },
    {
```

```
        "nombre": "Andrés López",
        "fecha_nacimiento": "04/6/1974"
    }
]
```

En el primer caso, como veremos posteriormente, para dibujar en una columna de la tercera fila la fecha de nacimiento de Andrés López tendremos que hacer

```
full[1]
```

Mientras que en el segundo caso, al tratarse de objetos *Javascript* bien formados podemos acceder directamente a través del nombre del campo mediante lo siguiente.

```
full.fecha_nacimiento
```

HTML

Lo primero que necesitaremos es tener definido una base *HTML* con la tabla y las cabeceras de las columnas que conforman la tabla. Colocaremos la tabla en el lugar que necesitemos en nuestro contenido web

```
<table id="tabla-datos">
     <thead>
          <tr>
               <th>Nombre</th>
               <th>Fecha de Nacimiento</th>
          </tr>
     </thead>
</table>
```

Hemos creado una tabla, definido su cabecera, que contendrá dos columnas *“Nombre”* y *“Fecha de Nacimiento”*

En negrita figura remarcado el id *HTML* de la tabla; este punto es clave puesto que es la forma óptima de referenciar a la tabla tanto para su creación como para cualquier otra operación que se pueda necesitar

Aunque se puede usar cualquier selector válido para *jQuery* lo ideal es usar identificadores

Javascript

Para minimizar el uso de variables globales, no recomendables bajo ningún concepto, haremos uso de las capacidades de *Javascript* para crear objetos con campos donde esos campos pueden ser variables o funciones y así el código y los datos que necesita quedarán encapsulados

Si existe más de una tabla dentro de la misma página bastará con llevar cuidado de no poner el mismo nombre a la variable que engloba todo el código en cada caso

Jquery

Como se ha comentado anteriormente *Datatables* es un *plugin* de *jQuery* por lo que es imprescindible que nuestra página disponga de *jQuery*

```
http://code.jquery.com/jquery-1.11.3.min.js
```

En el caso de los portales temáticos de la CARM, el tema *CARM-theme* ya incluye una versión de jQuery por lo que no es necesario incluir desde los contenidos ninguna referencia extra a *jQuery*

Creación de la tabla

Englobaremos todo el código de creación en un objeto javascript que se llame *DatatableControl.* En su forma básica tendrá esta forma

```
var DatatableControl = {
    returnedJson : null,
    table:null,
}
```

Hemos definido la variable *DatatableControl* con dos campos:

- ***returnedJson***: Guardará una copia del *JSON* devuelto por el servidor para su uso fuera del *datatable* (si fuera necesario)
- ***table***: Guardará una referencia a la tabla una vez creada

El siguiente paso es añadir al objeto *DatatablesControl* una función que inicialice la tabla, la llamaremos *initTable*

```
var DatatableControl = {
    returnedJson : null,
    table:null,
    initTable: function(){
      DatatableControl.table = jQuery('#tabla-datos').dataTable( {
                              "order": [[ 1, "asc" ]],
                              "ajax": {
                                  "url": "RUTA_AL_JSON",
                                  "cache":true,
                                  "dataSrc": function ( json ) {
                                      var dataTableJSON = new Object();
                                      dataTableJSON.data = json;
                                      DatatableControl.returnedJson =
dataTableJSON.data;
                                      return dataTableJSON.data;
                                      }
                                  },

                              language: DatatableCommon.spanishLabels,
                              "iDisplayLength": 50,
                              "columnDefs": [ {
                                "targets": 0,
```

```
                                "render": function ( data, type, full, meta
) {
                                  return full[0];
                                }
                              } ,
                              {
                                "targets": 1,
                                "render": function ( data, type, full, meta
) {
                                    return full[1];
                                }
                              }]})
                              }
                            }
```

Veamos paso a paso cada uno de los pasos de la función (En rojo figura la declaración de las columnas que se verá en el punto siguiente

- Creamos, y guardamos en *DatatableControl.table*, la tabla de datos
- Con *jQuery* seleccionamos el objeto *HTML* donde crear el *datatable* mediante la selección por id de la tabla que definimos en el punto anterior
- ***Order:****Array* de *arrays* que indica los criterios de ordenación por defecto de la tabla, en este caso ordenamos por la columna 1 de forma ascendente. Se podrían añadir tantas declaraciones de ordenación como fuera necesario
- ***Ajax*:** Objeto que le indica a *datatables* que la fuente de datos es una fuente remota. Cuenta con los siguientes campos:
  - ***url*:** La ruta del servidor de donde se debe obtener el *JSON*
  - ***cache*:** Un valor a true de este campo indica que la respuesta es susceptible de ser cacheada por lo que *jQuery* no le pondrá a la url un *timestamp* que impida el cacheo
  - ***dataSrc*:** función *callback* a ejecutar cuando la respuesta del servidor es satisfactoria. En este caso se guarda el *JSON* en la variable *DatatableControl.returnedJSON* y además se devuelve el *JSON*, sin modificar, para su uso por parte de datatables (***Si fuera necesario hacer un procesamiento previo del JSON antes de su* envío *a la tabla, este es el lugar para hacerlo)***
- ***language*:** Se le asigna la variable *DatatableCommon.spanishLabels* que es un objeto definido en el *CARM Theme* con las traducciones al castellano de las etiquetas de *datatables*
- ***iDisplayLength*:** Número inicial de registros por página

El código Javascript ha de ubicarse por debajo del HTML de la tabla o bien dentro de un *listener documentReady* de *jQuery*

Esta configuración asume que todos los datos son traídos desde el servidor de una sola vez, con una única llamada haciéndose la paginación, ordenación y búsquedas en el navegador del usuario

Declaración de columnas

Un aspecto clave de *datatable* es la configuración de columnas de la tabla. Esta configuración se pasa en la creación de la tabla en el campo *columnDefs* y consiste en un *array* de objetos *javascript* donde cada objeto del array define como se debe pintar una columna en cada una de las filas de la tabla.

En el caso más básico cada uno de los objetos está compuesto por dos campos *targets y render:*

- ***targets*:** Indica sobre qué columna/s se aplica la configuración que se está definiendo en ese objeto
- ***render*:** Función que indica como se debe pintar la celda. Recibe cuatro parámetros donde el más importante es el parámetro *full* que contiene la fila completa del array *JSON* correspondiente a la fila del *datatable.* Debe devolver un texto *HTML* válido pudiendo utilizar cualquier etiqueta HTML válida dentro de una tabla

Así que, por ejemplo, para pintar en la segunda columna de la tabla la fecha de nacimiento en negrita haríamos lo siguiente

```
{
      "targets": 1,
      "render": function ( data, type, full, meta ) {
            return '<strong>'+full[1]+'</strong>';
      }
}
```

No es necesario que el *JSON* devuelto por el servidor tenga las mismas columnas y en el mismo orden que la tabla *HTML* pero si que es obligatorio que el número de columnas definido en el *HTML* y el de *columnDefs* coincidan, de lo contrario *Datatables* emitirá un error de ejecución

Como ejemplo completo de creación de *datatable* tenemos el siguiente código *Javascript*

```
<script type="text/javascript">
var DatatableControl = {
    returnedJson : null,
    table:null,
    initTable: function(){
      DatatableControl.table = jQuery('#tabla-datos').dataTable( {
                              "order": [[ 1, "asc" ]],
                              "ajax": {
                                  "url": "URL_AL_JSON",
                                  "cache":true,
                                  "dataSrc": function ( json ) {
                                      var dataTableJSON = new Object();
                                      dataTableJSON.data = json;
                                      DatatableControl.returnedJson =
dataTableJSON.data;
                                      return dataTableJSON.data;
                                      }
                                  },
```

```
                            language: DatatableCommon.spanishLabels,
                            "iDisplayLength": 50,
                            "columnDefs": [ {
                              "targets": 0,
                              "render": function ( data, type, full, meta
) {
                                return full[0];
                              }
                            } ,
                            {
                              "targets": 1,
                              "render": function ( data, type, full, meta
) {
                                  return full[1];
                              }
                            }]})
                            }
                          }
DatatableCommon.initTable(); //Creación de la tabla
</script>
```

Tipo de datos

Por defecto *Datatables* considera que los datos son siempre cadenas de texto. Si bien esto es válido en la mayoría de situaciones puede ser que sea necesario especificar explícitamente el tipo de datos. Esto se hace mediante el campo *"type"* en la declaración de la columna.

Type puede tomar los valores:

- "*date-eu*": para fechas en formato dd/MM/yyyy
- "*currency*": para importes **(es necesario configuración adicional para especificar la moneda y los caracteres separadores de miles y decimales)**

```
{
      "targets": 6,
      type: 'date-eu',
      "render": function ( data, type, full, meta ) {
        return full[6];
      }
    }
```

Ordenación

Es posible desactivar la ordenación para una columna añadiendo el campo *orderable* y asignarle el valor *false*

```
{
      "targets": 6,
      type: 'date-eu',
      orderable: false,
      "render": function ( data, type, full, meta ) {
        return full[6];
      }
    }
```

Visibilidad

En ocasiones puede ser interesante ocultar una columna pero que sigue existiendo a efectos de filtrado y búsqueda; para ello *visible* y asignarle el valor *false*

```
{
      "targets": 6,
      type: 'date-eu',
      visible: false,
      "render": function ( data, type, full, meta ) {
        return full[6];
      }
    }
```

Búsqueda

Es posible desactivar la búsqueda para una columna añadiendo el campo *searchable* y asignarle el valor *false*

```
{
      "targets": 6,
      type: 'date-eu',
      searchable: false,
      "render": function ( data, type, full, meta ) {
        return full[6];
      }
    }
```

Datatables server-side

Con la configuración de *datatables* que hemos utilizado hasta ahora, sólo hay una única llamada *AJAX* en la creación de la tabla que se debe traer al navegador todo el conjunto completo de datos para la tabla. La paginación de resultados así como la ordenación y la búsqueda las hará el navegador del usuario a partir del conjunto completo traído desde el servidor.

Si el conjunto de datos es muy grande este enfoque podría no ser el mejor ya que tendríamos un tiempo de espera muy grande al inicio de carga de la tabla mientras se trae el conjunto completo de datos y un tiempo de espera muy grande también en cada operación de búsqueda, ordenación, paginación ya que el navegador del usuario tendría que lidiar con un conjunto muy grande de datos

Sin embargo existe la posibilidad de indicarle a *datatables* que todas estas operaciones han de realizarse en el servidor y solo recibir un subconjunto de datos ya paginado, filtrado y ordenado en el servidor.
Datatables enviará al servidor en cada operación del usuario, paginación, ordenación o búsqueda una solicitud *AJAX* donde le suministrará un conjunto de parámetros indicando:

- Número de registros por página
- Número de página actual

- Índices de las columnas por las que se está ordenando
- Criterio de ordenación de cada una de estas columnas
- Índices de las columnas por las que se está filtrando
- Criterio de búsqueda

Es responsabilidad del servidor recoger ese parámetro realizar las operaciones necesarios y responder a *datatables* con un *JSON* adecuado. Este *JSON* debe contener

- El número de registros totales
- El número de registros tras aplicar los filtros
- El conjunto de datos de acuerdo a los filtros, criterios de ordenación y paginación suministrados por *datatables*

Existe una extensión de *RESTFulCARM* para los portales temáticos capaz de procesar este tipo de solicitudes y devolver el *JSON* adecuado, para más información ver el manual de *RESTFulCARM*

Para configurar un *datatable server-side* es necesario simplemente añadir un parámetro de configuración en la creación de la tabla, indicado en negrita en el ejemplo siguiente

```
DatatableControl.table = jQuery('#tabla-datos').dataTable( {
                            "serverSide": true,
                            "order": [[ 1, "asc" ]],
                            "ajax": {
                                "url": "/rest-
services/services/restAPI/DatosIntervencion/",
                                "cache":true

                                },

                          language: DatatableCommon.spanishLabels,
                          "iDisplayLength": 50…
```

Para el caso de la extensión de *RestFulCARM* es necesario además que en cada columna definida en el campo *columnDefs* se le especifique el nombre de la columna que deberá ser igual al que el campo tiene en base de datos, esto permitirá a la extensión crear una sentencia SQL válida.

```
{
      "targets": 5,
      "name":"sociedad_ambito",
      "render": function ( data, type, full, meta ) {
        return  full[3];
      }
    },
```

Funcionalidades adicionales

A continuación se describen algunos comportamientos habituales detectados en los distintos portales web de la CARM que están haciendo uso de *datatables*. Alguna de esta funcionalidad se encuentra parametrizada mediante funciones Javascript disponibles en el tema *Liferay CARM Theme*

El tema cuenta con un fichero llamado *jquery.dataTables.plugins.js* donde se encuentra definida esta funcionalidad agregada.

Funciones de soporte del tema Liferay CARM Theme

Dentro de este fichero se encuentran definidos los tipos de datos '*date-eu*' y '*currency*' descritos en la definición de las columnas. Además cuenta con un objeto *DatatableCommon* que ofrece funcionalidad común a todos los *datatables* que deseen utilizarla:

- **spanishLabels:** Etiquetas en castellano para datatables
- **currencyConfig:** Configuración de moneda:
  - Moneda Euro
  - Separador decimal ','
  - Separador de miles '.'
  - Número de decimales 2

Aparte de esta funcionalidad básicas, el tema también dispone de funciones para facilitar la exportación de datos y el filtrado

Filtrado de datos

Datatables cuenta con una caja de búsqueda de texto libre que filtra en todos los campos, que no estén marcados como no válidos para búsqueda, mediante una consulta *like.*

Adicionalmente, puede ser interesante tener selectores (*combo-boxes*) que apliquen un filtro exacto a alguna de las columnas de la tabla.
El tema dispone de funciones para realizar esto teniendo el creador de contenido que únicamente definir el combo y ejecutar una función javascript.

El selector se poblará automáticamente con los valores de una columna del *JSON* y ejecutará la consulta sobre una columna específica de *datatables*

HTML

El primer paso que se debe dar para incorporar estos selectores es crear un elemento *HTML select* con un único elemento vacío siguiendo las normas de *HTML* de la guía de estilo.

```
<div>
      <label>Consejerias:  
            <select id="consejeriasselector">
                  <option value=""></option>
            </select>
      </label>
</div>
```

JAVASCRIPT

Por otro lado debemos invocar a la función *javascript* del tema que inicializa el combo. Deberemos invocarla en un punto del código donde se esté seguro de que el servidor ha devuelto el *javascript*, por lo que un buen punto para invocar la función es la función *callback dataSrc* de *datatables* que se ejecuta a la recepción correcta del *JSON* desde el servidor.

```
"dataSrc": function ( json ) {
           var dataTableJSON = new Object();
           dataTableJSON.data = json;
           DatatableControl.returnedJson = dataTableJSON.data;

      DatatableCommon.initSelector(jsonOrigen,"actuacionselector",0,0);
           return dataTableJSON.data;
      }
```

Los parámetros de la función son, por orden

- *JSON* origen de los datos
- id *HTML* del elemento *select HTML*
- indice de la columna en el *JSON* origen del que obtener los valores para poblar el *select*
- indice de la columna en la tabla donde se debe filtrar

La función:

- recorrerá el *JSON*,
- obtendrá un listado con todos los valores no duplicados de la columna indicada,
- los ordenará alfabéticamente
- añadirá una función al *select* para que, cada vez que cambie de valor, se ejecute una consulta en *datatables,* únicamente en la columna indicada en la invocación de la función

A continuación se muestra el código de la función; a partir del mismo no es costoso crear variaciones del mismo con comportamientos específicos para cada caso o con funcionalidad más compleja.

```
initSelector:function(jsonOrigen,idDOM,jsonIndex,columnIndex){
            var data = [];
            jQuery(jsonOrigen).each(function(index, element) {
                var sanitizedValue =
jQuery.trim(element[jsonIndex]).split("  ").join(" ");
                if(jQuery.inArray( sanitizedValue, data ) < 0){
                    data.push(sanitizedValue);
                }
            });
            data.sort(function(a, b){
                if(a < b) return -1;
                if(a > b) return 1;
                return 0;
            });
            jQuery.each( data, function( key, value ) {
                var o = new Option(value, value);
                jQuery(o).html(value);
                jQuery("#"+idDOM).append(o);
```

```
            });

            jQuery("#"+idDOM).on('change', function (e) {
                var optionSelected = jQuery("option:selected", this);
                var valueSelected = this.value;

DatatableControl.table.DataTable().columns(columnIndex).search(valueSelec
ted).draw();
            });
        },
```

Exportación de datos

El tema también cuenta con funcionalidad para, apoyándose en el servidor *Liferay*, generar exportaciones de la tabla de datos en formatos *PDF, XLS y CSV*

Existen dos modos de exportación:

- **Exportación de toda la tabla** (Hará que el servidor pida el *JSON* completo y genere el documento al vuelo con la información del configuración que se le pasa al servicio)
- **Exportación de la página actual** (Se envía al servidor el *HTML* de la tabla y con ese *HTML* se genera el *PDF*, el *XLS* o el *CSV*)

Simplemente hay que añadir contenedores en la zona del *HTML* donde se desee tener los botones

```
<div class="exportOptions" id="exportOptions"> </div>
```

Estos contenedores pueden repetirse ya que el tema inyectará el código a partir de selectores *jQuery* por lo que se puede especificar por ejemplo selectores por clase *CSS* y tener botones de exportación arriba y abajo de la tabla

La función para exportar toda la tabla completa es la siguiente

```
DatatableCommon.initDatatablesSingleExport('.exportOptions','#tabla-
datos',configuration);
```

donde

- **#exportOptions:** es un selector *jquery* para seleccionar donde quieres que se inserte el html de exportación)
- **#tabla-datos**: es un selector *jquery* para seleccionar la tabla. Lo normal es que sea por id
- **configuration** es un objeto javascript de configuración del servicio que se pasará al servidor como *JSON* para que el servidor sepa qué es lo que tiene que hacer.

A continuación se muestra un ejemplo completo de configuración usado en el portal de transparencia:

```
var configuration = new Object();
     var columna1 = new Object();
     columna1.nombre = "Código";
     columna1.indicesJSON = [0,1,2];
     columna1.separadores = [" ","/"];
     columna1.tipo="cadena";

     var columna2 = new Object();
     columna2.nombre = "Tipo Contrato";
     columna2.indicesJSON = [];
     columna2.indicesJSON.push(1);
     columna2.tipo="cadena";

     var columna3 = new Object();
     columna3.nombre = "Objeto";
     columna3.indicesJSON = [];
     columna3.indicesJSON.push(3);
     columna3.tipo="cadena";

     var columna4 = new Object();
     columna4.nombre = "Importe adjudicación";
     columna4.indicesJSON = [];
     columna4.indicesJSON.push(5);
     columna4.tipo="cadena";

     var columna5 = new Object();
     columna5.nombre = "Adjudicatario";
     columna5.indicesJSON = [];
     columna5.indicesJSON.push(6);
     columna5.tipo="cadena";

     var columna6 = new Object();
     columna6.nombre = "Fecha formalización";
     columna6.indicesJSON = [];
     columna6.indicesJSON.push(8);
     columna6.tipo="cadena";


     var columnasArray = [];
     columnasArray.push(columna1);
     columnasArray.push(columna2);
     columnasArray.push(columna3);
     columnasArray.push(columna4);
     columnasArray.push(columna5);
     columnasArray.push(columna6);
     configuration.url = "/rest-services/services/contratos/getData";
     configuration.nombreFichero = "contratos";
     configuration.columnas = columnasArray;
     configuration.columnasVisibles = [0,1,2,3,4,5];
```

El objeto configuración necesita de los siguientes campos:

- ***url*:** *URL* de donde el servidor ha de sacar los datos para montar el listado completo (solo soporta *array* de *arrays* que es lo que devuelve *RESTFulServices*)
- **columnas:** *array* de objetos con la definición de columnas para el listado completo
- **nombreFichero (opcional):** el nombre sin extensión del fichero a generar
- **columnasVisibles(opcional):** array de enteros que indica, para el listado de página actual, que columnas del *datatable* se mantienen en el listado, si se omite, se exporta todo
- **Objeto columna:**
  - nombre: nombre de la columna
  - indicesJSON: array de enteros que indica las posiciones en el array *JSON* de donde leer los datos
  - tipo: el tipo de datos final para cada uno de los elementos de la columna. Ahora mismo se soporta:
    - cadena
    - imagen
    - fecha
  - separadores(opcional): si en el array indicesJSON se ha especificado más de un valor, se puede especificar un array de cadenas que se intercalaran entre cada uno de los valores obtenidos. Por ejemplo en la definición de la columna 1 tenemos columna1.indicesJSON = [0,1,2]; y columna1.separadores = [" ","/"]; si procesamos un JSON que tiene algo como ["CONVENIO","2014","4","OTRO CAMPO",...]... con esta configuración el sistema generaría CONVENIO 2014/4 para la primera columna en la primera fila de la tabla

Para generar botones de exportación que permitan seleccionar al usuario si desea la tabla completa o la página de visualización actual basta con llamar a la función

```
DatatableCommon.initDatatablesExport('.exportOptions','#tabla-
datos',configuration);
```

Los parámetros y su significado son exactamente los mismo que los del caso anterior.

Esta funcionalidad está proporcionada por el componente ***Rest-Services*** desplegado en los servidores Liferay y que entre otras responsabilidades se encarga de

- Hacer de pasarela entre las peticiones *AJAX* de los *datatables* y el servidor *RESTFulCARM*
- Cachear las respuestas de estos servicios
- Ofrecer un servicio web de limpieza de caché
- Hacer de pasarela entre las peticiones de binarios del nuevo portal del CREM y del antiguo
- Ofrecer un servicio web que lee el organigrama de una unidad organizativa de la CARM de la web de IGES
- Ofrecer un servicio web para registrar en base de datos el acceso a contenido web de liferay
- Ofrecer un servicio web para la exportación de sitios web de Liferay sin necesidad de usar la interfaz visual
- Exportación de datatables

Ventanas modales

Por último, en casos por donde, bien por limitaciones de espacio, bien por necesidades de procesamiento o simplemente por no presentar al usuario una tabla muy compleja, no se muestran en cada fila todos los datos, pudiera ser interesante que alguna de las columnas presentara un enlace que al pulsar sobre el mostrara el resto de datos en una ventana modal.

Para realizar los modales nos apoyaremos en *Bootstrap* (www.getbootstrap.com) ya que el *framework* de referencia que usan los portales temáticos de la CARM

Definición de la columna

En este caso lo principal es ligar el *JSON* con la tabla para tener en todo momento una referencia a la fila donde el usuario ha hecho *click*.
Esto se consigue, en el renderizado de la columna que contendrá el enlace, mediante el campo *row* de la variable *meta* que suministra *datatables.* Una buena manera de conservar el valor es hacer uso de los atributos *data* de HTML 5 para después recuperarlos con *jQuery* (en negrita en el ejemplo).

```
{
      "targets": 3,
      "render": function ( data, type, full, meta ) {

        if(full[18] == 0){
          return '';
        } else if (full[18] == 1){
          return '<a href="#" data-row="'+meta.row+'" data-
codigo="'+full[0]+'" class="verDetalle">' + full[18] + '
referencia'+'</a>';
        }else{
          return '<a href="#" data-row="'+meta.row+'" data-
codigo="'+full[0]+'" class="verDetalle">' + full[18] + '
referencias'+'</a>';
        }

      }
    },
```

Es importante notar que los enlaces llevan una clase que no agrega ningún estilo '*verDetalle*' pero que se usará para ligarle el evento click a todos los enlaces a la vez vía *jQuery*

Evento de click

Para añadir el comportamiento que queremos debemos añadir un *listener* al evento de dibujado de *datatable* ya que estamos modificando el comportamiento habitual de un enlace y por tanto cada vez que se refresque la tabla por cualquier motivo deberemos añadir este comportamiento.

Para ello, en su forma más básica añadiremos este código inmediatamente después de la creación de la tabla

```
jQuery('#tabla-datos').on( 'draw.dt', function () {
```

```
      jQuery( ".verDetalle" ).click(function(event) {
            event.preventDefault();
      });
});
```

Este código simplemente añade, cada vez que *datatables* se recarga (paginación, búsqueda, ordenación) un evento de click a los enlaces *verDetalle* que anula el comportamiento del enlace (en negrita)

Recuperación de datos de la fila seleccionada

El siguiente paso es recuperar los datos de la fila clicada por el usuario para ello haremos uso de la copia del *JSON* que tenemos almacenada en el objeto de control así como del atributo *data* que hemos puesto en el enlace. Quedando el código del *click* de la siguiente forma

```
jQuery('#tabla-datos').on( 'draw.dt', function () {
      jQuery( ".verDetalle" ).click(function(event) {
            var row = jQuery(this).data("row");
            var val = DatatableControl.returnedJson[row];
            event.preventDefault();
      });
});
```

En la primera línea en negrita recuperamos el índice de la fila clicada y en la segunda línea guardamos la fila completa proveniente del *JSON* original

Underscore

Antes de mostrar el modal deberemos cargarlo con los datos provenientes de la selección del usuario puesto que cambiará de una fila a otra.
Para ello, si el modal presenta una complejidad razonable, podemos hacer uso de sistema de plantillas de *Underscore*.
*Underscore (*http://underscorejs.org/*)* es una librería de utilidades *Javascript*, que entre otras cosas permite utilizar en el navegador plantillas similares a *Velocity* con variables, bucles y sentencias de control. El tema *CARM Theme* incluye esta librería

A continuación se muestra como ejemplo un modal *Bootstrap* con una variable *Underscore*

```
<div aria-hidden="true" aria-labelledby="myModalLabel" class="modal"
id="panelmodal" role="dialog" tabindex="-1">
      <div class="modal-content">
            <div class="modal-header">
                  <button aria-label="Close" class="close" data-
dismiss="modal" type="button"><span aria-hidden="true">×</span></button>
                  <h4 class="modal-title"
id="myModalLabel">modalTitulo</h4>
            </div>
            <div class="modal-body">
                  <div id="mainContainer">
                        <div id="template-wrapper">
```

```
                            <h1 class="titulo-
seccion">&lt;%=variable%&gt;</h1>
                        </div>
                    </div>
            </div>
            <div class="modal-footer"><button class="btn btn-default"
data-dismiss="modal" type="button">Cerrar</button></div>
      </div>
</div>
```

En negrita figuran los aspectos importantes para *Underscore* siendo el resto el HTML estándar que espera *Boostrap* en un modal

Necesitamos un contenedor como punto de partida de la plantilla, en este caso está marcado con el id "*template-wrapper"*. Le indicaremos a *Underscore* que la plantilla empieza ahí

Además del contenedor de la plantilla necesitamos otro contenedor adicional puesto que en cada click necesitamos borrar lo que había y volverlo a crear, esto es responsabilidad del contenedor con id *mainContainer*

Dentro de la plantilla tenemos un encabezado de primer nivel *h1* que *Underscore* lo va a rellenar con el contenido de la variable *variable* para eso se emplea la misma notación que *JSP*

```
<%=variable%>
```

El editor *HTML* de Liferay no permite caracteres no *HTML* por lo que convertirá automáticamente esta expresión a **<%=variable%>.** El código *Javascript* debe tener en cuenta esto y deshacer este cambio

Javascript

Es necesario modificar el código de nuestra plantilla en distintos puntos para dar soporte a las plantillas y al modal. En negrita irán apareciendo lo que se debe ir añadiendo en cada lugar

Al igual que con el *JSON* y el *datatable* nuestro objeto de control va a guardar una referencia a la plantilla por lo que debemos tener una variable que la almacene

```
  var DatatableControl = {
    returnedJson : null,
    template : null,
    table:null,
…
```

En la función de creación de la tabla *initTable* ocultamos el modal y guardamos la referencia a la plantilla antes incluso de cargar la tabla

```
initTable: function(){
      jQuery('#panelmodal').hide();
      DatatableControl.template = jQuery('#template-wrapper').html();
      DatatableControl.table = jQuery('#legislacion').dataTable( {
…
```

Finalmente el código del evento de *click* del enlace queda como sigue.

```
jQuery('#tabla-datos').on( 'draw.dt', function () {
      jQuery( ".verDetalle" ).click(function(event) {
            var row = jQuery(this).data("row");
            var val = DatatableControl.returnedJson[row];
            var sanitizedTemplate =
DatatableControl.template.split("&lt;").join("<").split("&gt;").join(">")
;
            var tpl = _.template(sanitizedTemplate);
            jQuery('#mainContainer').empty();
            jQuery('#mainContainer').append(
                  tpl({variable: val[0],
                  })
            );
            event.preventDefault();
      });
});
```

Hemos añadido código que hace lo siguiente:

- Convierte los caracteres escapados por Liferay en caracteres válidos para *Underscore* en la plantilla original guardada en el objeto de control
- Crea el objeto plantilla de *Undercore*
- Vacía el contenedor principal
- Lo vuelve a crear con el contenido de la plantilla creando la variable *variable* con el valor de la primera posición del *JSON* al que apunta la fila ciliada por el usuario

Modal

Por último sólo queda mostrar el modal mediante *Javascript*

```
jQuery('#panelmodal').modal({show:true});
```