Explicación de los scripts utilizados (Práctica de JQuery)

Carlos Ortega & Pablo Collado

A continuación pasamos a comentar las características más importantes de los diferentes scripts que hemos empleado para la realización de la práctica. Esperamos que sean de ayuda.

Archivo Gallery.js

Al haber podido utilizar JQuery vemos que el script que dinamiza nuestra página es relativamente sencillo. Quizá lo más importante para entender el funcionamiento de la galería estática es conocer la estructura con la que hemos insertado las imágenes. Cada elemento **<img>** está contenido en un ** ** con la clase **info_popup**. Además de la imagen encontramos otro ** ** con la clase **popup** dentro del contenedor con la clase **info_popup**. La estructura queda por tanto:

```
<div class=”info_popup”>
      <img>
      <div class=”popup”></div>
</div>
```

Gracias a las clases de los contenedores podemos referirnos a ellos de una manera más directa con los selectores de JQuery. Pasamos entonces a explicar paso por paso cada de las piedras angulares de nuestro script:

En la primera línea vemos que vamos a ejecutar una función para cada **popup** a través del método **each()**. Lo que queremos lograr es añadir un párrafo (** **) a cada **popup** y que contenga el texto alternativo que le demos a cada imagen. Así, en vez de “hardcodear” el texto que queremos que aparezca lo tenemos todo “centralizado” en el elemento **<img>**. Nuestras imágenes serán por tanto del tipo: **<img src=”...” alt=”¡Yo aparezco!”>**. Para conseguir acceder a este texto tenemos que recorrer el DOM hasta llegar al elemento HTML que representa la imágen y obtener el valor de su atributo **alt**.

Solo nos queda añadir los eventos correspondientes a las imágenes y los contenedores para que en función de la posición del ratón aparezcan o no. Para ello tenemos que seleccionar todos los

elementos **img** y establecer el manejador del evento **mouseenter** con una función. Lo único que tenemos que hacer es mostrar el contendor "hermano" de la imagen y recorriendo el DOM con el método **next()** llegamos a él sin problema.

Ahora tenemos que tener presente que una vez que ha aparecido el contenedor de la clase **popup** nuestro ratón ya no estará sobre la imagen. Por eso debemos establecer del manejador del evento **mouseleave** para cada contendor de la clase **popup** y no para las imágenes. Si hacemos esto último veremos cómo el **popup** aparece y desaparece cíclicamente…

Cabe destacar que hemos empleado los métodos **fadeIn()** y **fadeOut()** para que las transiciones de los **popups** sean más suaves. Además, en vez de cargar todos los **popups** y luego esconderlos con el método **hide()** hemos preferido aplicarles un estilo con **display: none;** para evitar la primera carga. Así evitamos ver que aparecen y desaparecen al principio mientras que los eventos funcionan como esperábamos.

Archivo Dynamic_gallery.js

En este caso la estructura difiere ligeramente de la anterior. Las imágenes que aparecen en la parte superior de la pantalla son elementos análogos a los del caso anterior pero la imagen que mostramos está representada a través de un solo contenedor con la clase **main_image**. Tras éste tenemos otro que contendrá el pie de foto. Al tener que manejar una estructura algo más compleja veremos cómo el código se alarga ligeramente.

Dejaremos la función **update_showcase()** para el final con lo que comenzamos viendo que añadiremos el texto a cada uno de los contendores que aparecerán encima de nuestra imagen de la misma manera que antes. Tras ello nos disponemos a configurar la situación inicial de la galería. Para ello vamos a añadir la primera imagen al contendor **main_image**. Para hacer dicho cambio debemos **clonar** los elementos para que no desaparezcan de la vista inferior.

Recorriendo el DOM le daremos la clase apropiada al contenedor que acabamos de añadirle a **main_image** y le retiraremos la anterior. Tenemos que hacer ésto porque la forma de visualizar una imagen depende del sitio en el que se ubique. Este cambio de apariencia lo hacemos a través de la adición y supresión de clases predefinidas en CSS.

Nos queda por tanto mostrar el contenedor que nos marca qué imagen está seleccionada y pasaremos a añadir los manejadores de eventos. Este paso es análogo a lo que hicimos para la galería estática. Tras los manejadores añadiremos al pie de foto el contenido del atributo **alt** de la imagen que estemos mostrando en ese momento con lo que terminamos de configurar la disposición inicial de nuestra galería.

Para darle funcionalidad a los botones hemos definido un manejador de eventos para cada uno en caso de ser clicados. Lo que harán será comprobar el valor del índice (definido como una variable global) para luego incrementarlo, decrementarlo o darle el valor apropiado. Para todo ésto hemos definido la constante **N_IMAGES** que se inicializa al principio del script como la longitud del array que nos devuelve el selector **"img"**. Es decir, nos da el número de imágenes de la página. Con esta estrategia conseguimos hacer el diseño extensible para añadir más imágenes. Tras trabajar con el índice cada manejador llamará a **update_showcase()** para que se actualice el contenedor **main_image**.

La función **update_showcase()** se encarga primero de eliminar todos los elementos que "cuelguen" de **main_image** para luego añadir el elemento que corresponda localizándolo con el valor de **index**. Ahora añadiremos los manejadores de eventos al elemento añadido. Tenemos que hacerlo porque no queremos que las imágenes de la parte inferior cambien su aspecto si les pasamos el ratón por encima... Al igual que antes tenemos que eliminar y añadir las clases correspondientes para que la imagen se muestre correctamente. Mostraremos también el contenedor que nos marca que una imagen está actualmente seleccionada no sin antes esconder todos los demás. EN vez de esconder solo el contenedor de selección anterior vemos que esta solución facilita el código pues no tenemos que distinguir los casos entre los en que hemos ido "de derecha a izquierda" y viceversa. Si bien es menos eficiente (escondemos elementos que no son visibles) facilita mucho la comprensión del código. Solo tenemos que eliminar el contenido del pie de foto y asignarle el que corresponda, cosa que logramos encontrando la imagen recorriendo el DOM y la función ya ha terminado.

Al igual que en el caso anterior hemos optado por aplicar el estilo **display: none;** a los contenedores para no tener que esconderlos todos al principio. Podemos señalar también que todo el código que hemos descrito, tanto de este archivo como del anterior, se encuentra en la función que le pasamos al método **ready()** del objeto **document** con lo que todo el código se ejecuta al

cargar la página. Tomamos ventaja de esta forma de ejecución como hemos visto para establecer una condición inicial sabiendo que tras una "primera pasada" solo ejecutaremos los manejadores cuando desencadenemos sus eventos asociados.

Archivo Glider.html

Decidimos que una de las mejoras de nuestra práctica fuera añadir una galería a través de un plugin. Hemos elegido Glider.js [https://nickpiscitelli.github.io/Glider.js/] y para inicializar el objeto que representa la galería hemos incluido un pequeño fragmento de código en el archivo HTML correspondiente. Hemos tomado esta decisión porque tan solo tenemos que llamar al constructor de un objeto **Glider**, nada más. Así, observamos que de forma parecida a lo que hacíamos con el método **ready()** de JQuery aquí le asignaremos un manejador al evento de "carga" de la página a través del método **addEventListener()** del objeto **window** que representa una ventana de nuestro navegador. Solo llamamos al constructor de un nuevo objeto **Glider()** con una serie de parámetros de configuración de nuestra galería. Lo más destacable es que tenemos que "decirle" qué contenedor representa nuestra galería a través de un selector que toma como parámetro la clase **.glider**. Toda la configuración se explica en la página referenciada.

Podemos comentar algo de la instalación del plugin aunque ésta ha consistido tan solo en copiar los archivos **.css** y **.js** necesarios en una carpeta para luego incluirlos. Podemos señalar que tuvimos que modificar ligeramente el archivo **glider.css** para adaptar la apariencia de la galería a nuestras necesidades cambiando una serie de márgenes y colores.

Con todo esperamos haber hecho un buen trabajo y que esta explicación lo haya clarificado todo. Gracias por su atención.

NOTA: Nótese que las galerías se han desarrollado para una pantalla con una alta densidad de píxeles por lo que para una correcta visualización puede ser necesario ajustar el zoom del navegador.

Archivo Tablas.js

De inicio, se ocultan todas las tablas existentes y únicamente se muestra la que corresponde a la página 1, la primera. A continuación, se establecen los estilos tanto de las tablas como de los botones existentes.

Posteriormente se encuentran las funciones correspondientes tanto a los botones para ocultar elementos como para mostrarlos. La primera de ellas hace uso del método **hide()** mientras que la segunda de ellas hace lo propio mediante el método **show()**.

Por últimos, existen 4 funciones, uno por cada uno de los botones existentes para cambiar de página. Estas muestran la tabla correspondiente al botón pulsado y ocultan las demás y también llevan a cabo un cambio de los estilos tanto de los botones como de los círculos indicadores de la página, resaltando con un color diferente los correspondientes a la página seleccionada.