UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería Informática

TRABAJO FIN DE GRADO

Captura de contexto en sesiones Web mediante inyección de código

José Luis González González-Cela

Tutor: Alejandro Bellogín Kouki

Ponente: Fernando Díez Rubio

Junio 2017

Captura de contexto en sesiones Web mediante inyección de código

AUTOR: José Luis González González-Cela

TUTOR: Alejandro Bellogín Kouki

Dpto. Ingeniería Informática

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Junio de 2017

Resumen

En este Trabajo Fin de Grado se desarrolla la idea de dinamizar la web de manera externa en base a la navegación que realice el usuario. Se explica el desarrollo de un código que permitirá dinamizar la web creando elementos externos que alteren la web, mediante avisos al usuario.

El objetivo principal es que la dinamización de la web sea lo más sencilla posible. Para ello se ha desarrollado un código genérico que funcionará para cualquier dominio. La dinamización se definirá en otros ficheros de forma externa a estos, pudiendo ser modificados de forma sencilla para alterar la dinamización de la misma. La fácil manipulación de estos ficheros haría posible que sean editados por usuarios con escasos conocimientos en las tecnologías de la World Wide Web.

El proceso de dinamización que se ha desarrollado se basa en dos elementos que aparecerán en la navegación del usuario en base a unas condiciones establecidas de manera externa en los ficheros de configuración. Estos elementos serán mensajes de *alert* propios de JavaScript con avisos al cliente y popups emergentes con páginas webs diferentes que aporten contenido a la navegación del usuario.

Con esto se conseguirá poder realizar cambios en la web sin necesidad de realizar cambios en el código de la página, ya que de forma externa a ella y únicamente modificando los ficheros de configuración se conseguirán cambios de manera rápida y sencilla.

Palabras clave

Contexto de navegación web, cookies, inyección de código, dinamización web.

Abstract

In this Bachelor Thesis we have developed the idea of dynamizing the web in an external way based on the navigation made by the user. Here we explain the code development that makes dynamizing the web possible by creating external elements that alter the web, by showing messages to the user.

The main objective of this work is to make the dynamization of the web as simple as possible. For this, a generic code has been developed that will work for any domain. The dynamization will be defined in external files, which can be modified in a simple way to alter the dynamization of the webpages. The manipulation of these files is straightforward so they could be edited by users with little knowledge in the World Wide Web technologies.

The dynamization process that has been developed is based on two elements that will appear in the navigation of the user based on the conditions established externally in the configuration files. These elements will be JavaScript alert messages and popups with different web pages that provide content to user navigation.

Once this goal is achieved, we will be able to make changes on the web without having to modify the page source code. The changes will only be done on the external configuration files that will change the site quickly and easily.

Keywords

Context of web navigation, cookies, code injection, web dynamization.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

1 Introducción

1.1 Motivación

El uso de la World Wide Web crece a un ritmo exponencial desde que en 1990 Tim Berners-Lee concluyese su proyecto. Poco queda de aquellas primeras webs, desde la invención de nuevos lenguajes de programación para las webs como JavaScript o HTML5, las páginas web han ido cambiando de un modelo estático a uno cada vez más dinámico.

Para modificar el dinamismo de una web es necesario tener conocimientos acerca de cómo está desarrollada la misma y sobre los lenguajes que la cimientan. Centrándonos en ese dinamismo y en que estos nuevos lenguajes son desconocidos para el público en general, surge la necesidad de poder dinamizar de manera externa y sencilla una web, sin tener conocimiento avanzado en las tecnologías actuales.

1.2 Objetivos

El objetivo de este proyecto es dotar a las páginas web de un pequeño grado de dinamismo, en particular, que sea posible editarlo por personas sin amplios conocimientos en las tecnologías de la World Wide Web. Para ello haremos uso de estas tecnologías automatizando procesos, haciéndolos transparentes al usuario y consiguiendo, por tanto, una personalización de la web de manera rápida y sencilla.
Se generarán mensajes dinámicos en la navegación sobre el dominio en el cual esté funcionando el proyecto, y podrán ser editados de manera sencilla, donde únicamente modificando archivos de configuración se podrá cambiar la experiencia del usuario en la web, pudiendo, por ejemplo, guiarle a través de nuestro sitio a donde más nos interese en base a las acciones que éste haya realizado.

1.3 Organización de la memoria

La memoria de este proyecto consta de los siguientes capítulos.

- Capítulo 2 (Estado del arte): Exposición acerca de las tecnologías actuales que más se amoldarían al desarrollo de este proyecto, breve estudio de las mismas y conclusiones acerca de cuáles se han usado y la motivación que nos ha llevado a usarlas.

- Capítulo 3 (Análisis y Diseño): Explicación del análisis previo realizado sobre el proyecto. Presentación de aquellas decisiones de diseño tomadas y breve explicación de las mismas.
- Capítulo 4 (Desarrollo): Explicación detallada del proceso de desarrollo de este proyecto, haciendo hincapié en los scripts que se han desarrollado y la funcionalidad que desempeña cada uno en el conjunto global del proyecto.
- Capítulo 5 (Pruebas y resultados): Explicación de las pruebas realizadas sobre el proyecto en casos diferentes y conclusiones finales tras observar los resultados de las mismas.
- Capítulo 6 (Conclusiones y trabajo futuro): Conclusiones obtenidas tras la finalización del desarrollo, pautas a seguir en un trabajo futuro con el fin de realizar una versión más completa y operativa del proyecto.

2 Estado del arte

2.1 Frameworks de desarrollo

En esta sección se va exponer a modo de resumen el estado de los frameworks de desarrollo actuales y la evaluación de los mismos. Del mismo modo, se expondrá de manera más detallada aquellos que se han utilizado y los motivos que han llevado a elegir estos y no otros para el desarrollo del proyecto.

2.1.1 Extensiones de navegadores para inyección de código.

Entre la multitud de extensiones desarrolladas para cada uno de los muchos navegadores existentes a día de hoy, existen unas extensiones particulares que permiten al usuario ejecutar scripts que alteran las páginas que visita pudiendo estas cambiar o ser personalizadas a gusto del usuario. Estas extensiones no realizan acciones por sí mismas, únicamente permiten ejecutar el código que previamente debemos haber desarrollado. Existe una gran colección de los denominados “Scripts de usuarios”, que no dejan de ser los scripts que ejecutan estas extensiones, de dominio público que podremos descargar desde la web http://userscripts-mirror.org/index.html y añadir a nuestra extensión sin necesidad de desarrollar un script particular.

2.1.1.1 GreaseMonkey.

GreaseMonkey es la principal y la primera extensión para inyección de código. Su primera versión fue creada por Aaron Boodman en 2005 y en la actualidad la última versión estable data de julio del 2015. Se trata de una extensión desarrollada para el navegador Web Mozilla Firefox [15]. Mediante esta extensión se puede añadir scripts al navegador y seleccionar en qué dominios queremos que tengan efecto. El principal objetivo de la misma, al igual que del resto de extensiones de este tipo, es posibilitar la ejecución de código JavaScript denominado “Script de usuario” en una página o dominio determinado [6].

La extensión nos permite activar o desactivar cualquiera de las extensiones que hemos desarrollado, así como modificarlas directamente desde la extensión.

Los scripts para GreaseMonkey deben respetar una serie de reglas que nos predefine la extensión, como, por ejemplo, rellenar correctamente la cabecera de los scripts de acuerdo

a la documentación de GreaseMonkey y la extensión de los archivos, que debe ser X.user.js

2.1.1.2 TamperMonkey

Tampermonkey es la principal extensión de inyección de código que nos ofrece el navegador Google Chrome. En un principio era una extensión exclusiva de Google Chrome, pero recientemente han sacado una versión para Mozilla Firefox, pudiendo usar los mismos scripts en ambos navegadores Web.
Su lanzamiento fue posterior al de GreaseMonkey, pero su funcionalidad es la misma. Donde tiene diferencias es en los scripts de usuario, dado que varía el formato que tienen los scripts respecto a los de GreaseMonkey.

2.1.1.3 ViolentMonkey

ViolentMonkey es una extensión publicada en el año 2013 como solución para adaptar GreaseMonkey al navegador Opera, basado en este mismo. Esta extensión comparte gran cantidad de scripts con GreaseMonkey, por lo que es posible añadir las extensiones desarrolladas para Firefox al navegador Opera [13].

2.1.1.4 Scriptish

Scriptish es una extensión desarrollada a partir de GreaseMonkey y publicada en el año 2010 para el navegador Mozilla Firefox. Está basada en la anterior, dado que parte de la misma, pero cambia el enfoque de GreaseMonkey. Esta extensión nace con el objetivo de crear una extensión especifica con el fin de modificar el modelo DOM de la web, lo que es básicamente modificar la presentación de una determinada web al usuario.

2.1.1.5 Scriptify

Scriptify es una extensión para Mozilla Firefox que permite dentro de una misma extensión ejecutar scripts de usuario realizados tanto para Scriptish como para GreaseMonkey [11].

2.1.1.6 NinjaKit

Se trata únicamente de una adaptación de la extensión GreaseMonkey de Mozilla Firefox para el navegador Safari [10].

2.1.2 Frameworks para el desarrollo en JavaScript

2.1.2.1 JQuery

JQuery es una biblioteca multiplataforma de JavaScript que permite simplificar la manera de interactuar con elementos de HTML con el código JavaScript [6].

Es una biblioteca de software libre y código abierto, por lo que está constantemente recibiendo actualizaciones por parte de una comunidad que con el paso de los años se ha ido haciendo más numerosa. Está orientada a la manipulación del código HTML y CSS desde scripts cambiando u obteniendo información de la página sobre la cual se ejecuta.

Las principales características de esta biblioteca serían:

- Seleccionar los elementos del DOM de una manera más sencilla respecto al lenguaje JavaScript "puro".
- Manipulación de forma cómoda de la hoja de estilo CSS que define el estilo de una página HTML.
- Compatibilidad con la gran mayoría de navegadores del mercado.

A la hora de hacer uso de esta biblioteca lo más característico es la función $() [9].

Esta función recibe como parámetro una etiqueta del HTML o una expresión en CSS, y nos devolverá la expresión que se corresponda con el parámetro pasado como argumento[7].

2.1.2.2 Backbone

Backbone es una herramienta de desarrollo para JavaScript centrada en el paradigma Modelo-Vista-Controlador. Su diseño está orientado al desarrollo de *single-page application* (SPA), aplicaciones web con una sola página con el propósito de dar al cliente una experiencia más fluida. Este framework nació a partir de *DocumentCloud*, y es también una biblioteca de código abierto.

2.1.2.3 EmberJS

EmberJS también es una herramienta de desarrollo para JavaScript basada en el paradigma Modelo-Vista-Controlador. Permite el desarrollo de aplicaciones escalables dentro de la misma página web.

2.1.2.4 AngularJS

AngularJS es un framework de JavaScript con el propósito de crear aplicaciones Web de una sola página (SPA) basándose en el Modelo-Vista-Controlador (MVC). Esto le sirve

para que el desarrollo y las pruebas sean más sencillos para los usuarios que utilicen esta biblioteca.

Esta biblioteca lee el HTML mediante las etiquetas y las transforma en variables estándar de JavaScript, pudiendo ser modificadas o guardadas como recursos en objetos JSON.
También se utiliza para combinar un entorno en tiempo de ejecución como Node.js, el framework de servidor Express.js y la base de datos no relacional MongoDB. Esta combinación de elementos forma el conjunto MEAN, conjunto para desarrollo de aplicaciones y páginas web dinámicas.

2.1.2.5 NodeJS

NodeJS es un entorno en tiempo de ejecución multiplataforma de código abierto para la capa del servidor. Está basado en el lenguaje ECMAScript y fue creado para el desarrollo de programas de red escalables, como pueden ser los servidores Web. La principal característica que tiene es que, a diferencia de la mayoría de frameworks de JavaScript, no se ejecuta en el lado del cliente, si no que está pensado para ejecutarse en el lado del servidor.

2.1.2.6 Vanilla JavaScript

Vanilla JavaScript es un sinónimo de lo que se conoce corrientemente como JavaScript puro. Se trata únicamente de código que se ejecuta del lado del cliente sin ayuda de ningún framework para su desarrollo.

2.1.3 Conclusiones

De entre todas las extensiones para inyección de código en el navegador, se ha optado por utilizar GreaseMonkey. La motivación para utilizar esta extensión y no otra radica básicamente en que es la pionera entre este tipo de extensiones, por lo que su desarrollo se ha prolongado más en el tiempo que ninguna, y cuenta con una comunidad de usuarios más grande que la del resto de sus competidoras.

Otra razón que nos lleva a elegir GreaseMonkey frente al resto es que la mayoría de extensiones surgen a partir de esta, por lo que la calidad de sus versiones estables ha permitido multitud de variantes convirtiéndola en el estándar al ser la base de la mayoría.

A la hora de elegir un framework de desarrollo de JavaScript, dado su comodidad a la hora de interactuar con ficheros HTML, parte importante en la realización de este proyecto, se ha elegido utilizar JQuery. El uso de JQuery nos permitirá acceder a los elementos del DOM de manera fácil y poder manipularlos sin la necesidad de extender nuestro código de manera innecesaria.

Otro argumento de peso para seleccionar JQuery y no otro framework radica en la comunidad que tiene y en la cantidad de sitios Web que actualmente hacen uso del mismo [14].

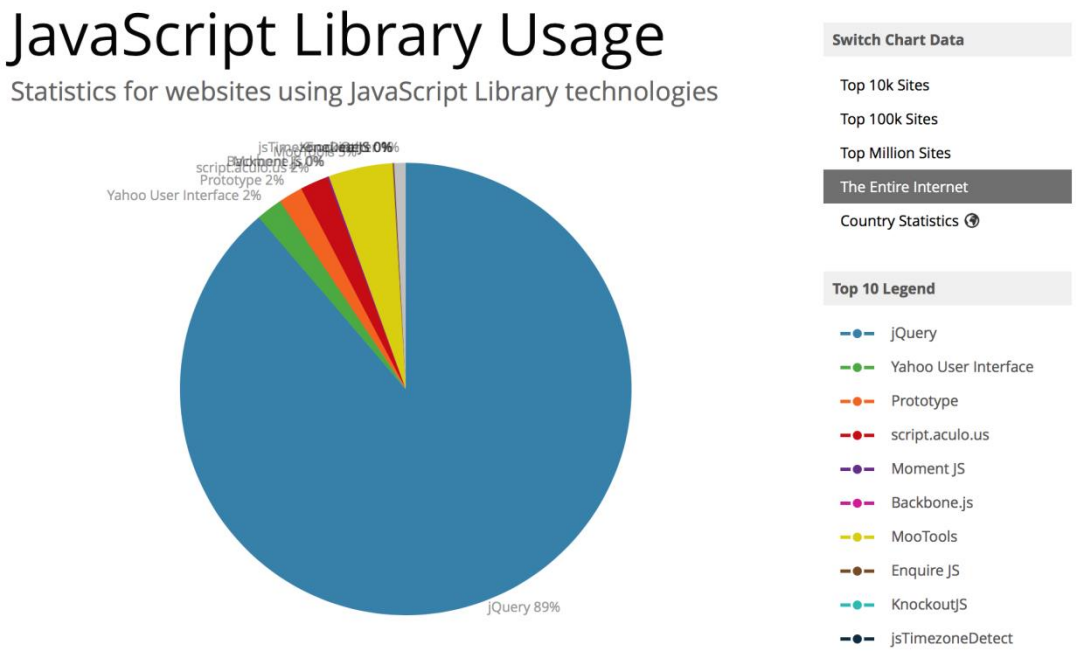

Figura 2-1: Gráfico frameworks JavaScript

En este gráfico se puede observar el uso que en la actualidad tiene JQuery frente otras librerías de JavaScript.

Por tanto, dado que son los dos elementos que más se amoldan y mayores facilidades proporcionarán a la hora de desarrollar el proyecto, se ha elegido utilizar GreaseMonkey y ficheros JavaScript utilizando JQuery [5].

3 Análisis y Diseño

3.1 Análisis

El proyecto debe permitir la ejecución de acciones dentro de una web mediante la inserción de un código externo. Para ello será necesario el uso de un servidor donde se aloje el código a inyectar.

3.1.1 Requisitos funcionales

RF1 El administrador del proyecto podrá realizar conexiones al servidor y modificar los archivos en el mismo.

RF2 No se le mostrarán popups, ni alerts repetidos a un cliente en una misma sesión Web.

RF3 El proyecto será fácilmente adaptable a varias webs para la realización de diferentes demos.

RF4 Todos los procesos que realizan el proyecto serán transparentes para el cliente, únicamente observará las acciones derivadas de aquellas reglas que se cumplan.

3.1.2 Requisitos no funcionales

RNF1 El proyecto debe funcionar en el navegador Mozilla Firefox.

RNF2 Su funcionamiento no dependerá del sistema operativo que se utilice.

RNF3 Se almacenará información en el navegador de los usuarios con el fin de registrar su actividad.

RNF4 La ejecución de los scripts debe ser síncrona y en un intervalo de tiempo no mayor a los dos segundos.

3.1.3 Casos de uso del administrador

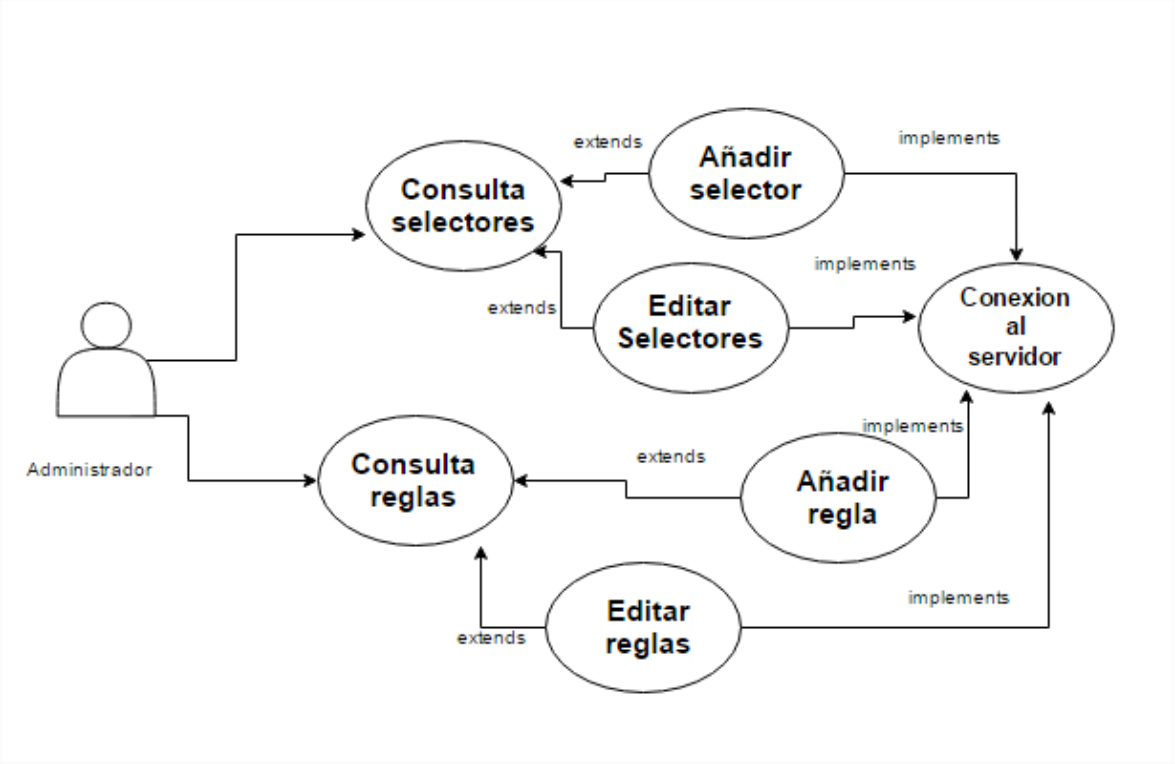

Figura 3-1: Casos de uso

El administrador deberá tener acceso al servidor, y podrá realizar las acciones descritas en el diagrama de casos de uso. En un futuro se pretende que se haga de forma automática mediante páginas en el servidor, pero actualmente es necesaria la edición de los ficheros por parte del administrador

3.2 Diseño

El objetivo de este proyecto es dinamizar la experiencia del cliente mediante inyección de código y facilitar al administrador de la Web esa dinamización. Para ello, este proyecto se centra en dinamizar la página de manera "externa", es decir, sin que necesitemos modificar el código de la página original. Para ello nos serviremos de una serie de ficheros en JavaScript que estarán en ejecución en el navegador del cliente, de manera totalmente transparente para él. Estos ficheros estarán alojados en un servidor al cual el administrador de la página accederá para cambiar los eventos que sucederán durante la navegación del usuario.

A partir de esa idea se ha creado un prototipo que funciona en modo local. El código es insertado mediante la extensión del navegador GreaseMonkey; ya tratada con anterioridad, y el servidor será un servidor local montado sobre nuestra máquina.

3.2.1 Diseño de red

En esta sección se explica cómo funciona el prototipo que hemos desarrollado.
En primer lugar, se hará uso de un servidor, en el caso particular de este prototipo se ha realizado sobre un servidor de manera local. En este servidor estarán alojados los ficheros encargados de realizar las acciones predeterminadas, así como aquellos archivos destinados a realizar el rastreo correspondiente de la página HTML.

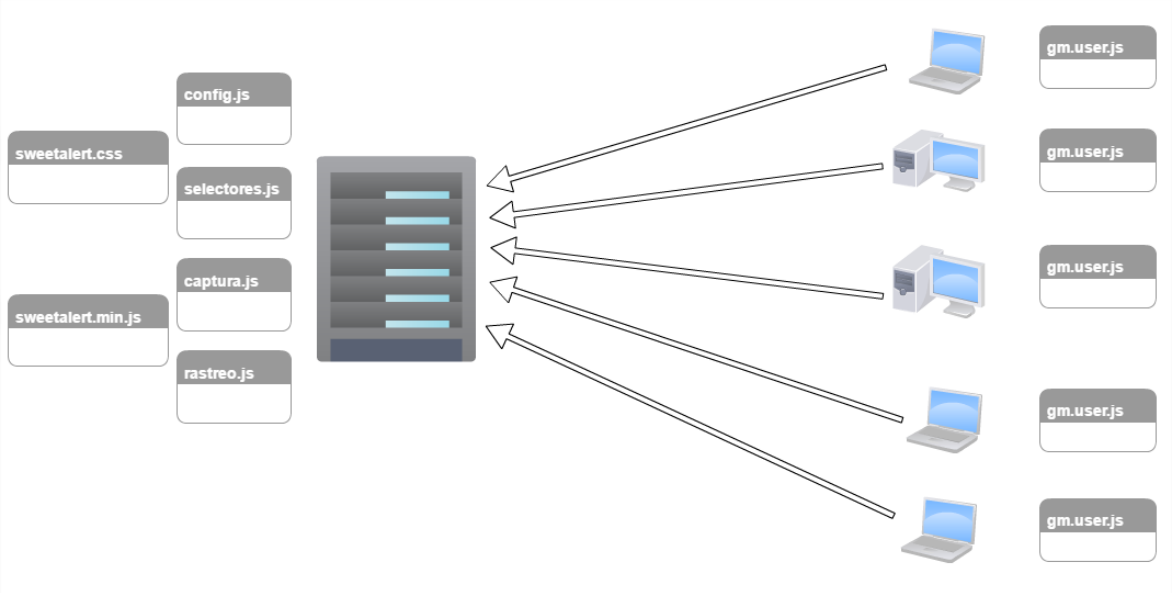

Figura 3-2: Mapa de red

Como se puede observar en la **Figura 3-2**, el cliente debe tener instalado y activado el script de usuario *gm.user.js*, el cual funcionará sobre la extensión GreaseMonkey del navegador Mozilla Firefox.

Desde este script de usuario se realizarán cinco peticiones al servidor de los archivos indicados en el diagrama, salvo la del fichero rastreo.js, la cual se realizará más adelante durante la ejecución de *captura.js*. Si alguna petición de las cinco fallase, el proyecto no se ejecutaría de manera correcta. Tras realizar las peticiones almacenará en caché los ficheros y comenzará la ejecución de los mismos después de finalizar el script de usuario.

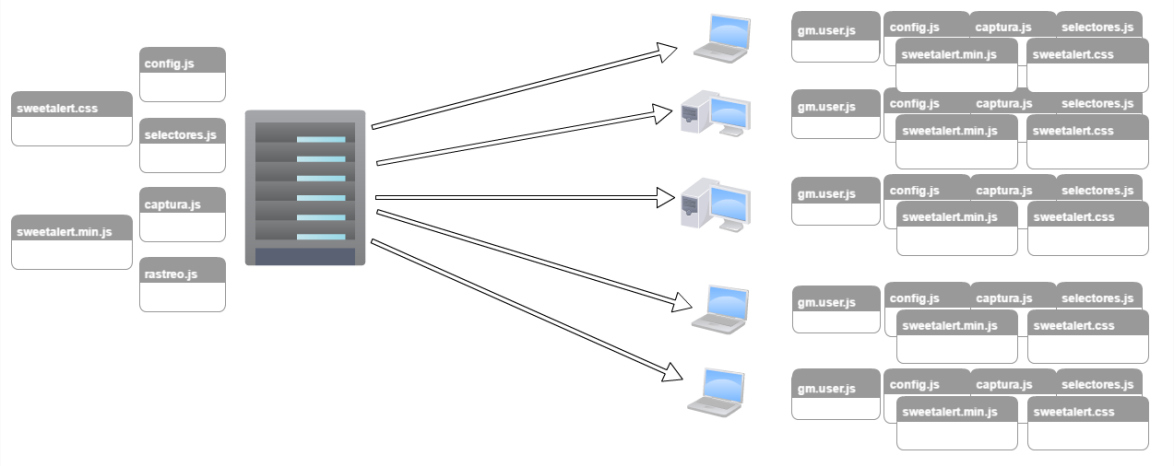

Figura 3-3: Mapa de red tras respuesta de servidor

El script de usuario lleva definido en su cabecera el dominio donde se ejecutará, el autor del propio script y una serie de parámetros adicionales como el nombre y la versión del script. Este script solamente se activará en el dominio especificado, y se encargará de realizar peticiones al servidor de los archivos que allí contiene, tras lo que pasarán a ejecutarse estos en el propio navegador del cliente y quedarán almacenados en caché.

El resto de los ficheros estarán almacenados en el servidor, y los archivos *config.js* y *selectores.js* podrán ser editados al gusto del administrador, de manera que se adecuen a las necesidades o preferencias que este tenga.

Figura 3-4: Flujo de carga

En la **Figura 3-4** se muestra el diagrama de flujo explicando los procesos que se realizan al dirigir la navegación hacia el dominio donde nuestro script de usuario actúa.

3.2.2 Diseño del Script de usuario para GreaseMonkey

A la hora de comenzar el desarrollo del script de usuario de GreaseMonkey se ha seguido el diseño marcado por la propia extensión para su correcto funcionamiento, esto implica diseñar correctamente la cabecera estableciendo los parámetros de forma adecuada.

```
// ==UserScript==
// @name          TFG
// @namespace     localhost
// @description   TFG
// @author        jggcela
// @homepage      localhost
// @match         http://www.uam.es/*
// @match         https://www.uam.es/*
// @match         http://www.productosoficiales.uam.es/*
// @run-at        document-idle
// @noframes
// @version       1.2
// ==/UserScript==
```

Figura 3-5: Cabecera script de usuario

En esta cabecera se incluirá en el argumento @match las páginas en las cuales el script de usuario se ejecutará, no teniendo efecto ninguno en el resto de páginas. En este ejemplo, el script se ejecutaría en las páginas del dominio de la UAM y en la tienda oficial.

Tras la cabecera comenzaría el script propiamente dicho, que, en este caso, realizará la carga de la biblioteca de JQuery si la página no la ha cargado previamente, y después se realizarán llamadas a los ficheros alojados en el servidor de localhost, construyendo previamente la URL donde se encuentran alojados.

3.2.3 Diseño de la Estructura de Datos almacenada en el navegador

Para almacenar las variables del usuario se hace uso de la variable *localStorage*.
LocalStorage es una variable que proporciona el navegador donde se pueden almacenar datos como cookies sin que caduquen.

En esta variable vamos a ir guardando los datos referidos a la navegación de cada cliente, de manera que se vayan guardando y consultando los mismos cuando sea necesario. Dentro de *localStorage* [3] almacenaremos varios objetos, estos son:

- *timestampnow*: En esta variable se almacenará el tiempo actual devuelto por el sistema. Esta variable nos servirá para eventos que usen la variable tiempo y también para prevenir posibles timeouts, explicado más adelante en la Sección 4.6.5.
- *rulesFlags*: Se almacenarán los identificadores de posición de aquellas reglas que ya hayan sido ejecutadas, con el objetivo de que no se vuelvan a ejecutar otra vez. Se explicará de manera más detallada en el apartado 4.6.2.
- *pestanas*: En esta variable se almacenarán todos los identificadores de las pestañas abiertas donde se está ejecutando el proyecto. Se verá con mayor profundidad en el apartado 4.6.1.
- *pestanaActual*: Se almacenará el identificador correspondiente a la pestaña actual, es decir aquella que tiene el abierta el navegador, al igual que la variable *pestanas*, se profundizará en esta variable en el apartado 4.6.1.
- *varNavCli*: en esta variable se almacenarán todas las variables del usuario, tanto las variables de página como las de navegador. Estarán almacenadas mediante clave-valor. Aquellas variables que no tengan aún ningún valor dado que no han recogido aún el correspondiente de la página no estarán incluidas, y en el momento que tomen un primer valor se añadirán. Sobre esta variable se realizan las comparaciones de las reglas definidas, tal y como se explica en el apartado 4.6.2.

Figura 3-6: Estructura variable *localStorage*

En la **Figura 3-6** se observan los valores que toman las diferentes variables dentro del objeto *localStorage* en una prueba realizada en el dominio de la Universidad Autónoma de Madrid.

4 Desarrollo

4.1 Introducción

Tras explicar el análisis y el diseño en la sección anterior, pasamos a explicar cómo se ha realizado el desarrollo del proyecto.

En este desarrollo se ha utilizado un ordenador personal con acceso a internet. En particular el software utilizado ha sido:

- Mozilla Firefox, navegador Web con la extensión de GreaseMonkey activada y el script de usuario cargado.

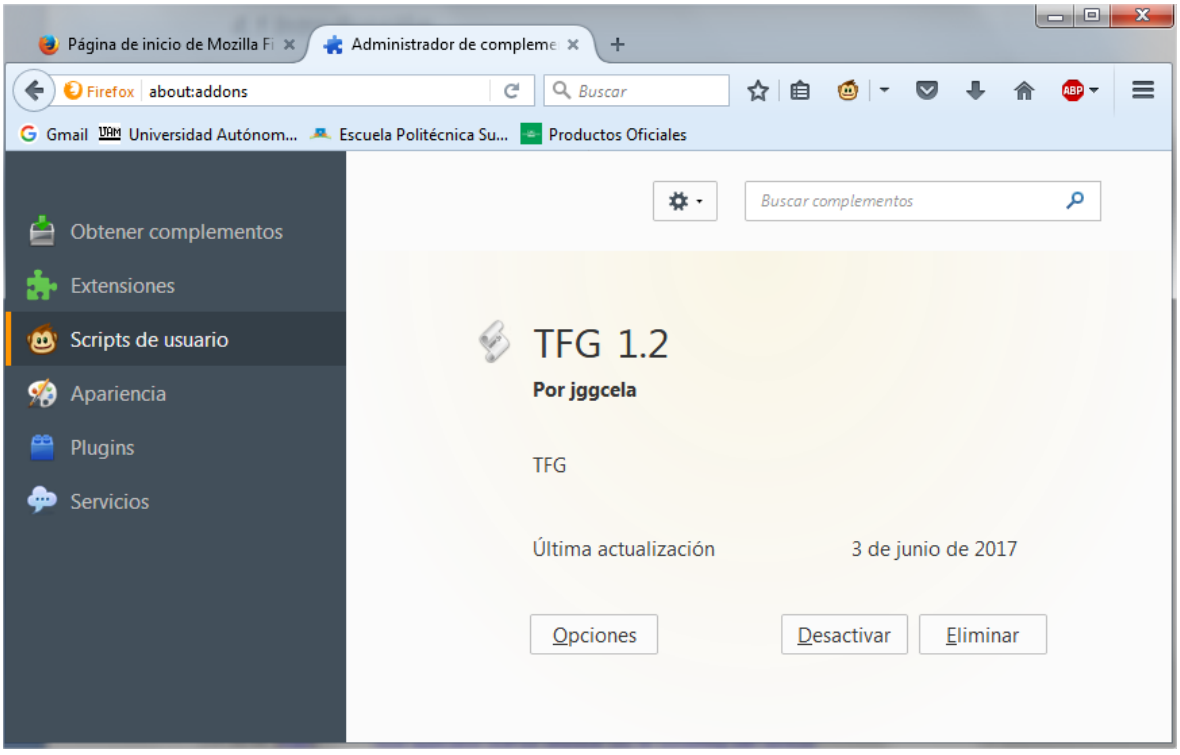

Figura 4-1: Panel control GreaseMonkey

- XAMPP, programa que nos ha servido para simular nuestro servidor, en este servidor se alojarán los scripts que se almacenarían del lado del servidor. A la hora de la realización de pruebas o demostraciones el servidor debe estar encendido y con los servicios de Apache y MySQL desplegados.

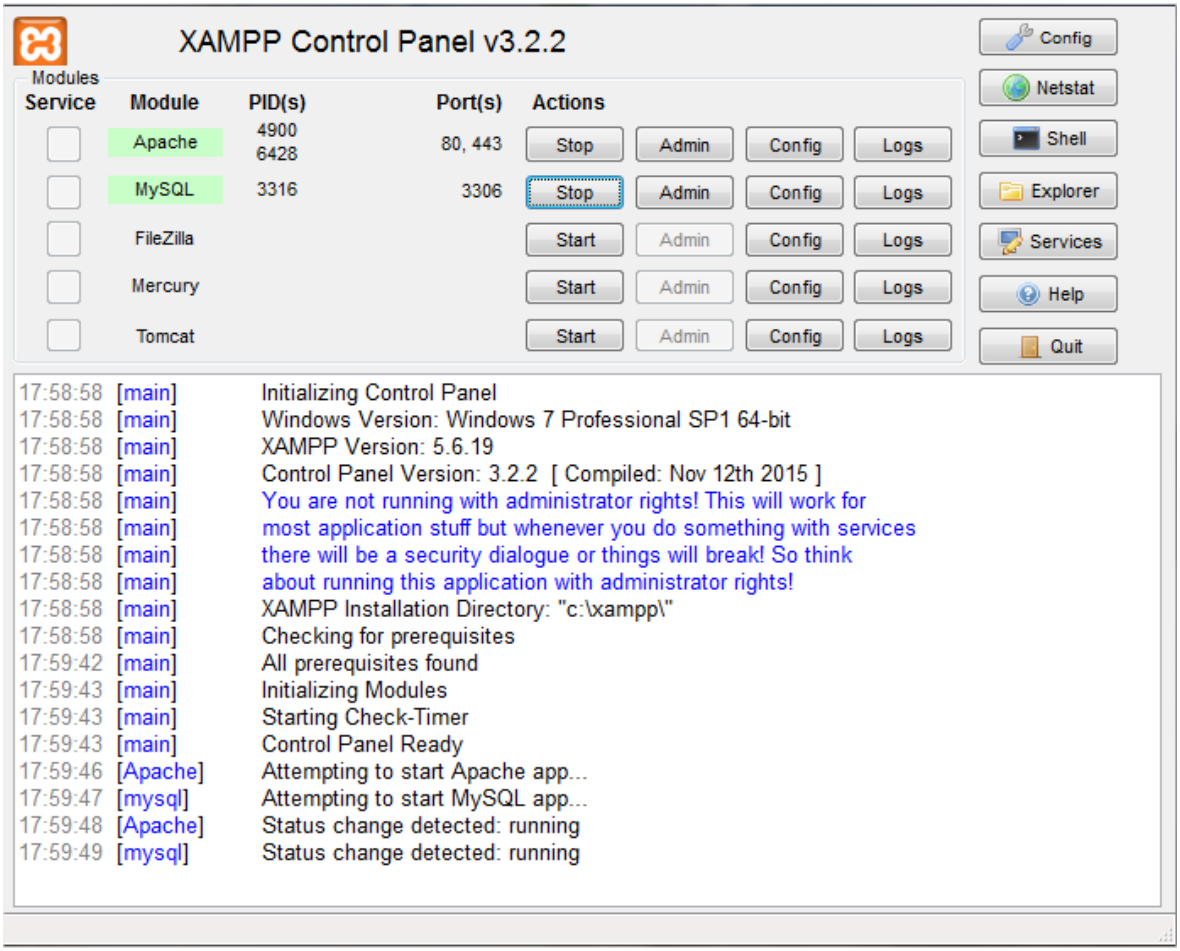

Figura 4-2: Panel de control XAMPP

- Sublime text, editor de código utilizado para el desarrollo del mismo.

Figura 4-3: Editor de textos SublimeText

- Consola del navegador Mozilla Firefox. Los avisos que se invoquen desde el código desarrollado se imprimirán por la consola de JavaScript del navegador, por ello y para realizar un seguimiento del código ejecutado se ha realizado un uso intensivo de esta consola, para así poder localizar errores en tiempo de ejecución.

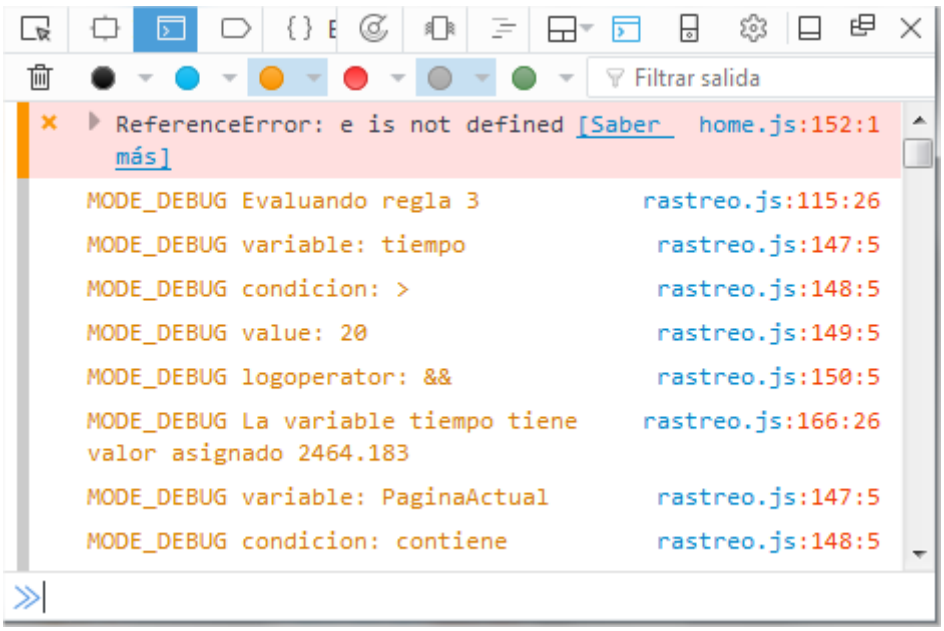

Figura 4-4: Consola de Mozilla Firefox

4.2 Desarrollo del Script de GreaseMonkey

En primer lugar, se realizó el desarrollo de este primer script de manera que estuviese funcionando correctamente en nuestro navegador, para ello se utilizaron scripts de usuario a modo de prueba para asegurarnos de que el funcionamiento era correcto.

A la hora de comenzar el desarrollo, tal y como se incluye en el apartado de diseño, se comenzó indicando la cabecera del script, tras lo cual se comenzó a realizar llamadas a los archivos necesarios en el proyecto. En primer lugar, se carga la librería de JQuery, ya que el resto de archivos hacen uso de ella, y sin esta carga el proyecto no tendría un funcionamiento correcto [8]. Tras esta carga construimos las URLs correspondientes a los archivos que nos devolverá el servidor.

```
var host = 'localhost';
var baseURL = 'https://' + host + '/Proyecto/wordpress/wp-content/uploads/InyeccionCodigo/';
var coreURL = baseURL + 'core/';
var demoURL = baseURL + 'demo/';

// Se usa en captura.js para cargar rastreo.js
window.gloCoreURL = coreURL;
```

Figura 4-5: Construcción URLs

En la URL correspondiente a demoURL obtendremos los archivos de datos. Estos dos archivos son:

- *selectores.js*: Archivo que contiene las variables a capturar en la página.
- *config.js*: Archivo que contiene un objeto con las reglas a ejecutar en la página.

Estos dos archivos serán explicados en detalle más adelante en las secciones 4.3 y 4.4.

En la URL denominada *coreURL* se realizará la carga de tres ficheros:

- *Sweetalert.css* Archivo CSS utilizado para dar estilo a los alerts que generará el código.
- *sweetalert.min.js*: Biblioteca propia de SweetAlert para JavaScript, utilizada para invocar alerts con el estilo propio de SweetAlert [12].
- *captura.js*: Archivo que realiza una búsqueda en el DOM comparando las variables obtenidas en el archivo *selectores.js*, al encontrar alguna ocurrencia guardará el valor en las cookies del navegador. También la carga del archivo rastreo.js, que se encargará de comparar las reglas obtenidas del archivo *config.js* con los valores que se hayan guardado en las cookies del navegador.

La carga de estos archivos se realiza en JavaScript mediante la función insertScript().

```
/*SweetMasterAlert es una biblioteca de estilos que se usa para los Alert*/
insertCSS(coreURL + 'sweetalert-master/dist/sweetalert.css');
insertScript(coreURL + 'sweetalert-master/dist/sweetalert.min.js');
insertScript(demoURL + 'selectores.js');
insertScript(demoURL +'config.js');
insertScript(coreURL + 'captura.js');
```

Figura 4-6: Carga de scripts del servidor

Finalmente, en el script se añadirán funciones recomendadas por GreaseMonkey para que su funcionamiento sea el correcto.

4.3 Desarrollo del Script selectores.js

El archivo *selectores.js* estará alojado en el servidor, contiene la información relativa a cada variable en un objeto JSON.

El archivo podrá ser modificado, añadiendo nuevas variables dentro del objeto JSON.
Cuando la información es correcta, el proyecto podrá capturar las variables y utilizarlas como variables en alguna de las reglas [2].

Se distinguen dos tipos de variables definidas dentro del objeto, variables de página y variables de navegador.

4.3.1 Variables de página

Como variables de página se han definido a aquellas variables que toman el valor de algún selector o tag insertado en el código. Estas pueden ser añadidas y modificadas por el usuario respetando la estructura con la que se deben definir en el fichero.
La estructura de la variable de página será la siguiente:

```
"Noticias":[
2,
"#wrapper > div.main_content > div:nth-child(3) > div.col_8_18 > div:nth-child(1) > h2",
3
],
```

Figura 4-7: Definición de un selector

En primer lugar, se indicará el nombre de la variable, nombre que no debe ser el mismo que el de otro selector que ya hubiese sido definido [1].

El primer campo del objeto puede tomar los valores 1 o 2:

- Si toma el valor 1 significará que la variable se trata de una variable INPUT, esto es que la variable se capturará desde un input de la página.
- Si toma el valor 2 significará que la variable estará definida como un párrafo en una página, su valor se capturará al finalizar la carga de la página.

El segundo campo corresponde al selector CSS de la variable que queramos capturar, este valor se obtiene desde el navegador inspeccionando el código HTML de la página.
El tercer campo define el tipo de dato de la variable. Están soportados dos tipos de datos, el tipo entero y el tipo String.

- Si toma el valor 3 el dato será de tipo String.
- Si toma el valor 4 el dato será de tipo entero.

4.3.2 Variables de navegador

Se utilizan en el proyecto una serie de variables que no se obtienen directamente de la página, si no que las obtiene el script del navegador, y las recoge de forma separada al resto, mediante código JavaScript que nos devuelve la propia página.

Estas variables de navegador son:

- Idioma: guardaremos en formato String el idioma por defecto configurado en nuestro navegador, lo obtenemos de la variable *navigator* usando *navigator.language.*
- Página Actual: guardado en formato String la página actual, obtenida de la propia página mediante *location.href.*
- Página Anterior: guardado en formato String, lo obtenemos mediante *document.referrer.*
- Título de la página: guardado en formato String, es obtenido a través de la variable *document.title.*
- Navegador: guardado en formato String, lo obtenemos de la variable navigator, haciendo referencia a *navigator.userAgent.*
- Sistema operativo: guardado en formato String lo obtendremos mediante *navigator.platform.*
- Móvil, según desde el dispositivo que accedamos tendrá valor True si se trata de un dispositivo móvil o false si no se tratase de un dispositivo de este tipo.

Respecto a su formato en el archivo, guarda algunas diferencias con respecto a las variables de página.

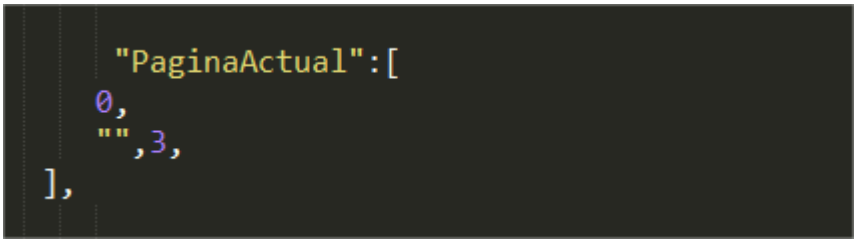

Figura 4-8: Definición de un selector de navegador

En primer lugar, indicaremos como hemos hecho anteriormente el nombre la variable. Dentro del objeto, en el primer campo indicaremos con un 0 que se trata de una variable del propio navegador, ya que indicándole un 1 o un 2 el proyecto interpretaría esta variable de forma errónea.
El segundo campo debemos dejarlo vacío, dado que estas variables no actualizan sus valores mediante búsquedas de selectores en la página, carece de sentido indicarles un selector, ya que carecen de él.
El último campo sí es común con las variables de página. Mediante un 3 indicamos que se trata de una variable de tipo String, mientras que si indicamos un 4 sería una variable de tipo entero. En el caso de las variables de navegador definidas en este proyecto todas corresponden a tipo String.

4.4 Desarrollo del Script config.js

El archivo *config.js* está alojado en la carpeta demo, y también podrá ser editado añadiendo nuevas reglas para añadir mayor dinamismo a la página utilizando las variables definidas en *selectores.js*.

Las reglas están definidas en formato JSON, contenido este objeto dentro de la variable lista reglas. El formato deberá ser respetado en caso de que se modifique alguna regla existente o se añada alguna nueva.

Cada regla tendrá como nombre *condlist,* y contendrá dos campos obligatoriamente. El primer campo será el de las condiciones, en este campo se definen las condiciones que se deben cumplir para que la regla se satisfaga. La estructura a seguir será la siguiente:

[{\"variable\":\"*NOMBRE_VARIABLE*\",\"condicion\":\"*TIPO_DE_COMPARACION*\",\"value\":\"*VALOR_A_COMPARAR*\",\"logoperator\":\"*OPERACION LOGICA*\"}],

En *NOMBRE_VARIABLE* se completará con el nombre de la variable definida en *selectores.js*. En *TIPO_DE_COMPARACION* deberemos distinguir si la variable es de tipo entero o String. Si es de tipo String las posibilidades serán “contiene” o “no contiene”, mientras que si es de tipo entero las posibilidades serán “>” “<” “>=” “<=” o “==”.
En *VALOR_A_COMPARAR* se debe añadir con el cual debemos comparar el valor de la variable que hemos añadido anteriormente.

En *OPERACIÓN LOGICA* se añadirá un operador lógico, ya sea “&&” o “||”, con cualquiera de estos dos operadores deberemos añadir una nueva condición con el formato que se ha seguido hasta ahora. En caso de que no se añada una nueva condición se debe colocar en la parte del operador lógico un “;”.

Con esto completamos el primer campo, quedando definidas las condiciones de una regla, en el segundo campo se define la acción a ejecutar en caso de que la anterior lista de condiciones se cumpla.
Separando ambos por comas, en el segundo campo deberemos indicar el tipo de acción que queremos que se ejecute, puede ser de dos tipos:

- Alert: Se generará un alert de JavaScript usando los estilos de SweetAlert, indicándole como parámetro el mensaje que deseamos mostrar.
  \"acción\":{\"tipo\":\"alert\",\"parametros\":[\"El plazo de inscripción para nuevos alumnos empieza el próximo viernes 9 de Junio\"]}}
- Popup: Se generará un pop up en la ventana del navegador, las dimensiones del mismo, así como la URL que se debe mostrar en el popup. Primero se indicará la URL para después indicar el ancho y el alto que deberá tener el popup que deseemos mostrar.
  \"accion\":{\"tipo\":\"popup\",\"parametros\":[\"https://secretaria-virtual.uam.es/Navegacion/InicioAlumno_mat.html\",\"1000\",\"550\"]}}

Añadiendo reglas siguiendo estas pautas se desarrolló el script de reglas, y este mismo formato es el que se ha seguido para su posterior lectura.

4.5 Desarrollo del Script captura.js

El script *captura.js* se encarga de capturar los selectores en la página que actualmente está mostrando el navegador, para ello utiliza la biblioteca de JQuery para acceder a los selectores comparándolos con los definidos en *config.js*.

Según se inicia la ejecución del código se realiza la carga del fichero *rastreo.js*, esto se realiza por que el archivo rastreo.js debe ejecutarse después del archivo *captura.js*.
El desarrollo de este script se centró en recoger datos de los selectores HTML según el modelo DOM, y compararlos con los selectores definidos en *selectores.js*.

Si se encuentra alguna coincidencia, se guardará el valor del selector en las cookies del navegador, para que ese valor sea posteriormente rescatado por rastreo.js y evaluado por las reglas con los valores guardados en las cookies. Se ejecutará la función principal al terminar la carga de la página, haciendo uso de la biblioteca de JQuery [4].

```
$(document).ready(function() {
    if (localStorage.timestampnow && ((Date.now() - window.localStorage.timestampnow) > timeOut)) {
        for (varName in varNavCli)
            delete localStorage[varNavCli[varName]];
        if(navigat['Debug']) console.log("Variables de sesion invalidadas");
    }
//llamada de nuevo a capturar
    updatetimer();
    if(navigat['Debug']) console.log("Timer actualizado");
    variableDinamica = ' ';
    captureStaticVariables();
    waitForElement();

    localStorage.timestampnow = Date.now();
});
```

Figura 4-9: Proceso de *captura.js*

La variable *timeout* controla que la ejecución se realice de manera regular. En cuanto el documento HTML esté cargado comprobará si el timeout se ha producido. En caso de que se produzca se eliminará todo el registro que se haya dejado en el *localStorage* del navegador. Tras esta comprobación se llamará a las funciones que se encargan de medir tiempo, capturar variables del navegador y, por último, capturar los inputs.

La función **updatetimer()** se encargará de actualizar la variable de tiempo, y nos servirá para detectar si se ha producido algún timeout, tras lo cual deberemos invalidar el registro de variables.

La función *captureStaticVariables()* se encargará de recoger los datos de las variables que nos facilita el navegador. Estas son definidas en *config.js* pero no toman valores desde la página, si no que obtienen su valor de las variables que gestiona el navegador.

En la función *waitForElement()* buscamos coincidencias entre los electores definidos en el fichero *config.js* y la página actual, en caso de haber coincidencias se guardará en el *localStorage* una nueva variable con el nombre definido en *config.js* y con el valor que hayamos obtenido de la página. Posteriormente este valor se recuperará para realizar la comprobación de las reglas.

4.6 Desarrollo del Script rastreo.js

La mayor parte de la funcionalidad de este proyecto está ubicada en este script. En el mismo se realizan varias operaciones de manera síncrona en un intervalo de dos segundos. Este intervalo se ha tomado debido a que nos interesa que el código se ejecute constantemente, pero se quiere evitar que dos instancias de código se estén ejecutando de manera simultánea.

La manipulación de variables por parte de dos instancias provoca fallos como, por ejemplo, la no ejecución de alguna acción cuando la lista de condiciones se ha cumplido satisfactoriamente. Por tanto, se han tomado intervalos de dos segundos, dado que se producirá la ejecución del script de manera constante y con esa diferencia de tiempo no habrá solapamiento entre instancias.

Explicaremos de forma detallada cada operación realizada por este script en los subapartados siguientes.

4.6.1 Gestión de pestañas.

La idea de realizar una gestión de las pestañas surgió en el momento en que con dos pestañas abiertas se producían fallos a la hora de capturar variables, dado que al estar almacenadas en una variable del navegador estaban constantemente sobrescribiendo valores y produciendo errores [2].

Para solucionar estos conflictos se ha utilizado la política de que únicamente la pestaña actual estará en ejecución, dejando el resto de pestañas en “standby” hasta que alguna de ellas sea la pestaña actual y esta tome el control. Para realizar esto hemos almacenado en el *localStorage* dos objetos:

- *pestanas*: en esta variable almacenaremos todos los identificadores de las pestañas que estén ejecutando nuestro código, al iniciarse una pestaña nueva, se le asignará un nuevo número aleatorio, que se almacenará en esta variable como un nuevo elemento de un array.
- *pestanaActual*: en esta variable se guardará el identificador de la pestaña actual, si en el momento de ejecución, el identificador de la pestaña no coincide con el contenido en esta variable finalizarán el resto de comprobaciones, forzando así a que sea una única pestaña la que realice las comprobaciones.

A su vez hemos tenido que guardar cada valor de pestaña en una variable que únicamente afecte a la pestaña, dado que si esa variable es compartida por todas no podríamos tener registro de todas las pestañas que existen ni hacer un cambio de pestaña actual de manera correcta. Para esto, se ha utilizado la variable de pestaña window.name, de forma que al crearse una nueva pestaña el número aleatorio generado se guardará en esa variable antes de ser añadido al array de pestañas del *localStorage*.

Para realizar un cambio en la pestaña actual se define una función que estará a la escucha en todas las pestañas. Esta, al recibir un evento de tipo *focus(),* se activará, cambiando la variable *pestanaActual* por el valor de su variable window.name, pasando a tomar el control y realizar las comprobaciones requeridas.

Para garantizar la pérdida de control de una pestaña cuando esta deja de ser la pestaña principal se ha implementado un función paralela, que se activará al recibir un evento de tipo *.blur(),* borrando su identificador de la variable *pestanaActual*.

4.6.2 Procesamiento de reglas

La parte del procesamiento de las reglas es la parte central del proyecto. Esta se realiza después de que la pestaña activa compruebe que su identificador es el correspondiente al almacenado en *pestanaActual*. Tras esto, realiza una carga del objeto *listaReglas* almacenado en *config.js*, convirtiéndolo a una variable de JavaScript que hemos denominado rules. La variable rules estará definida como un array de reglas, de manera que cada regla tendrá una posición dentro del array. Esta posición nos ayudará a llevar el control de las reglas que ya se han ejecutado.

Una vez cargadas las reglas se cargan los valores de las variables, que han sido almacenadas en el *localStorage* dentro del objeto *varNavCli* en el script *captura.js*, en una nueva variable para realizar las comprobaciones necesarias. Una vez obtenidas estas dos variables se irá llamando a evaluar cada regla de una en una. Si el identificador de una regla aparece en la variable ruleFlags se ignorará, pasando a procesar una regla que aún no haya sido ejecutada. Esto se realiza para evitar que una regla sencilla esté ejecutándose constantemente, dificultando la navegación del usuario y la ejecución de otras reglas. Si la regla no ha sido ejecutada aún se evaluará la lista de condiciones que la define, esperando que se devuelva un valor booleano en caso de que la lista de condiciones se satisfaga o no.

En caso de que la evaluación de la regla sea “true” se realizará la llamada a la ejecución de la acción que lleva asociada, y se almacenará *rulesFlags* la posición de la regla dentro del array rules que ya se ha cumplido, con el objetivo de que no se vuelva a ejecutar más.

4.6.3 Evaluación de las reglas

Durante el procesamiento de las reglas se realizará una evaluación de la lista de condiciones de cada regla a procesar. Para ello se hará uso de la lista de reglas y de los valores de las variables que se han almacenado en *localStorage*. Se evaluará cada condición, atendiendo al tipo de dato de la variable con el que se evalúa la misma.

Así pues, para los casos de String se contempla que la variable contenga o no contenga una determinada cadena de caracteres, tras lo cual se enviará un valor true o false en esa condición, pasando a evaluar la siguiente regla de la lista. En caso de los enteros se realizará una expresión regular y se evaluará, devolviendo como en el caso de los Strings un valor booleano true si se cumple la condición o falso si no.

Tras evaluar todas las reglas de la lista se devolverá un valor booleano al método de procesamiento de reglas indicando si se satisface o no la lista de reglas.

4.6.4 Ejecución de acciones asociadas a reglas

Si durante la evaluación de una regla su lista de condiciones se satisface, el procesador de reglas llamará a la ejecución de la acción asociada. Se realizarán acciones diferentes según sea la acción definida para esa regla.

En caso de que la acción asociada sea la ejecución de un alert, se insertará el texto definido en la acción en un alert específico de la biblioteca *SweetAlert*. Para ello se hace uso de una función que proporciona dicha biblioteca [12].

Si la acción es de tipo popup es necesario realizar una serie de comprobaciones previas a la ejecución. En primer lugar, se realizará una comprobación sobre la URL que contendrá el popup, verificando que su formato es acorde al de una URL. Tras esta comprobación se realizará el inicio del popup, con las dimensiones que hayamos configurado en la acción de la regla.

Con la ejecución del popup se crea un botón en la esquina superior derecha del mismo. Este botón lleva un *listener* con la función *cerrarPopup*, que al recibir un click sobre el mismo ejecutará la función cerrando el popup

4.6.5 Actualización de la variable tiempo

La variable tiempo no está definida como variable de página ni como variable de navegador, aunque es a su vez guardada por el programa. Obtenemos su valor del *timestamp* y podemos hacer uso de ella para la realización de las reglas.

Con la actualización de esta variable finalizaría el código del script rastreo.js, aunque cada dos segundos realizará una nueva ejecución de este script.

5 Integración, pruebas y resultados

La última fase del proyecto y a su vez la más extensa ha sido la fase de pruebas. Dado que el proyecto ha sido desarrollado en JavaScript la fase de pruebas se ha centrado en su mayoría en realizar pruebas de validación.

Se realizó una versión de demo sobre dos dominios con el fin de realizar pruebas de validación una vez el proyecto estuvo en una primera versión funcional.

5.1 Pruebas sobre el dominio de la UAM

En el primer dominio donde se llevaron a cabo pruebas fue en el dominio de la Universidad Autónoma de Madrid. En este dominio se definieron una serie de reglas simulando una campaña de matriculación.

Se definieron las siguientes reglas:

Mostrar alert informativo indicando las fechas de matriculación en la UAM al permanecer 5 segundos en cualquier página del dominio.

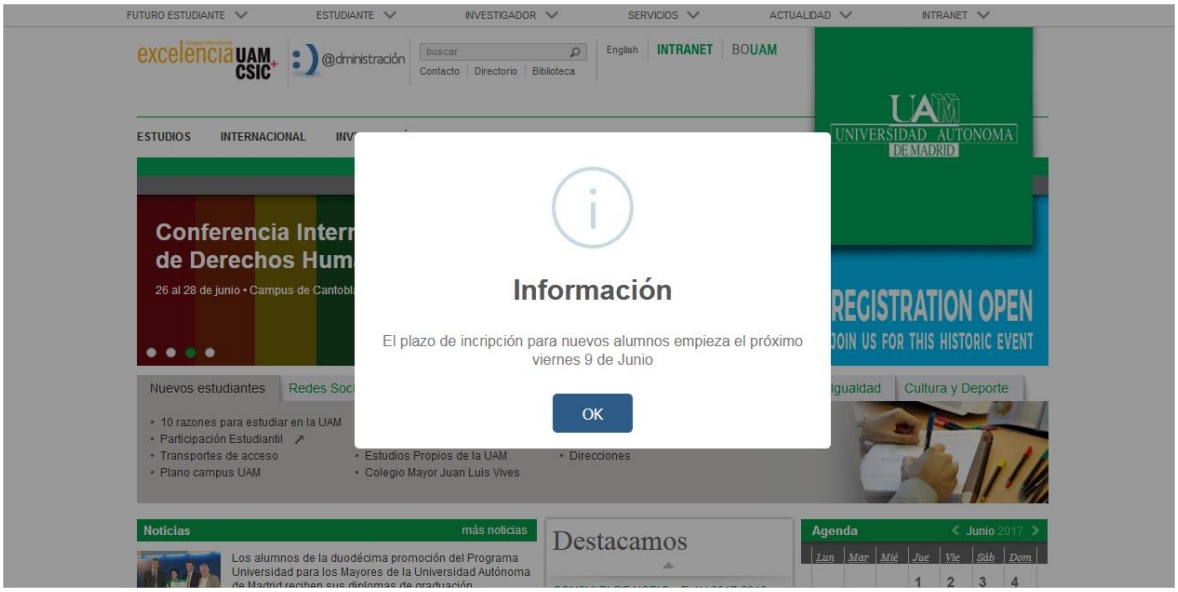

Figura 5-1: Ejecución de la acción 1 (alert)

Si el usuario cambia el idioma a inglés se le mostrará un alert dándole la bienvenida al portal de la UAM en inglés.

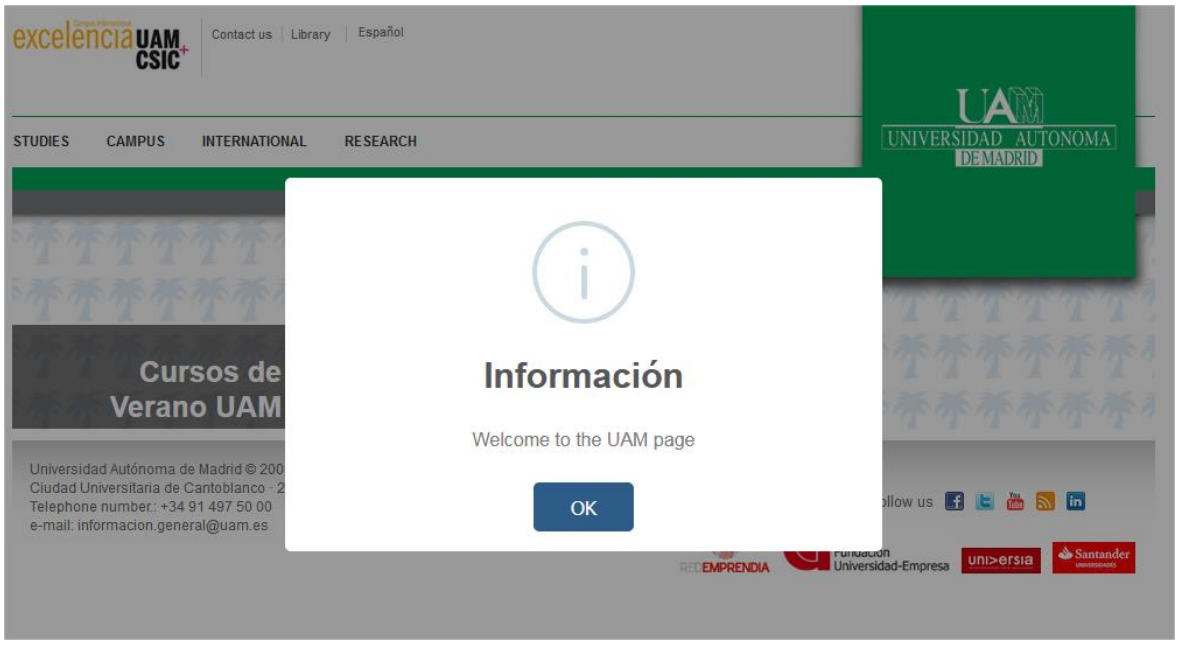

Figura 5-2: Ejecución de la acción 2 (alert)

Si el usuario accede a la página de la EPS y accede a la pestaña presentación, al permanecer 10 segundos se mostrará un popup con una entrevista al rector de la UAM.

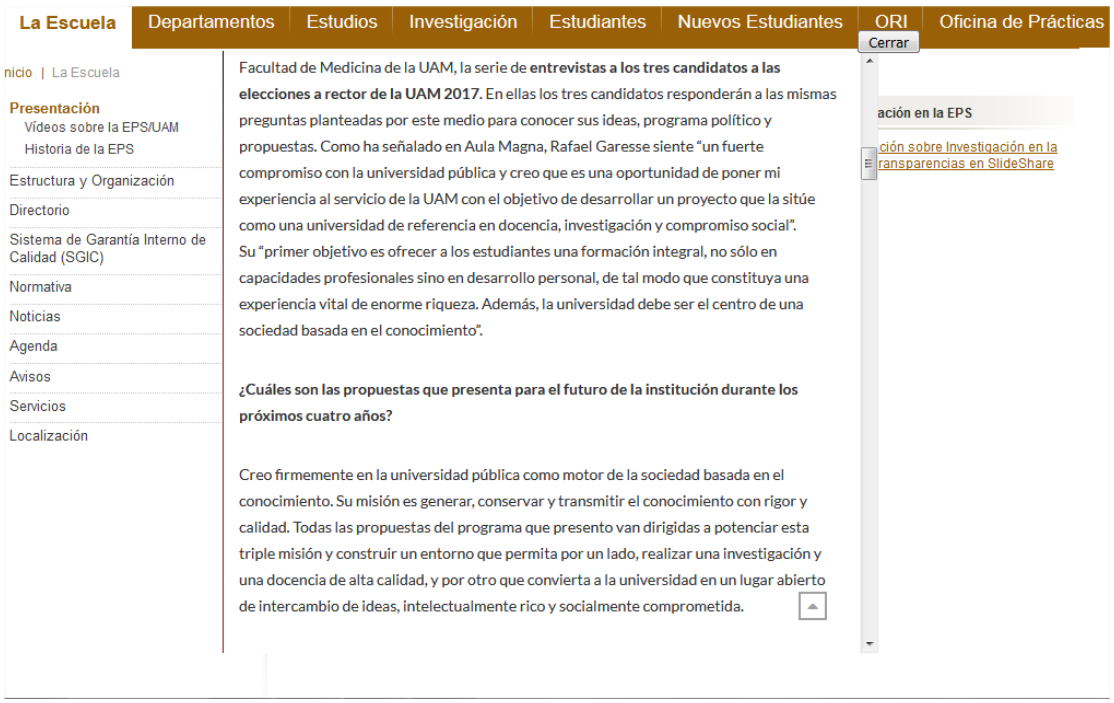

Figura 5-3: Ejecución de la acción 3 (popup)

Si el usuario dentro de la web de la EPS accede a la información de nuevos alumnos se le mostrará un popup con SIGMA UAM para la realización de la automatriculación.

Figura 5-4: Ejecución de la acción 4 (popup automatriculación sigma)

Si el usuario accede dentro de la web de la EPS a Asuntos Académico Administrativos se le mostrará un nuevo popup con SIGMA UAM para que el alumno pueda acceder directamente desde allí.

Figura 5-5: Ejecución de la acción 5 (popup enlace a sigma)

A raíz de estos resultados podemos observar el correcto funcionamiento de las reglas creadas, la correcta definición de los selectores utilizados, así como el correcto funcionamiento del proyecto en todo el dominio de la UAM.

5.2 Pruebas sobre el dominio PCcomponentes

Otro dominio donde se ha realizado pruebas es la tienda online de PCcomponentes. En esta tienda se definieron tres reglas y se ha comprobado su funcionamiento. Fue necesario configurar un fichero de reglas y otro de condiciones diferentes a los usados anteriormente en las pruebas sobre el dominio de la UAM.

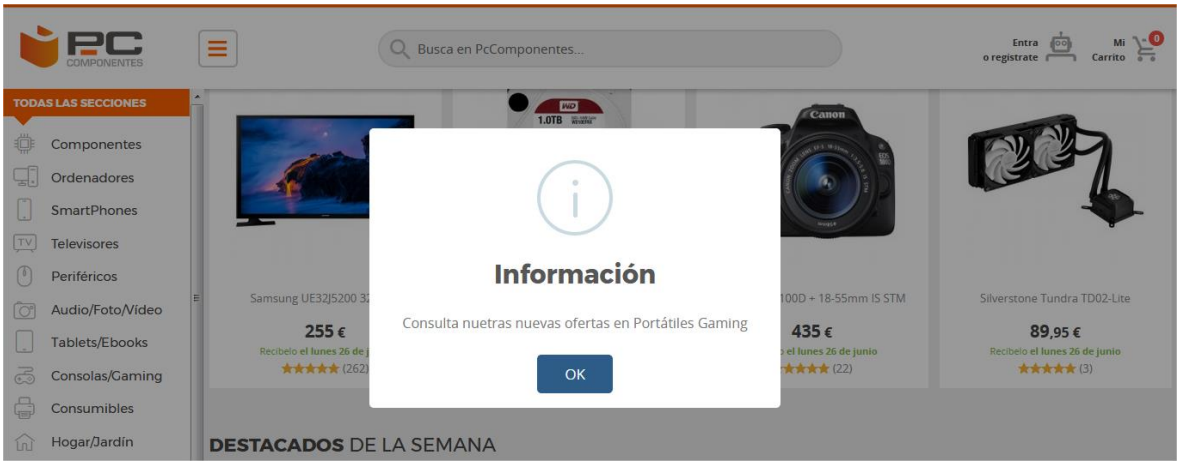

Figura 5-6: Ejecución de la acción 1 (alert ofertas)

La primera regla definida se ejecutará al entrar en el sitio de PCcomponentes, anunciándonos que existen ofertas en portátiles gaming.

Figura 5-7: Ejecución de la acción 2 (alert carrito)

Para esta segunda regla se define previamente la variable *NumProducts* en *selectores.js*. Esta variable recogerá el valor del número de productos que actualmente tenemos en el carrito, tal y como se puede observar en la esquina superior derecha de la figura 5-7.

La regla muestra un mensaje informándonos de que actualmente tenemos dos productos en el carrito, y que añadiendo un tercero podríamos beneficiarnos de descuentos. Esta regla se ejecuta cuando los productos de nuestro carrito sean dos.

Figura 5-8: Ejecución de la acción 3(alert precio)

La tercera regla diseñada sobre este dominio también nos muestra un mensaje informativo. Para su realización obtenemos una nueva variable y la definimos en *selectores.js*. Esta variable será el total del producto, la cual se define como entero, y con esto podremos definir operaciones más complejas que si fuese tomado como String.
Para este caso particular, cuando esta variable tenga un valor comprendido entre 800 y 1000 nos mostrará un mensaje animándonos a superar los 1000€ de compra para obtener un descuento del 20%.

Las pruebas realizadas sobre este dominio han dado un resultado satisfactorio. Así como las pruebas en el dominio de la Universidad Autónoma de Madrid se centraban más en variables de tipo String, en este dominio hemos querido centrar las pruebas sobre las variables de tipo entero, su correcto parseo y su utilización en reglas más complejas que las definidas anteriormente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

La realización de este proyecto me ha parecido una experiencia enriquecedora. He adquirido infinidad de conocimientos sobre los lenguajes usados en la web y las infinitas posibilidades que aportan.

Durante este proceso se ha aprendido no solo de las tecnologías utilizadas de manera directa en el proyecto, si no de tecnologías similares, como otros frameworks de JavaScript u otras extensiones para scripts de usuario en navegadores, con lo que he adquirido una visión global de ambas.

Una vez decididas las tecnologías que se consideraron óptimas para llevar el proyecto a buen término, surgieron multitud de problemas, principalmente debido a la dificultad de depurar código en JavaScript. A pesar de estos problemas se consiguió solventarlos y esto ayudó a ampliar conocimientos, adquiridos con el fin de solventar los mismos.

Debido a los resultados obtenidos se continuará el desarrollo del proyecto, con el objetivo de crear un producto más completo y sencillo de utilizar para un usuario sin conocimientos técnicos en las tecnologías de la World Wide Web. En la siguiente sección se explican las pautas y los problemas en los que se centrará el trabajo futuro y los objetivos de la implementación de estas mejoras.

6.2 Trabajo futuro

El proyecto es más ambicioso que el expuesto en este trabajo, en el futuro se pretende continuar en desarrollo orientándonos en varios frentes.

En primer lugar, cambiar el entorno del proyecto, pasando de estar en localhost como ha estado alojado hasta ahora, a estar en un servidor de internet, el mismo donde esté alojado el dominio donde tendría efecto el proyecto.

A la hora de probar con mayor precisión el proyecto se pretende prescindir del script de usuario desarrollado hasta este punto, siendo la propia web la que cargue los ficheros de su mismo servidor, ya sea por peticiones o mediante inserción de los scripts por JavaScript.

En ese mismo servidor se pretende realizar una web de administración donde se podrán definir las reglas y sus acciones derivadas mediante un formulario dinámico, que escribirá en el fichero *config.js* la configuración definida en el formulario en cuestión.

Otra página que se alojaría en el servidor sería una página de definición de los selectores, funcionando de igual manera que funcionaría la página de definición de reglas. Un formulario dinámico donde se completarían de forma más clara e intuitiva a como se hace hasta ahora la definición de los selectores.

Referencias

[1] Troy Miles, “JQuery Essentials”, Packt Publishing, 2016.

[2] Cesar Otero, Rob Larsen, “Professional JQuery”, John Willey & Sons, 2012.

[3] Luc Van Lancker, “JQuery, El framework JavaScript de la Web 2.0”, Eni Ediciones, 2014.

[4] 5 Differents ways to declare functions in JQuery https://www.sitepoint.com/5-ways-declare-functions-jquery/ [Junio 2017]

[5] Estadísticas de uso de JQuery http://www.anieto2k.com/2010/10/26/estadisticas-de-uso-de-jquery-en-la-red/ [Junio 2017]

[6] Grease Monkey https://en.wikipedia.org/wiki/Greasemonkey#Userscripts.org [Mayo 2017]

[7] JQuery https://es.wikipedia.org/wiki/JQuery#Caracter.C3.ADsticas

[8] JQuery Installation, Overview, and Getting Started https://www.slideshare.net/martyhall/javascript-and-jquery-programming-tutorial-jquery-installation-overview-and-getting-started [Mayo 2017]

[9] JQuery, write less, do more https://api.jquery.com/html/ [Mayo 2017]

[10] Safari 5 Extensions – Greasemonkey Clone Ninjakit, Youtube Downloadr & More https://sites.google.com/site/thelegendofpaz/new/safari5extensions [Junio 2017]

[11] Scriptish vs Greasemonkey http://forums.mozillazine.org/viewtopic.php?f=7&t=2455787 [Mayo 2017]

[12] SweetAlert, A beautiful replacements for JavaScript “Alert” http://t4t5.github.io/sweetalert/ [Abril 2017]

[13] ViolentMonkey improves Greasemonkey Script support in Opera http://www.instantfundas.com/2012/12/violent-monkey-improves-greasemonkey.html

[14] What is the future of jQuery for 2017 and later? https://www.quora.com/What-is-the-future-of-jQuery-for-2017-and-later [Mayo 2017]

[15] ¿Qué es GreaseMonkey? http://www.taringa.net/posts/ebooks-tutoriales/1278811/Que-es-GreaseMonkey-Scripts.html

Glosario

Alert	Mensaje emergente de JavaScript.
Cookies	Información enviada por un sitio web almacenada en el navegador.
GreaseMonkey	Extensión para inyección de código del navegador Mozilla Firefox.
HTML	HyperText Markup Language.
JavaScript	Lenguaje de programación interpretado.
JSON	JavaScript Object Notation.
JQuery	Framework de JavaScript.
Listener	Función que se ejecuta cuando sucede un determinado evento.
MVC	Modelo-Vista-Controlador.
Popup	Ventana emergente mostrada en un navegador web.
Selector/tag	Nexo de unión entre hoja de estilos y los documentos a los que se aplica dicha hoja.
Servidor Web	Aplicación en ejecución capaz de atender a peticiones del cliente y devolverle una respuesta en concordancia.
SPA	Single Page Application.
String	Variable formada por una cadena de caracteres
Timeout	Parámetro que indica el tiempo máximo de espera antes de abortar una tarea o acción.
URL	Uniform Resource Locator.