GEOLOCALIZACIÓN, LOCAL STORAGE Y SESSION STORAGE

Geolocalización

Cada cosa que haces en Internet deja una huella. Me gusta llamar a esto un tracking digital que de ser aprovechado en una comunidad es la mejor forma de generar contenido único.

Si analizas los principales proyectos frecuentes en la web y la forma como aportas a su comunidad, detectarás que hay un posible tracking que ayuda a que grandes bases de datos se conviertan en páginas de contenido.

Detectar la ubicación exacta de un usuario te ayuda a personalizar su experiencia, a ofrecerle publicidad mucho mejor segmentada y a dejarle pistas junto a ese rastro digital que está generando. Y la geolocalización no depende de tener un GPS, no depende únicamente de un teléfono móvil, los navegadores han ido implementando esta capacidad con plugins y diversos proyectos nativos, hasta que HTML5 lo adoptó como una de sus opciones.

Con esta capacidad, el navegador hace uso de muchos métodos (GPS, Skyhook, Google Geo, IP) para darte la latitud y longitud de quien lo utiliza. Obviamente, requiere inicialmente el permiso del usuario para proceder. Y lo mejor es que funciona en cualquier PC, no sólo en teléfonos.

Pero, ¿cómo lograría geo-localizarme en un portátil en donde no tengo GPS?, Muy fácil, resulta que los carros de google streetview, los smartphones con Android, IOS y Windows phone tienen una función la cual se encarga de recoger datos de las redes Wi-Fi disponibles en el lugar que se encuentren y mandarlas a una base de datos con la geolocalización respectiva, luego cuando la api de HTML5 solicita permiso para compartir la ubicación, recoge las redes Wi-Fi disponibles

en tu laptop y las compara con las bases de datos del servidor (Donde previamente se habían geolocalizado los Routers Wi-Fi) y finalmente termina lanzando tu ubicación actual con un radio de 100 metros.

Ahora vamos al código

```
      <script type="text/javascript">
      var lat, lon, velocidad, vector;
      $(document).on("ready", inicio);
      function inicio() {
              geolocalizar();
      }
      function geolocalizar(){
              navigator.geolocation.getCurrentPosition(mostrar,
CorrerEnCirculos);
      }
      function CorrerEnCirculos(errorsh){
              alert("Te encontrare muy pronto")
              console.log(errorsh)
      }
      function mostrar(geo){
              lat             = geo.coords.latitude;
              lon             = geo.coords.longitude;
              velocidad       = geo.coords.speed;
              vector          = geo.coords.heading;
              document.write ('Latitud: ', lat, "<br />");
              document.write ('Longitud: ', lon, "<br />");
              document.write ('Velocidad: ', velocidad, "<br />");
              document.write ('Vector: ', vector);
      }
      </script>
```

Este es el código del JavaScript para obtener la latitud, longitud, velocidad (e.j: 30Km/h) y Vector (e.j: 30ºN) del dispositivo a rastrear. Básicamente la linea de código que nos da la localización seria:

```
      function geolocalizar(){
              navigator.geolocation.getCurrentPosition(mostrar,
CorrerEnCirculos);
      }
```

La cual nos arrojara dos casos, el primero en donde todo sale bien y obtenemos las coordenadas, velocidad y vector del dispositivo y las guardamos y mostramos en la función mostrar:

```
      function mostrar(geo){
              lat             = geo.coords.latitude;
```

```
            lon             = geo.coords.longitude;
            velocidad       = geo.coords.speed;
            vector          = geo.coords.heading;
            document.write ('Latitud: ', lat, "<br />");
            document.write ('Longitud: ', lon, "<br />");
            document.write ('Velocidad: ', velocidad, "<br />");
            document.write ('Vector: ', vector);
    }
```

Y tenemos el segundo caso es en donde el usuario no nos da permiso de compartir su ubicación y no obtenemos los datos imprimiendo en la consola el error en la función CorrerEnCirculos:

```
    function CorrerEnCirculos(errorsh){
            alert("Localizacion no permitida")
            console.log(errorsh)
    }
```

Demo: [html5demos]
[Mejorando.la nos hablo sobre esto.]

Local Storage y Session Storage

Cuando la web nació hace muchos muchos años de la mano de Tim Bernes Lee, nació como una mera transacción electrónica entre un cliente y un servidor: el cliente solicita un recurso, el servidor ofrece una respuesta. Listo, no hay más magia que esto. Para el servidor, una petición es indistinta de otra inmediata, y el ciclo comienza nuevamente. Es por esto que se dice que el protocolo HTTP no tiene estado, o sea, es stateless. Es nuestra responsabilidad como desarrolladores solventar esta limitación y asegurarnos de mantener el estado entre peticiones.

Para esta tarea, era muy habitual utilizar cookies, estos pequeños pedacitos de información que viajan en cada petición entre el cliente y el servidor, de manera de recordarle "Hey! ¿te acuerdas de mi?". Entonces, si tenemos el problema resuelto, ¿para qué necesitamos otro método nuevo? Bueno, resulta que las cookies no son tan buena solución:

Aumentan el peso de cada petición al servidor, ya que toda la

información guardada en las cookies debe viajar al servidor, y volver.Tienen una limitación de 4kb de espacio disponible.No todo el mundo tiene las cookies habilitadas, sobre todo luego de que se publicaran como ciertas empresas utilizan las cookies para registrar nuestro comportamiento en la web.
Por suerte, una de las nuevas específicaciones de la W3C y WHATWG, indica como podemos utilizar Javascript para guardar información en los navegadores de nuestros usuarios que expire al finalizar la sesión, o que no expire en lo absoluto a menos que el usuario lo indique.
Entran a la cancha localStorage y sessionStorage¿Qué son localStorage y sessionStorage? Puesto de manera sencilla, son dos métodos que nos ofrecen los navegadores para guardar pares de valores llave/valor, de manera sencilla y dándonos de manera standard unos 5 megabytes. Al igual que las cookies, esta información no desaparece si uno navega fuera de la página. Al contrario de las cookies, esta información queda guardada en la computadora del cliente, y no viaja con cada petición.

No hay excusas para no utilizar ninguno de los dos métodos, ya que se encuentran soportados por prácticamente todos los navegadores, excepto OldIE y Safari 3.2.

Comenzar a utilizarlo es muy sencillo:

```
/* Utilizando la Api de Local Storage, podemos
   guardar y luego recuperar un valor */

localStorage.setItem('saludo', 'Bienvenidos a localStorage');
valor = localStorage.getItem('saludo');
/* Pero esta no es la única manera, ya que
   también podemos hacer lo siguiente */

localStorage['saludo'] = 'Este saludo también es válido';
valor = localStorage['saludo'];
/* Y claro, esto también es válido */
localStorage.saludo = 'Y este es mi método favorito';
valor = localStorage.saludo;
```

Pero esto no es lo único que podemos hacer, ya que las APIs de Local Storage como la de Session Storage nos ofrecen un par de métodos más

para controlar la información que contienen.

```
/* Podemos saber cuántos elementos tenemos guardados */
alert('Tenemos ' + localStorage.length + ' elementos dentro de Local
Storage');
/* Eliminar elementos también es muy sencillo de lograr */
localStorage.removeItem('saludo');
/* Y si queremos acceder a la información de localStorage
   de manera secuencial, entonces lo podemos hacer gracias
   a su método 'key' que devuelve la clave en determinada posición */

for(var i=0, t=localStorage.length; i < t; i++) {
    key = localStorage.key(i);
    alert('Para la clave ' + key + ' el valor es: ' + localStorage[key]);
}
```

Vale aclarar que todo lo mismo vale en este punto, si queremos trabajar con sessionStorage, que guardará los valores durante el tiempo de session del navegador, y se borrarán cuando el navegador se reinicie. Esto no sucede con localStorage, que conservará los valores aunque cerremos y volvamos a acceder al navegador.

Cómo cargar local Storage de manera más inteligente.
Luego de estar trabajando un tiempo con localStorage o sessionStorage, comienza a notarse un patrón cuando queremos asegurarnos de la disponibilidad del Storage. Al comienzo hacemos las siguientes pruebas:

```
if('localStorage' in window && window['localStorage'] !== null) {
    alert('Genial, tenemos un navegador decente que soporta
LocalStorage);
    var storage = localStorage
} else {
    alert('Como seguimos utilizando un navegador viejo, Santa Claus no
nos traerá nada esta Navidad');
}
```

Como puden ver, el utilizar el "if" para chequear la existencia de localStorage se puede volver un tanto molesto y un "anti-patrón". ¿Y qué pasa si no tenemos soporte para localStorage? ¿Nuestra aplicación debe fallar? ¿O deberíamos ofrecer una solución que degrade de manera "grácil"?

Aquí es donde surge la necesidad de trabajar con un método que sirva ya en tanto tengamos soporte, o no. Y que no de errores al momento de chequear la disponibilidad de localStorage. Este es el código que estoy utilizando en mis desarrollos:

```
// Existe localStorage?
var storage;
try {
    if (localStorage.getItem) {
        storage = localStorage;
    }
} catch(e) {
    storage = {};
}
```

De esta manera, dentro de la variable storage tendremos o una referencia a localStorage lista para utilizar, o un objeto que no reemplazará la funcionalidad de localStorage, pero al menos no dará error inmediatamente cuando lo utilicemos, para luego implementar alguna librería que de soporte, como Amplify.js.

No Strings Attached

Una de las limitantes más molestas de localStorage y sessionStorage es que, al igual que con las cookies, solo podemos guardar cadenas de texto. Nada de valores nativos como enteros, float, ni siquiera un triste boolean (que sería "false"). Ni hablar de los objetos:

```
persona = {nombre: 'Pablo Rigazzi', edad: 33};
localStorage.autor = persona;
alert(localStorage.autor);
// Esto nos devuelve el texto
"[object Object]"
```

Entonces, ¿Cómo podemos hacer para superar esta limitante? Afortunadamente, hay una técnica bastante simple para guardar objetos completos dentro de localStorage, para luego recuperarlos y volver a

convertirlos, y este método es utilizar el objeto JSON, disponible en los mismos navegadores que soportan localStorage. Retomando el ejemplo anterior:

```
// Primero convertimos el objeto en una cadena de texto
localStorage.autor = JSON.stringify(persona);
/* Y ahora, al recuperarlo, convertimos el string
   nuevamente en un objeto */
var autor = JSON.parse(localStorage.autor);
alert(typeof autor);
// Aqui obtendremos que el tipo de la variable es "object"
```

Vientos de cambio

Los navegadores nos dan un mecanismo para enterarnos cuando otra página realizó cambios sobre los valores de localStorage (y solo localStorage). Para esto, lanzan un evento llamado "storage" que es muy fácilmente atrapable. Lo malo, no es un evento "cancelable", esto es, el navegador solamente nos avisa del cambio, pero no tenemos manera de impedir que ese cambio se realice. Hay que tener en cuenta que este evento se dispara si el cambio se realizó en otra página (abierta en otra ventana o en otro tab), que esté asociada al mismo localStorage con el que estamos trabajando (o sea, es del mismo dominio).

```
hay_cambio = function(e) {
    console.log("Dentro de la clave " + e.key + " el valor anterior era "
+ e.oldValue + ' y el nuevo es ' + e.newValue);
}
if(window.addEventListener) {
    window.addEventListener('storage', hay_cambio, false);
} else {
    // Hay que soportar IE6, 7 y 8, lo lamento :P
    window.attachEvent('onstorage', hay_cambio);
}
```

Aquí es donde para probar, podemos abrir otra ventana del mismo dominio donde estemos trabajando, disparar FireBug o la consola de Chrome, y tipear: localStorage.clave = "nuevo valor.", y ver como se dispara el evento en el tab o ventana original.

¿Qué ejemplos para utilizar Session Storage y Local Storage se nos ocurren?

Guardar resultados temporalmente de una llamada a un webservice. Mantener estado de la interfaz de usuario de manera sencilla.Referencias

SmashingMagazine
Paper Killed Rock
W3C
Dive Into HTML5