CREANDO APPS

aprende a programar aplicaciones móviles

PARTE 2

GESTOS

Para empezar bien el curso, vamos a desarrollar nuestra primera aplicación: una app de gestos. Los gestos (en inglés *gestures*) son las interacciones que el usuario hace con la pantalla táctil. Con esta app vamos a ver cómo podemos detectar los gestos que hace el usuario y actuar en consecuencia. Además, aprenderemos cómo realizar algunas animaciones en nuestra app.

2.1 Gestos / EVENTOS Y FAST-CLICK

Para empezar, vamos a crear una aplicación muy sencilla con dos botones, que al hacer *click* (*tap* para dispositivos móviles) en ellos cambiarán el color de fondo de la pantalla de la app. Con esto aprenderemos a distinguir el click del tap. Además, veremos cómo hacer más responsiva (fluida) la interacción con nuestra app.

Partimos de una plantilla web, con HTML, CSS y JavaScript. Con el HTML para establecer la estructura de la app. En el CSS le daremos unos estilos. Y en el fichero JavaScript (JS) estableceremos el comportamiento (*behaviour* en inglés) de nuestra app. Todos están incluidos en una carpeta *www*.

En el HTML tenemos simplemente dos botones. Además, hemos enlazado un fichero CSS y otro JavaScript (a parte del fichero JS de Apache Cordova).

En el CSS le damos formato a los botones usando estilos. Además, preparamos dos estilos que podemos aplicar al *body* de la página para ponerle un color de fondo oscuro o claro.

En el JavaScript está el comportamiento de la app. Para ello, hemos creado un objeto que encapsula todas las funciones que necesitamos y que invoca a su función principal, `inicio`. Esta función `inicio` hace varias cosas:

- En primer lugar accede a elementos del HTML de la página (en adelante lo llamaremos DOM - Document Object Model) utilizando el método `document.querySelector` que recibe como parámetro un selector CSS. En este caso, usamos los selectores para acceder por identificador (`#identificadorDelElemento`) a los botones que hemos creado en el HTML.

- A continuación, sobre estas variables que tienen referencias a los botones en el DOM, invocamos el método `addEventListener` (añadir un escuchador de eventos). Esto indica qué evento vamos a escuchar sobre el elemento del DOM, y cómo lo va a gestionar. Escuchamos el evento “click” (primer parámetro) y lo gestionamos con una función (segundo parámetro, manejador de eventos) que se ejecutará al recibir un evento de este tipo sobre el botón.

2.1 Gestos / EVENTOS Y FAST-CLICK

Finalmente tenemos las funciones manejadoras de los eventos *click* sobre los botones. Ambas acceden al *body* de la página y le añaden un estilo para ponerlo claro u oscuro. Para ello, utilizamos el atributo className del *body*, que es la forma de referirse desde JavaScript al atributo class de HTML que tiene la información de estilo.

En este primer ejemplo hemos usado el evento click. Pero en un terminal móvil lo natural es el tap (toque), no el click. Para poder emular el click los navegadores de los terminales móviles esperan 300ms (milisegundos) para ver si de verdad es un tap o vas adesplazar el dedo para hacer cualquier otro gesto. Para evitar este retardo usamos una librería llamada Fast-click[1], que evita esa espera y lanza inmediatamente el evento click. De esta forma, la aplicación es mucho más responsiva.

Para utilizar Fast-click en nuestra app, tenemos que importar la librería en el HTML. Además, en el JavaScript escucharemos el evento DOMContentLoaded (contenido del DOM cargado) que nos indica que el dispositivo ya ha cargado y procesado completamente el HTML, CSS y JS de nuestra app. En este momento (al iniciarse este evento) ya podemos arrancar Fast-click y después nuestra aplicación. Podemos comprobar que la sensación al usarla es que la app es mucho más responsiva.

(1) Descarga Fast-click
`https://github.com/ftlabs/fastclick`

2.2 Gestos /

DETECCIÓN DE EVENTOS CON HAMMER.JS

A continuación vamos a añadir más funcionalidad a nuestra app de gestos. Añadimos una zona de gestos, de forma que los gestos que hagamos en esa parte de la pantalla vamos a detectarlos y mostrar su nombre por pantalla. Vamos a explorar varios de los gestos usados para interactuar con un dispositivo móvil:

- Tap: ya hemos visto que equivale a un toque.
- Double tap: toque doble.
- Pan: sirve para mover algo en horizontal, manteniéndolo apretado.
- Swipe: similar a pan pero más rápido se usa, por ejemplo, para pasar una página.
- Press: cuando pulsamos y mantenemos apretado.
- Pinch: con dos dedos, los separamos o juntamos para, por ejemplo, hacer zoom-in y zoom-out.
- Rotate: con dos dedos, sirve para rotar algo es una u otra dirección

Para detectar los gestos usamos una librería JavaScript llamada Hammer.js[2]. Para eso la cargamos desde nuestro HTML. Es conveniente utilizar la versión minificada (min) de la librería que es más compacta (el tamaño del archivo es menor) y tarda menos en cargar, aunque es casi imposible leerla porque el código está ofuscado. Si sólo vamos a utilizar la librería es muy útil usarla en este formato.

En el fichero JavaScript, hemos separado la lógica en 3 funciones que siguiendo buenas prácticas son pequeñas y con un nombre descriptivo de su comportamiento:

(2) Descarga Hammer.js
`http://hammerjs.github.io`

2.2 Gestos / DETECCIÓN DE EVENTOS CON HAMMER.JS

La primera, `iniciaBotones`, lanza los escuchadores de eventos sobre los botones.

La segunda, `inciaFastClick`, arranca FastClick.

La tercera, `iniciaHammer`, pone a funcionar la librería Hammer y es donde está todo el código nuevo de la app. Al iniciar Hammer le decimos que actúe sobre la zona de gestos que hemos Definido en el HTML. Para escuchar los eventos *pinch* y *rotate* le tenemos que pasar una opción especial. Para terminar, con el objeto de Hammer escuchamos los eventos táctiles (tap, doubletap, pan, swipe, press, pinch, rotate) y registramos una función escuchadora que se ejecutará cuando suceda el evento.

La función escuchadora de todos estos eventos lo único que hará será pintar el nombre del evento que sucede (tomado del parámetro `event`) en el elemento `h1` del HTML que hemos preparado para pintar esta información. El objeto evento[3] de Hammer.js transporta mucha información que podremos utilizar al reaccionar al evento, y que está detallada en la documentación de Hammer.

Al probar la aplicación en un dispositivo, nos damos cuenta de que los eventos swipe y rotate no aparecen. Esto es porque están enmascarados por otros eventos que suceden antes, *pan* enmascara a *swipe* y *pinch* a *rotate*. Por eso, Hammer te permite elegir sobre qué zona de la pantalla van a funcionar determinados eventos.

(3) Información del objeto evento en Hammer
`http://hammerjs.github.io/api/#event-object`

2.3 Gestos / ANIMACIÓN

Ahora vamos a terminar nuestra app realizando animaciones sobre la zona de gestos. Para realizar estas animaciones de forma nativa utilizamos animaciones CSS que tienen la ventaja de ser más eficiente que las que pudiéramos hacer con JavaScript.

En nuestro CSS vamos a añadir el código para las animaciones. En primer lugar realizaremos las de doubletap y press. Para ello especificamos los keyframes, es decir, los estados principales de la animación que indicamos envalor de %, siendo el 0% el inicio de la animación, el 50% la mitad y el 100% el final. Notad que en el código usamos el prefijo webkit en keyframes, y esto es debido a que estas animaciones aún no son estándares entre todos los navegadores y queremos asegurarnos de implementarlas tanto en navegadores tipo webkit como los de iOS y android. En ambos la propiedad CSS que modificamos es el color de fondo.

Para arrancar estas animaciones simplemente modificamos el estilo de la zona de gestos aplicando una clase CSS con la animación. Es importante que quitemos esta clase CSS al terminar la animación porque si no sólo podremos utilizarla una vez. Esto lo hacemos escuchando el evento `webkitAnimationEnd` que se lanza al terminar una animación, y cuando lo detectamos ponemos borramos las clases CSS de la zona de gestos.

Ahora pasamos a realizar animaciones más complejas para el swipe y el rotate. Para swipe seguimos usando keyframes pero creamos una animación para cada sentido del swipe: sobre la propiedad `width` (ancho) para el swipe a la izquierda o `margin-left` (margen a la derecha) para el swipe a la derecha. Para el evento rotate usamos una transformación para que gire 360º.

En el JavaScript tenemos que escuchar los eventos swipe, y a través de la información del objeto evento (propiedad `direction`) debemos aplicar la animación a la swipe-right o a la swipe-left. Para el caso del rotate, esperamos a que el usuario haya hecho un desplazamiento significativo para lanzar la rotación. Para controlar que sea significativo, comprobamos que la propiedad `distance` (distancia) del objeto evento sea mayor que un umbral. De esta forma, la animación no se ejecuta hasta que no llegamos a ese umbral y queda mucho más natural.

Con ésto hemos terminado la exploración de los gestos y las animaciones CSS.