Conociendo ya una primera visualización de nuestra aplicación, y viendo que la creación de las gráficas funcionaba correctamente se procedió a la ampliación del número de funcionalidades que aportaba el entorno.

De esta manera se mejoró ligeramente la visualización del gridster y los widgets, para intentar pasar al siguiente ciclo, la implementación de un servidor, guardado y aplanamiento de las gráficas y la reproducción de las mismas si se quiere acceder de nuevo a ese entorno de trabajo. Este pequeño paso intermedio era importante para hacerle un uso semejante al de traza con el que discernir durante su programación si el funcionamiento era correcto o no.

Para lograr esto se procedió al uso de django y python. Un servidor a través del cual se añadirían las urls y del cual se haría uso de sus bases de datos en primera instancia para poder hacer las pruebas pertinentes. Véase que si en un futuro se quisiera subir la aplicación a un entorno web permanente existen herramientas que nos pueden hacer la vida más fácil pero la mayoría de ellas no hacen uso de las bases de datos sqlite de las que hace uso django por defecto.

Es de este modo que se presenta la siguiente organización de urls que nos permitirán interactuar con nuestra aplicación:

A vista del usuario:

../ : Dashboard vacío, página home desde el que empezar a utilizar nuestra aplicación

../\d+: Dashboard ya creado que se carga para seguir modificándolo a nuestro gusto

Desde el punto de vista del servidor:

../:

Get:

Dashboard vacío, página home desde el que empezar a utilizar nuestra

/db/:

Get:

Listado de ids de los dashboards creados

Post:

Creación de un nuevo dashboard.

/db/\d+:

Get:

Carga de un dashboard ya creado, devuelve el json de configuración del mismo.

Put:

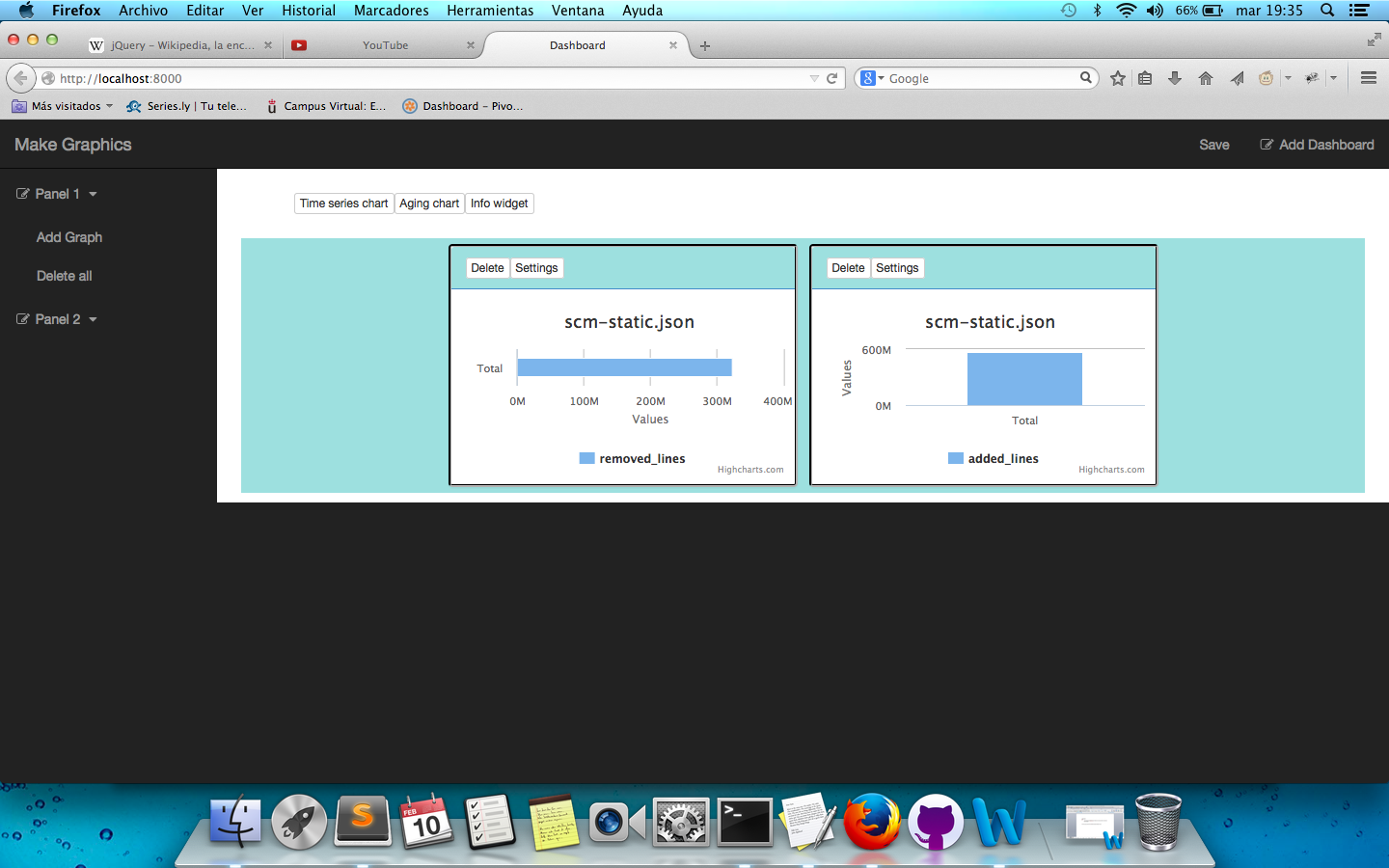
Sobreescritura de un dashboard ya creado con una nueva configuración.

/db/default:

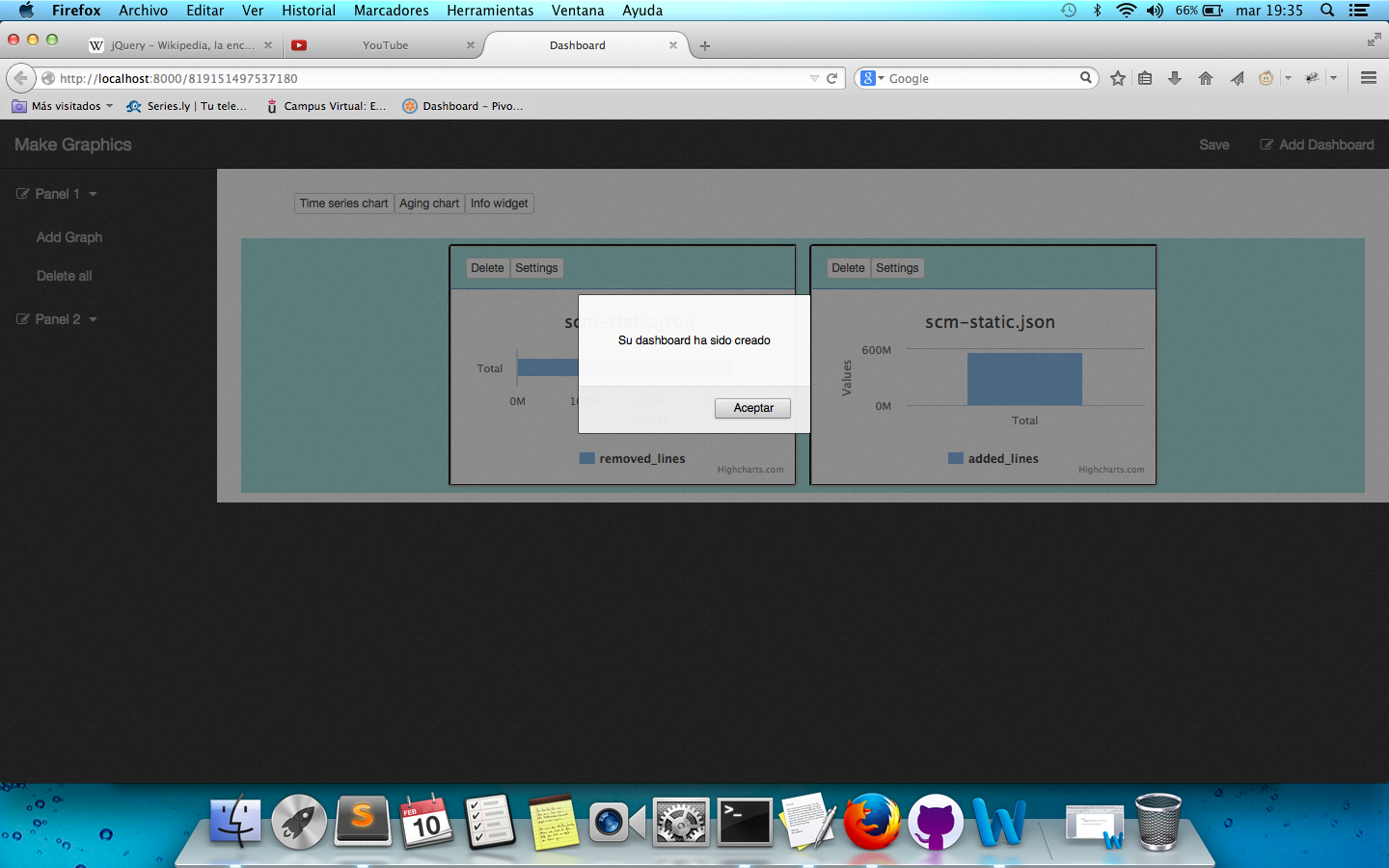
Get:

Presentación del dashboard por defecto con el que mostrar al usuario la funcionalidad.

Con esta nueva estructura tendríamos algo semejante a esto:



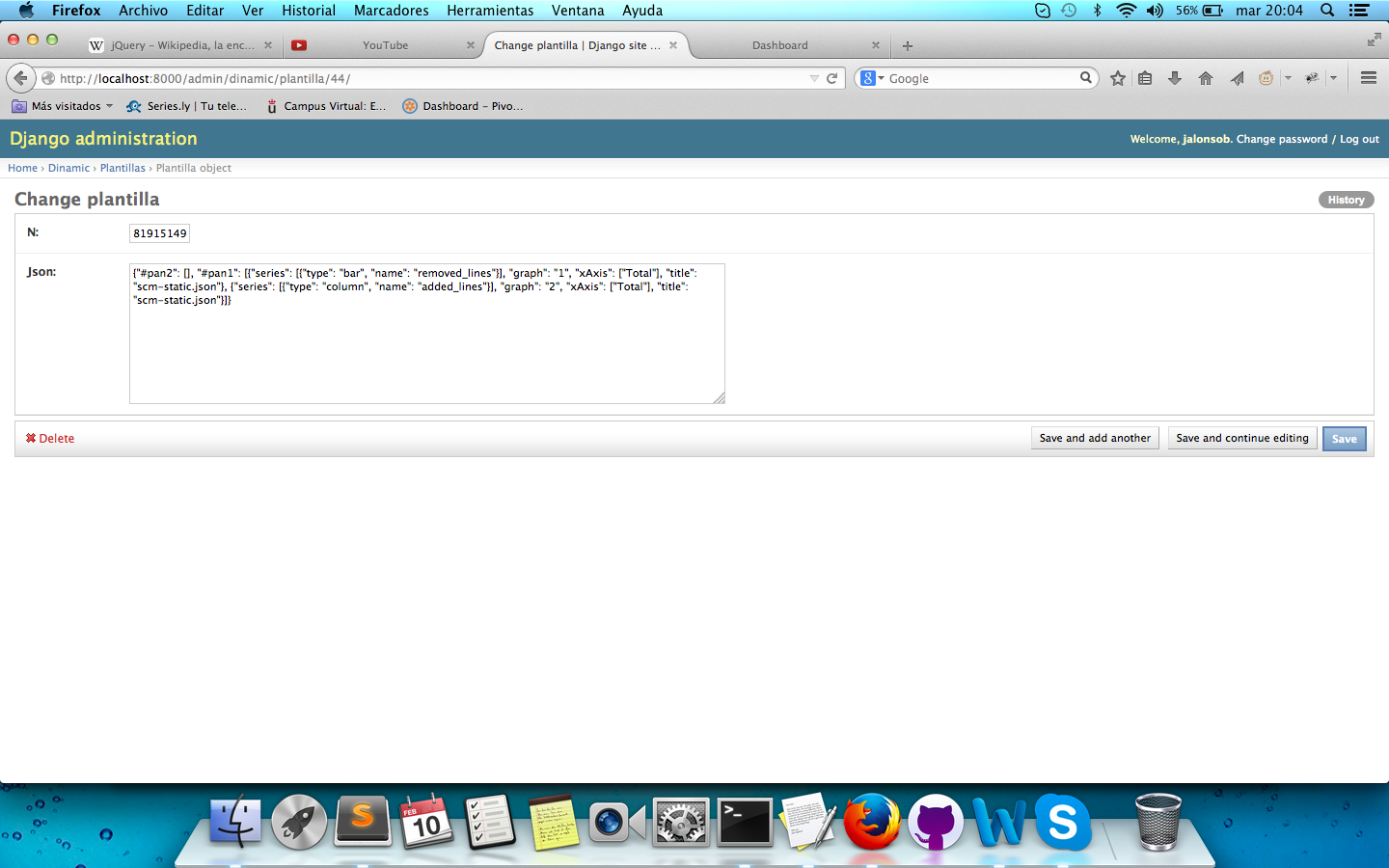
Como observamos en la figura anterior hemos creado dos gráficas básicas sobre un dashboard vacío aún no creado. Las aplicación permite el guardado de las mismas mediante el botón “Save” situado arriba a la derecha. Al pulsarlo…



Se creará un nuevo dashboard en nuestro servidor, un mensaje aparecerá en pantalla para avisarnos de tal evento.

Como podemos ver en la segunda imagen la url en la que estamos posicionados ha cambiado conforme hemos guardado, situándonos ya en el lugar correspondiente de nuestro entorno de trabajo.

Si entramos en el funcionamiento del fichero de configuración tendríamos actualmente algo de este formato.



Cada dashboard está identificado por un número único N y por un json de configuración. Dicho json es un objeto cuyas claves son los paneles que dibujamos, y cuyo contenido son arrays. Dentro de cada posición de los arrays tenemos objetos con las características necesarias para guardar debidamente una gráfica. El título, las métricas y su formato así como el identificador de gráfica son elementos importantes a la hora de guardarlo dentro del servidor para posteriormente reproducirlo debidamente.

Uno de los mayores problemas que existen al hacer uso de una aplicación que necesita de peticiones de datos de gran envergadura es el tiempo de carga con respecto al tiempo de espera de los datos.

Por eso fue necesaria una reestructuración completa del código orientada a la espera de los datos para después hacer uso de los mismos. Es decir, pido los datos si son necesarios, si están pedidos espero a recibirlos, si no los recibo lanzo un evento orientado a un error en la descarga, y si ha resultado correcta la petición pinto la gráfica sobre la que estaba trabajando y las subsiguientes que tal vez dependían del mismo tipo de json.

Establecido nuestro servidor, nuestro guardado y reproducción de charts, pasamos al siguiente ciclo donde puliremos algunos defectos que actualmente posee nuestra aplicación.

Implementación:

Debido a que en esta parte tocamos el código más en profundidad es necesario abrir una segunda sección desde la cual explicar cómo está estructurado el código.

Todo nuestro código se basa en 3 partes distintas correspondientes a cada uno de los widgets. Cada parte a su vez se subdivide en una serie de funciones de funcionamiento semejante pero especializada en cada widget, lo que nos permite ser más claros en el código y traducirlo a un formato de objetos si fuera necesario en un futuro. Para verlo más claro partiremos del widget de funcionamiento más simple que tenemos ahora mismo, el estático y estudiaremos sus funciones principales y más importantes que hacen que pueda ser dibujado.

-showInfoSettings(): Función que permite mostrar el menú de creación de una gráfica estática. Símplemente crea en el árbol DOM un html destinado a ser un formulario con el que permitir al usuario que use la aplicación seleccionar lo que le interesa representar.

|  |
| --- |
| function showInfoSettings(dash){ |
|  |

|  |
| --- |
|  |
| keys=configuration.static |
|  |
|  |
| $("#settings"+dash).slideUp("slow"); |
|  |
|  |
| $("#making").slideDown("slow") |
|  |
|  |
| if((Object.getOwnPropertyNames(takeinfo).length === 0) || (Object.getOwnPropertyNames(configuration).length === 0)){ |
|  |
|  |
| $("#conten").append('<div id="actualMenu"></div>') |
|  |
|  |
| $("#actualMenu").append('<p>Los ficheros de configuración no han sido cargados con normalidad. Compruebe su conexión<p>'); |
|  |
|  |
| $("#actualMenu").append('<button onclick="deleteCreation()" type="button" class="btn btn-xs btn-default">Continue</button>') |
|  |
|  |
|  |
|  |
|  |
| }else{ |
|  |
|  |
| if($("#actualMenu")){ |
|  |
|  |
| $("#actualMenu").remove(); |
|  |
|  |
| } |
|  |
|  |
| $("#conten").append('<div id="actualMenu"><p id="list" style="height: 200px; overflow-y: scroll;"></p></div>') |
|  |
|  |
| $("#actualMenu").append('<p><input type="radio" name="toSeeOptions" class="radios" value="column">Columnas <input type="radio" name="toSeeOptions" class="radios" value="bar">Barras </p>'); |
|  |
|  |
| keys.forEach(function(element){ |
|  |
|  |
| $("#actualMenu #list").append('<p><input type="checkbox" value="'+element+'"> '+element+'</p>') |
|  |
|  |
| }) |
|  |
|  |
|  |
|  |
|  |
| $("#actualMenu").append('<button onclick="takeDataInfo()" type="button" class="btn btn-xs btn-default">Create</button>') |
|  |
|  |
| $("#actualMenu").append('<button onclick="deleteCreation()" type="button" class="btn btn-xs btn-default">Cancel</button>') |
|  |
|  |
| } |
|  |

|  |
| --- |
| } |

* takeDataInfo(): Función que permite obtener los datos que necesitamos del menú creado en la función anterior, ligado al botón “Create” situado en la parte inferior del menú.

|  |
| --- |
| function takeDataInfo(numGraph){ |
|  |

|  |
| --- |
|  |
| var forma=$("#actualMenu input[type='radio']:checked").attr("value"); |
|  |
|  |
| if(forma!=undefined){ |
|  |
|  |
| var selected = []; |
|  |
|  |
| var forma=$("#actualMenu input[type='radio']:checked").attr("value"); |
|  |
|  |
| $("#actualMenu input[type='checkbox']:checked").each(function() { |
|  |
|  |
| objaux={ |
|  |
|  |
| name:$(this).attr('value'), |
|  |
|  |
| form: forma |
|  |
|  |
| } |
|  |
|  |
| selected.push(objaux); |
|  |
|  |
| }); |
|  |
|  |
| if(selected.length!=0){ |
|  |
|  |
| makeGraphInfo(actualDash,selected,numGraph) |
|  |
|  |
| }else{ |
|  |
|  |
| alert("¿Qué clase de gráfica pretende representar sin datos?") |
|  |
|  |
| } |
|  |
|  |
| }else{ |
|  |
|  |
| alert("Es indispensable que seleccione un tipo de forma a representar") |
|  |
|  |
| } |
|  |

|  |
| --- |
| } |
| - makeGraphInfo(): Función que nos permite crear el widget y dibujar su contenido. Hagamos especial énfasis en esta parte del código la cual resulta más interesante. Como hemos mencionado en este mismo apartado, existen problemas de sincronicidad a la hora de dibujar una gráfica una vez obtengamos un dato en concreto del json necesario. Gracias a las funciones de jquery .on() y .trigger() este problema se vio rápidamente solucionado permitiéndonos dibujar varias gráficas de manera simultánea o continuar creando widgets hasta que la descarag de los ficheros se completara.   |  | | --- | | //función que crea la gráfica en su conjunto tanto en error como en success | |  |  |  | | --- | |  | | function makeGraphInfo(dash,selected,graph){ | |  | |  | |  | |  | |  | | //si graph no está definido significa que creo una nueva gráfica | |  | |  | | if(isNaN(graph)){ | |  | |  | | numGraph+=1; | |  | |  | | graph=numGraph; | |  | |  | | var inDash= "dash"+dash.toString() | |  | |  | | var gridster = $("#"+inDash+" ul").gridster().data('gridster'); | |  | |  | | color=($("#dash"+dash).css('background-color')) | |  | |  | | gridster.add\_widget('<div id= "graph'+numGraph+'" class="panel panel-primary" style="border-style: groove;border-color: black;border-width: 3px"><div class="panel-heading" style="background-color:'+color+'"><button id="deleteButton" onclick="deleteGraph('+dash+','+graph+')" type="button" class="btn btn-xs btn-default">Delete</button><button id="settingsButton" onclick="settingsInfoGraph('+numGraph+')" type="button" class="btn btn-xs btn-default">Settings</button></div><div id="'+numGraph+'" class="panel-body"> </div></div>', 12, 8); | |  | |  | | }else{ | |  | |  | | //en otro caso cojo el chart ya creado y es el que uso para primero destruirlo y luego crearlo de nuevo | |  | |  | | var chart = $('#'+graph).highcharts(); | |  | |  | | chart.destroy() | |  | |  | | } | |  | |  | |  | |  | |  | | //Destruyo el actual menú de seleccion | |  | |  | | $("#actualMenu").remove(); | |  | |  | | $("#making").slideUp("slow") | |  | |  | | //En caso positivo lo dibujo | |  | |  | | $("#"+graph).on("DrawInfo",function(event,trigger,data){ | |  | |  | | var serie= parserGraphInfo(selected,data) | |  | |  | | drawGraphInfo(graph,serie,takeinfo.static.inside) | |  | |  | | $("#"+this.id).off() | |  | |  | | }) | |  | |  | |  | |  | |  | | //En caso negativo dibujo un widget roto | |  | |  | | $("#"+graph).on("ErrorGraphInfo",function(){ | |  | |  | | drawErrorWidget(this.id) | |  | |  | | $("#"+this.id).off() | |  | |  | | }) | |  | |  | |  | |  | |  | | //si no tengo los datos entonces los solicito | |  | |  | | if(takeinfo.static.state==0){ | |  | |  | |  | |  | |  | | //actualizo mi estado a en proceso | |  | |  | | takeinfo.static.state=1; | |  | |  | |  | |  | |  | | $.getJSON("templates/json/"+takeinfo.static.inside).success(function(data) { | |  | |  | |  | |  | |  | | //actualizo mi estado a terminado | |  | |  | | takeinfo.static.saveData=data | |  | |  | | takeinfo.static.state=2; | |  | |  | |  | |  | |  | | $("\*").trigger("DrawInfo",["DrawInfo",data]) | |  | |  | | }).error(function(){ | |  | |  | | //En caso de error levanto el evento de gráfica mal pintada | |  | |  | | $("\*").trigger("ErrorGraphInfo") | |  | |  | | takeinfo.static.static=0; | |  | |  | |  | |  | |  | | }); | |  | |  | | }else if(takeinfo.static.state==2){ | |  | |  | | //Si lo tengo en caché lo dibujo directamente | |  | |  | | var serie= parserGraphInfo(selected,takeinfo.static.saveData); | |  | |  | | drawGraphInfo(graph,serie,takeinfo.static.inside) | |  | |  | | } | |  |  |  | | --- | | } | |

- drawGraphInfo(): Función que permite dibujar el widget una vez lo tengamos todo cargado y los datos estén parseados. Aquí vemos un ejemplo de aplanamiento de datos simple donde se puede comprobar la simplicidad que ofrece Highcharts para dibujar un gráfico de manera sencilla y rápida.

|  |
| --- |
| //función que dibuja una gráfica de tipo Info |
|  |

|  |
| --- |
|  |
| function drawGraphInfo(id,serie,title){ |
|  |
|  |
| var options={ |
|  |
|  |
| chart:{ |
|  |
|  |
| renderTo:id.toString(), |
|  |
|  |
| width: 350, |
|  |
|  |
| height: 177 |
|  |
|  |
| }, |
|  |
|  |
|  |
|  |
|  |
| xAxis: { |
|  |
|  |
| categories: ["Total"] |
|  |
|  |
| }, |
|  |
|  |
| title: { |
|  |
|  |
| text: title |
|  |
|  |
| }, |
|  |
|  |
| series: serie |
|  |
|  |
| } |
|  |
|  |
| var chart= new Highcharts.Chart(options); |
|  |

|  |
| --- |
| } |