Con el avance de la aplicación nos vimos en la situación de que una vez concluida esta podría ser de utilidad a alguien desde el punto de vista externo para darle unos usos propios y añadir una serie de widgets personalizados. De esta manera en este ciclo se ha buscado una conversión total del código de centralizado a dividido en secciones.

Visto desde un punto de vista iterativo, los objetivos a cumplir en esta parte fueron:

-División del código y las gráficas en objetos

-Reestructuración de los paneles en formato objeto

-Recreación de los menús por parte de cada widget de manera individual y propia, totalmente ajena al código general del dashboard

-Aplanamiento de manera individual de cada widget, olvidándonos de cómo se estructura y cómo se recrea.

-Comprobación de dicha estructura creando dos nuevos widgets

La estructura a establecer de este modo siguió la siguiente jerarquía:

Panel {

var id;

var widgets;

var color;

var settings;

this.getSettings;

this.getId;

this.getWidgets;

this.flatten;

this.changeColor;

this.pushElement;

this.deleteElement;

this.deleteAll;

}

Lista de variables:

-id: Identificador propio del panel.

-widgets: Array de los widgets que posee. Se almacenan como objetos completos no aplanados.

-color: color de fondo del widget por si se deseara modificar.

-settings: barra de botones de creación del widget.

Lista de funciones:

-this.getSettings: Devuelve la barra de botones

-this.getId: Devuelve la id

-this.getWidgets: Devuelve la lista de widgets

-this.flatten: Función que devuelve el panel aplanado con todos sus elementos

-this.changeColor: Cambia el color

-this.pushElement: Incluye un nuevo elemento al panel

-this.deleteElement: Elimina un elemento del panel

-this.deleteAll: Elimina todos los elementos del panel

Widget {

this.id;

this.color

this.panel;

this.content;

this.buttons;

this.square;

this.gridsterWidth;

this.gridsterheight;

this.toDraw;

this.changeContent;

this.changeColor;

this.drawErrorWidget;

}

-this.id: Número de identificación del widget

-this.color: Color del widget

-this.panel: Id del panel al que pertenece

-this.content: Contenido del panel

-this.buttons: Botones del panel

-this.square: div de botones y contenido unido para implementarlo directamente en el gridster

-this.gridsterWidth: Ancho del widget

-this.gridsterheight: Alto del widget

-this.toDraw: Función para dibujar el widget en el gridster

-this.changeContent: Cambia el contenido

-this.changeColor: Cambia el color

-this.drawErrorWidget: Dibuja un widget de error

A partir de esta primera estructura genérica cualquiera puede aplicar las bibliotecas que necesite para la inclusión de nuevos widgets que se pueden aplicar a diversas tareas. En términos generales, lo más apropiado es que al aplicar una nueva biblioteca creemos un nuevo widget padre del que sacar los hijos para cada una de sus aplicaciones. Esto es debido a que por regla general en el entorno de las bibliotecas gráficas de representación las funciones para distintas gráficas suelen ser las mimas, y aun que su representación sea distinta su forma de modelarlas es semejante.

Con respecto a esto veamos un pequeño ejemplo con lo que tenemos entre manos. Tomando nuestra biblioteca predilecta HighCharts. En primer lugar debemos crear la clase padre con la que crear las subclases para la realización de los widgets de distintos tipos. Esto es lo que sigue.

HighWidget {

this.getSerie;

this.existLabel;

}

Como podemos comprobar apenas tenemos dos funciones, pero no necesitamos más. Estas funciones son las que unen a todos los widgets que hacen uso de esta biblioteca y que dando igual la naturaleza de lo que dibujemos las necesitaremos para formar los widgets a posteriori.

-this.getSerie: Obtiene la característica “series” de un widget ya pintado

-this.existLabel: Comprueba si una métrica existe dentro de un widget gráfico ya pintado.

A partir de este punto podemos crear los objetos hijos que se corresponden a dicha biblioteca. Dado que actualmente tenemos 3 tipos de gráficas cuyo código es bastante extenso, nos centraremos nuevamente en la más simple, la de tipo información estática.

HighInfo {

this.buttons;

this.square;

this.title;

var series;

this.flatten;

this.makeMenu;

this.takeData;

this.MakeWidget;

this.Parser;

this.Draw;

this.getSeries;

this.settings;

this.redraw;

}

-this.buttons: Variable con los botones nuevos adaptado al widget en concreto

-this.square: Variable con el recuadro nuevo adaptado al widget en concreto

-this.title: Titulo del chart

-var series: Listado de series y su formato para la creación del widget gráfico

-this.flatten: Función de aplanamiento para guardar el widget en un formato json

-this.makeMenu: Función para recrear el menú propio de este tipo de widget gráfico

-this.takeData: Función para obtener los datos de menú creado por el mismo widget

-this.MakeWidget: Función que permite al widget dibujarse a sí mismo.

-this.Parser: Función de parseo necesario para los datos propia de este widget

-this.Draw: Función para dibujar sobre un canvas una gráfica con la biblioteca que estamos usando

-this.getSeries: Devuelve las series guardadas del gráfico

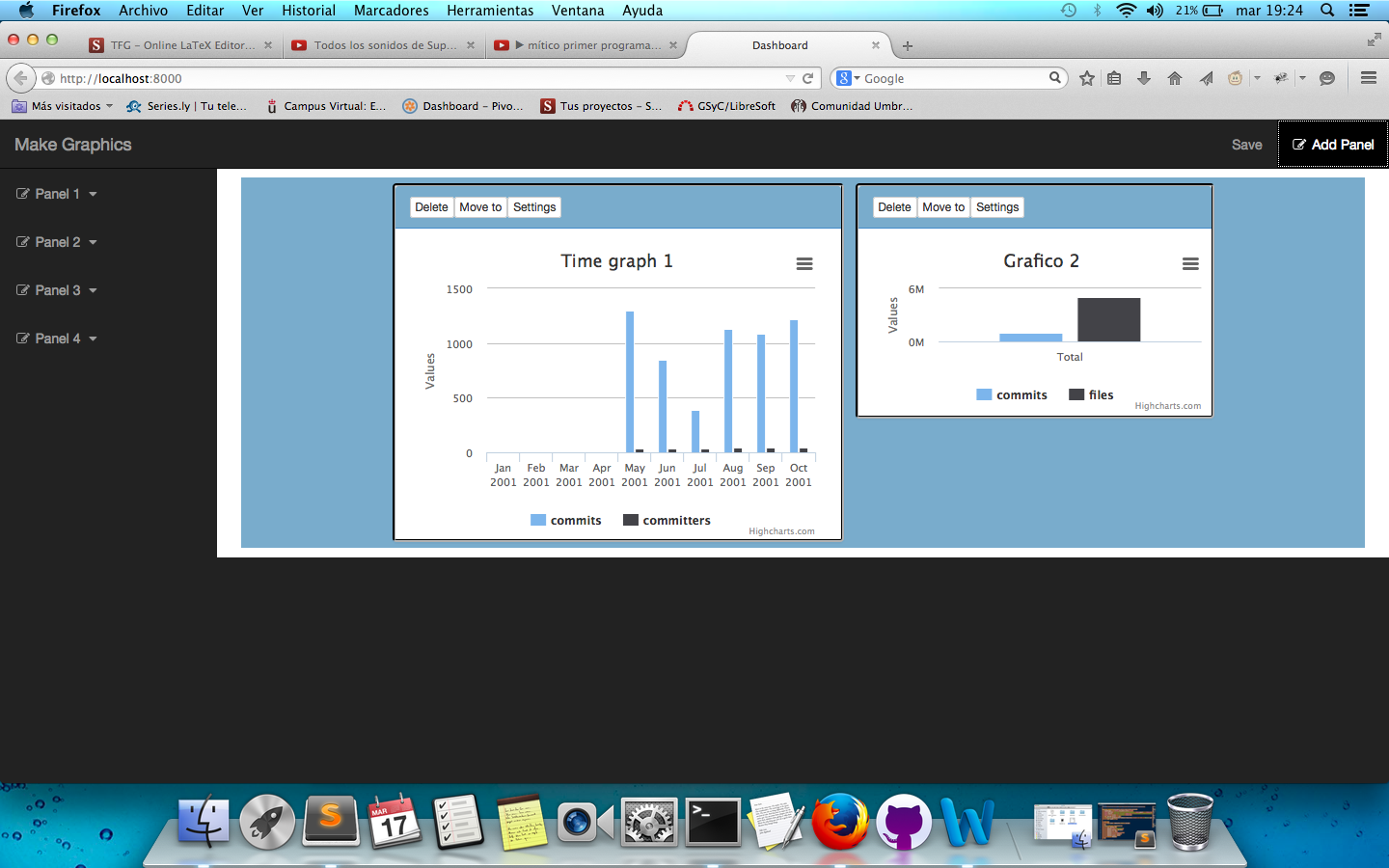
-this.settings: Reproduce un menú con las opciones que el widget está representando marcadas.

-this.redraw: Redibuja el widget con las nuevas opciones.

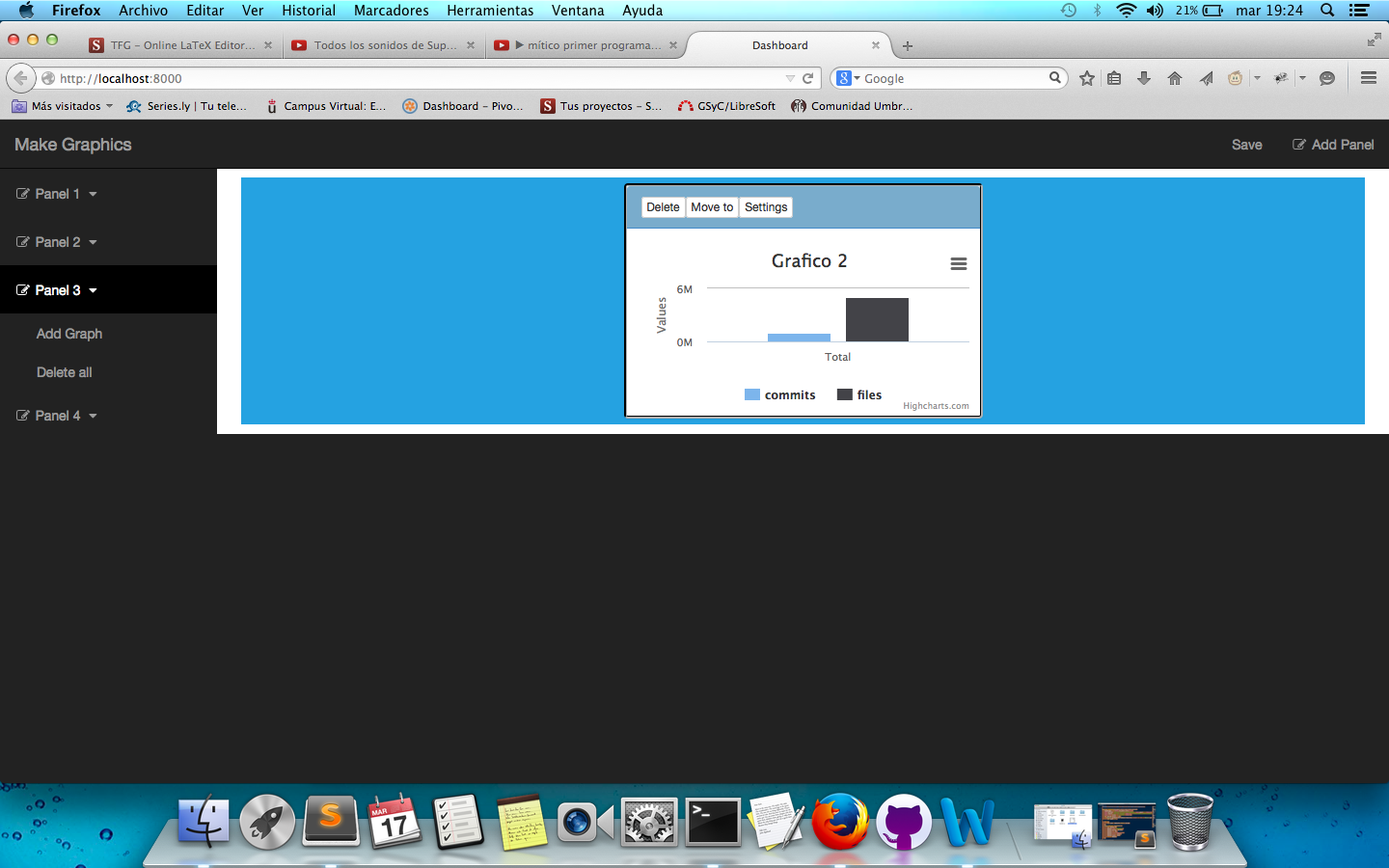
Este nuevo modelo simplifica notoriamente el funcionamiento de nuestro dashboard a la hora de introducir nuevos tipos de widgets, así como el aplanamiento de las gráficas para posteriormente ser capaces de reproducirlas nuevamente.

Por otro lado el tratar cada objeto de manera independiente nos facilita el manejo de las distintas gráficas, no solo para aplanarlas y guardarlas, si no también para ser capaces de cambiar las gráficas de lugar. Dicho esto veamos de manera más gráfica los resultados de esta iteración.

En el primer ejemplo veremos el movimiento de las gráficas del que hablamos. Como podemos ver en la imagen que hay a continuación tenemos presentadas dos gráficas, una de evolución temporal y otra de información estática.

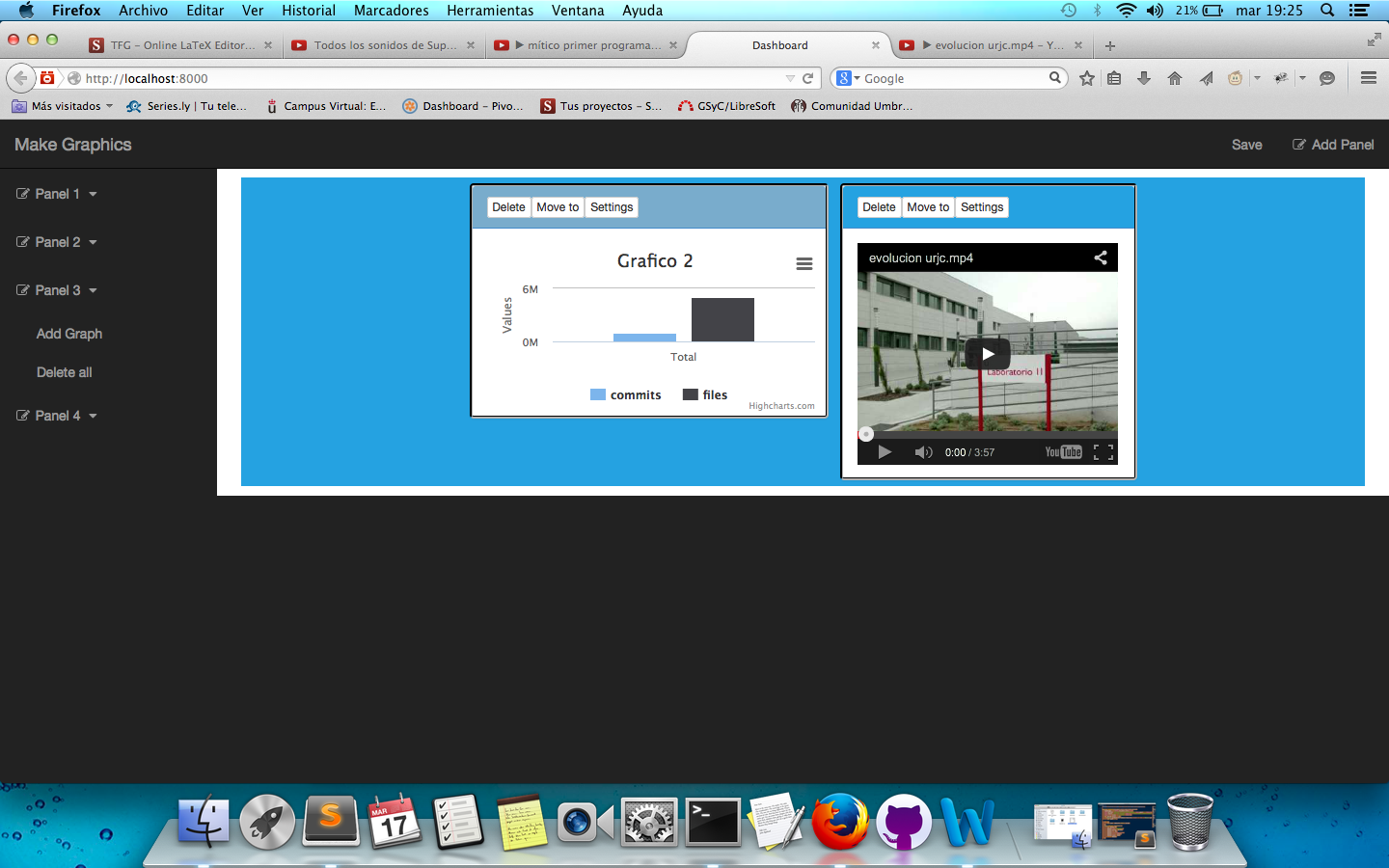


Creados 4 paneles distintos decidimos desplazar nuestro gráfico estático al panel 3.



Este se desplaza automáticamente una vez seleccionado el lugar donde queremos moverlo. Mantiene la apariencia, desde el color del widget, pasando por el tamaño y el formato de los datos.

Viendo que la jerarquía funciona de cualquier modo decidimos crear en este panel 3 un widget de video, que como hemos explicado anteriormente es hijo del widget principal, dado que en este caso no tenemos uso de nuestra biblioteca gráfica.



Por último decidimos ahora realizar un guardado del dashboard sobre el que estamos trabajando. Al guardar, se almacena del mismo modo que se hacía en el ciclo anterior, con la diferencia de que ahora cada widget se aplana por sí solo.

