UNIVERSIDAD POLITÉCNICA DE CARTAGENA

TRABAJO DE FIN DE GRADO

Julio 2018

BAST ED: Seguridad Web mediante macro análisis de JavaScripts

*Un proyecto parte del **Reto Big Data y Ciberseguridad** propuesto y tutelado por la **Fundación Telefónica** en colaboración con la **ETSIT/UPTC***

Autor: Francisco Gálvez Griñán

Directora: María Dolores Cano Baños

Co-director (Telefónica): Marcos Arjona Fernández

Agradecimientos

Después del largo viaje que ha supuesto la realización del presente Trabajo de Fin de Grado, este llega a su final y me gustaría agradecer a todas las personas que lo han hecho posible. Ha sido todo un periodo de aprendizaje a nivel profesional y personal, sintiéndome profundamente realizado por los resultados alcanzados, que no hubieran sido posibles sin todo el apoyo recibido.

Primeramente, me gustaría agradecer a mis incansables compañeros Juan Rafael Cuenca, Francisco Javier González González, Moisés Francisco Gea Bartolomé y Juan José Espín Fernandez, que han hecho de mi estancia en la Universidad una experiencia inolvidable. Y en particular, a mi tutora María Dolores Cano Baños, por su labor como docente realizada durante la supervisión de este trabajo.

Así mismo, agradecer tanto a Telefónica como a la Universidad Politécnica de Cartagena la oportunidad que me han brindado con el Reto Big Data y Ciberseguridad, haciendo mención especial a mi tutor en Telefónica, Marcos Arjona Fernandez.

Y también quisiera agradecer a mis padres, por sus sabios consejos y compresión más allá de todo el esfuerzo económico que han realizado. Una especial mención a mi hermana, por ser una excelente guía durante la realización del proyecto y una imagen en la que fijarse. Y finalmente, a mi novia y mis amigos que han estado ahí en los momentos difíciles.

Muchas gracias a todos.

Índice general

Índice de ilustraciones

1 INTRODUCCIÓN

La realización de este proyecto en particular posee como objetivo fundamental la implementación de un sistema que monitorice los archivos que transitan por la Web con el objetivo de poder monitorearlos para hacerla más segura.

1.1 Motivación

Hoy en día vivimos en un entorno donde el uso de la tecnología ha impregnado todos los ámbitos de la sociedad. Es inevitable afirmar que tal y como se desarrolla hoy la sociedad, no podríamos vivir ya sin el rápido, eficaz y masivo tránsito de información que ofrece Internet. Desde muchos puntos de vista, la revolución tecnológica que comenzó a finales del siglo pasado y vivimos en la actualidad nos está brindando una inmensa cantidad de oportunidades.

Internet y la Web han sido, son y serán dos de los pilares fundamentales en la informática y por supuesto en la sociedad. El uso de estas tecnologías nos ha permitido romper con las barreras físicas y económicas que han supuesto para el ser humano a lo largo de nuestra historia, permitiendo así una multitud de posibilidades al alcance de un solo click.

Por su parte, las aplicaciones web proporcionan multitud de posibilidades como, por ejemplo, la creación de páginas web que se ajustan al perfil del usuario bien actúe como particular o empresa. Además, las aplicaciones web permiten interactuar otros sistemas informáticos, tales como, gestión de inventario, publicidad o contabilidad mediante el propio portal web[1].“Las aplicaciones web se encuadran dentro de las arquitecturas cliente/servidor: un ordenador solicita servicios (el cliente) y otro está a la espera de recibir solicitudes y las responde (el servidor)” [S. L. Mora, 2013].

Que el uso de las nuevas tecnologías nos está sirviendo para avanzar es un hecho, pero no hay que olvidarse que las nuevas tecnologías también pueden ser usadas en contra del propio desarrollo constructivo de la sociedad. Hoy en día a pesar de que con un clic se puede hacer una trasferencia bancaria, comprar *online* o adquirir un billete de avión, también es importante estar seguros de que nuestros datos personales y nuestros documentos no son suplantados y modificados en la red.

Este escenario nos plantea un gran reto, que es el de conseguir que todos los archivos que son enviados a través de Internet, en el camino que recorren desde la petición del cliente hasta la respuesta del servidor, seamos capaces de monitorizarlos para estar seguros de que no han sido modificados por el camino.

[1] Vía https://gplsi.dlsi.ua.es/~slujan/materiales/pi-cliente2-muestra.pdf

En realidad, resulta imposible quedarse al margen de las nuevas tecnologías, lo que implica muchas ventajas, pero, no hay que olvidar que también existen desventajas. Estas desventajas suponen un peligro que hay que gestionar, para ello hacemos uso de la Ciberseguridad.

"La Ciberseguridad es el área de la informática que se enfoca en la protección de la infraestructura computacional y todo lo relacionado con esta y, especialmente, la información contenida o circulante. Para ello existen una serie de estándares, protocolos, métodos, reglas, herramientas y leyes concebidas para minimizar los posibles riesgos a la infraestructura o a la información"[Introducción a la Ciberseguridad, 2016].

Contar con sistemas sólidos en cuanto a ciberseguridad se refiere es vital para todos los consumidores y en su gestión deben implicarse el conjunto de la sociedad.[2]

Este contexto actual es el que ha llevado a la *Fundación Telefónica* junto con varias universidades españolas, entre ellas, la Universidad Politécnica de Cartagena (UPCT), a realizar el *Reto Big Data y Ciberseguridad* y, de forma concreta, al desarrollo de este proyecto: *BAST ED: Seguridad Web mediante macro análisis de JavaScripts*.

La justificación de llevar a cabo este proyecto viene marcada por la necesidad de enfrentar los riesgos de subir archivos a Internet desde cualquier portal web. La incógnita que se intentar despejar con la realización de este proyecto es la que existe actualmente respecto de la seguridad en la gestión de archivos en Internet, que no es otra que ser capaces de garantizar al usuario que efectivamente sus archivos no han sido modificados en el proceso.

En consecuencia, el software que se propone en este proyecto comprueba que el archivo que el usuario sube a la Web coincide con el que recibe el servidor y que no ha sido dañado por el camino por ningún *malware* (software maligno cuya función es dañar un sistema o causar un mal funcionamiento de este), permitiendo afirmar que los archivos no han sido modificados y la información del usuario no corre ningún peligro.

1.2 Objetivos

Este proyecto tiene como objetivo desarrollar una herramienta que sea útil, fácil para el usuario y que no requiera de gran conocimiento de programación para hacer un uso de ella. El principal objetivo viene marcado por la necesidad de controlar que nuestros archivos viajen por la red sin ser modificados por el camino. Por lo tanto, podríamos definir nuestro objetivo principal como: **desarrollar un software que garantice el envío seguro de archivos, concretamente JavaScripts, en la Web.**

El trabajo realizado en este proyecto es la semilla de algo que podría consolidarse como una gran herramienta de control del tráfico web. Si bien es cierto que el proyecto solo contempla la supervisión de archivos **JavaScript**, una implementación posterior debería ofrecer la

[2] Vía https://home.kpmg.com/es/es/home/servicios/advisory/risk-consulting/it-advisory/ciberseguridad.html

posibilidad de controlar todos los archivos **JavaScript** contenidos en una página web sin necesidad de hacerlo manualmente, simplemente gracias a una extensión para el navegador.

Del objetivo principal, el proyecto abarca otros dos objetivos secundarios como son: por un lado, ofrecer la posibilidad de controlar las versiones de los archivos **JavaScript** que se cargan; y por otra, ofrecer la posibilidad de comprobar las diferencias que existen entre las diferentes versiones de un mismo archivo **JavaScript**.

El control de versiones es fundamental en el entorno de la Ciberseguridad ya que un archivo puede ser modificado voluntariamente por el usuario y no por ello haber sido *hackeado*. Es muy común que cualquier archivo sea modificado y se extraiga una nueva versión de él. Pues esto mismo pasa con los archivos **JavaScript**, de los que se pueden crear versiones diferentes bajo un mismo nombre, pero siendo a su vez una versión nueva. Se trata de un objetivo secundario ya que en definitiva lo que se busca con el control de versiones es conseguir lo que hemos marcado como objetivo principal, pero introduciendo la cláusula de que un archivo modificado por el usuario no es un archivo *hackeado*, sino una versión diferente de un mismo archivo.

El segundo objetivo secundario no es más que permitir controlar y mostrar al usuario la diferencias que existen entre las diferentes versiones de los archivos. Este objetivo está vinculado a su vez al objetivo secundario primero, citado anteriormente, gracias al cual podemos extraer y mostrar al usuario las diferencias en cuanto a código se refiere entre dos versiones diferentes de un mismo archivo. Esto nos ofrece la posibilidad de controlar si todo está funcionando correctamente y de poder observar con facilidad las diferencias entre versiones.

Este trabajo se ha madurado con la intención de ofrecer un primer acercamiento de la cuestión a los profesionales en la materia y de permitir explotar su gran potencial. Sin duda alguna queda mucho trabajo por hacer si queremos acercar esta prueba de concepto a conseguir que el día de mañana los usuarios puedan disfrutar de la seguridad que persigue este objetivo a largo plazo, que no es más que la realización de una extensión para los navegadores web. Son infinitas las posibilidades que esto ofrecería dentro del ámbito de la Ciberseguridad tan importante en plena revolución tecnológica.

1.3 Metodología

La investigación llevada a cabo en el presente Trabajo de Fin de Grado no es más que una investigación aplicada, un desarrollo práctico de un software que responde a unos objetivos concretos en base a los conocimientos y tecnologías ya existentes.

La metodología quedaría definida según el siguiente guion:

I. Desarrollo de un *back-end* incluyendo base de datos y motor de procesamiento de **JavaScript**.

II. Desarrollo de un *front-end* simplificado para mostrar informes y diferencias de ficheros **JavaScript**.
III. Como propuesta futura de mejora, desarrollo de una extensión de navegador web que permita realizar automáticamente el análisis web manual y recurrente.

En primer lugar, se ha diseñado un *back-end* con capacidad de almacenaje y de procesamiento de los scripts y librerías capaz de *hashear* a nivel de función/método cada uno de los ficheros **JavaScript**. Esta granularidad hash de los ficheros es clave para los propósitos de *Bast-ed*.

El servidor web está provisto de una base de datos capaz de almacenar los hashes, identificadores, versiones, fechas de control, entre otra información susceptible de ser almacenada. Además, incorpora un catálogo de urls con una serie de referencias a cado uno de los scripts y librerías **JavaScript** que ejecuta, que también se encuentran guardados en la máquina con la que ha sido desarrollado el proyecto.

Finalmente, se ha diseñado un *front-end* simplificado que permite replicar la misma función que la extensión de análisis de web y también mostrar, en caso de colisión entre librerías, un diff[3] con los cambios introducidos entre la original y la modificada.

También se han implementado una serie de funcionalidades adicionales que mejoran las capacidades de *Bast-ed* y que enriquecen la operativa global.

1.4 Resumen de los contenidos de la memoria

El planteamiento de esta memoria sigue el esquema marcado previamente en el índice del proyecto.

De esta forma, en el capítulo primero se presentan los motivos y necesidades que nos han llevado a la realización de la memoria.

En los capítulos 2 y 3 se explica la puesta en marcha del entorno utilizado en el proyecto, además de una pequeña introducción de conceptos esenciales necesarios para comprender el desarrollo del proyecto.

En los capítulos 4, 5, 6 y 7 se analiza en profundidad el código del software desarrollado, y se cierra la memoria con el desarrollo de las conclusiones, la presentación de la bibliografía y los anexos.

[3] Diferencias entre versiones.

2 TECNOLOGÍAS EMPLEADAS

Este proyecto propone una aplicación web que emplea tecnologías como **Java**, **HTML**, **CSS** y **JavaScript**.

En los siguientes apartados se explica en forma esencial los conceptos necesarios a conocer para entender el desarrollo del presente proyecto. Así como el resto de las tecnologías empleadas, que son más fáciles de entender una vez introducidos en el mundo de la programación orientada a objetos de la que hacemos uso en Java.

2.1 Qué es un objeto

"Java es un lenguaje que originalmente fue desarrollado por un grupo de ingenieros de *Sun*, utilizado por *Netscape" [CRC, 2017]* (empresa responsable del navegador *NetsCape navigator*) y posteriormente como base de la tecnología **JavaScript**. Si bien su uso se destaca en el desarrollo web, sirve para crear todo tipo de aplicaciones (locales, intranet o internet).

Partimos de la base de que todo lo que hay en Java son objetos, que son el pilar de este famoso lenguaje de programación, de forma que se puede decir que todo puede verse como un objeto. Sin embargo, para ser más claro un objeto puede verse como un elemento de software que cumple con las siguientes características:

Encapsulamiento

"Encapsulamiento significa que el objeto es autocontenido, o sea que la misma definición del objeto incluye tanto los datos que éste usa (**atributos**) como los procedimientos (**métodos**) que actúan sobre los mismos" [L. Castillo, J. Dimas, 2016].

Herencia

"Respecto a la herencia, significa que se pueden crear nuevas **clases** que se hereden de otras preexistentes; esto simplifica la programación, porque las **clases** hijas incorporan automáticamente los métodos de las madres" [L. Castillo, J. Dimas, 2016].

2.2 Qué es una clase

En el apartado anterior se habla de las **clases** para explicar el significado de herencia en cuanto a un objeto se refiere, lo cual hace necesario introducir qué es una **clase** y las oportunidades que nos ofrece en sí.

"Las **clases** en Java son básicamente una plantilla que sirve para crear un objeto. Si imaginásemos las **clases** en el mundo en el que vivimos, podríamos decir que la **clase** "Persona" es una plantilla sobre cómo debe ser un ser humano. Todos y cada uno de nosotros,

los seres humanos, somos objetos de la **clase** "Persona", ya que todos somos personas. La **clase** "Persona" contiene la definición de un ser humano, mientras que cada ser humano es una instancia u objeto de dicha **clase**" [A. G. González, 2015].

Para que esta analogía que más clara se presenta el siguiente ejemplo.

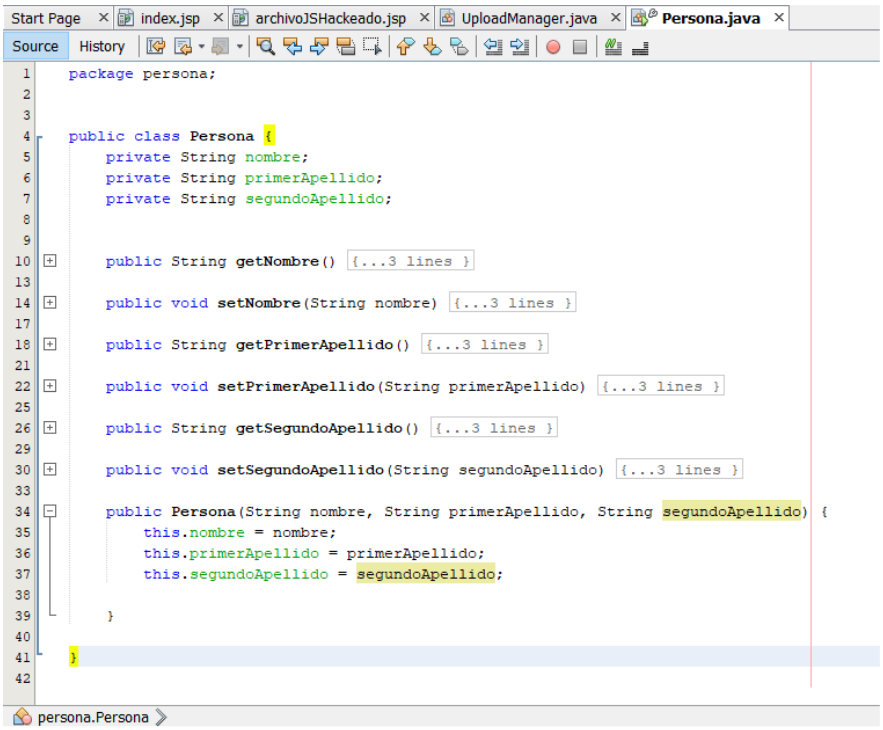

Ilustración 1. Clase "Persona"

Como se puede apreciar en la imagen se ha creado una **clase** "Persona" que contiene tres propiedades: *nombre*, *primerApellido* y *segundoApellido*. Este ejemplo sirve como analogía pues en la realidad cada persona está identificada con un nombre y unos apellidos concretos (esta sería la **clase**, es decir, la plantilla) pero sus diferentes combinaciones designan diferentes personas (objetos determinados de dicha **clase**). También se introduce un *constructor* al final del código gracias al cual se podría crear un nuevo registro, como en la realidad, al nacer una nueva persona se puede registrar con un nombre y unos apellidos determinados. Por último, la **clase** "Persona" contiene una serie de *métodos* que permiten obtener y establecer propiedades de dicha **clase**, como el nombre y los apellidos.

2.3 Qué es un servlet

"Un **servlet** es un objeto de java que pertenece a una clase que extiende de javax.servlet.http.HttpServlet, este nos permite crear aplicaciones web dinámicas, lo que quiere decir que podemos realizar consultas, insertar y eliminar datos" [Guía para montar un servlet, 2018].

"Son pequeños programas escritos en Java que admiten peticiones a través del protocolo HTTP. Los **servlets** reciben peticiones desde un navegador web, las procesan y devuelven una respuesta al navegador normalmente en HTML. Para realizar estas tareas podrán utilizar las **clases** incluidas en el lenguaje Java. Estos programas son los intermediarios entre el cliente (casi siempre navegador web) y los datos (Base de datos, BBDD)" [Qué es un servlet, 2018].

2.4 Qué es el número hash

"Los **hash** o funciones de resumen son algoritmos que crean a partir de una entrada (ya sea un texto, una contraseña o un archivo), una salida alfanumérica de longitud normalmente fija que representa un resumen de toda la información que se le ha dado (es decir, a partir de los datos de la entrada crea una cadena que solo puede volverse a crear con esos mismos datos)" [M. Ramírez, 2013].

Las funciones **hash** representan mediante un valor en hexadecimal un archivo o conjunto de datos que suele ser de mayor tamaño que el **hash**, independientemente del propósito de su uso.

Este sistema de criptografía usa algoritmos que conociendo el **hash** es imposible saber los datos que contiene el archivo, lo que indica que es una **función unidireccional**[4].

Uso específico del hash en el proyecto

En este proyecto se utiliza para calcular el **hash** concreto del archivo **JavaScript** que se desea subir, y para también para calcularlo cuando el archivo llega al servidor. De esta forma podemos comprobar si el archivo ha sido modificado por el camino y asegurarnos de que llega correctamente.

La función **hash** utilizada en el proyecto es sha512:

[4] Vía https://www.genbetadev.com/seguridad-informatica/que-son-y-para-que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales)

SHA-512 opera en un bloque de mensajes de 1024 bits y un valor **hash** intermedio de 512 bits. Esencialmente es un algoritmo de autenticación de bloques de 512 bits que cifra el valor **hash** intermedio usando el bloque de mensaje como clave.

La compresión del algoritmo SHA-512 se realiza mediante el diagrama que se muestra en la ilustración 2:

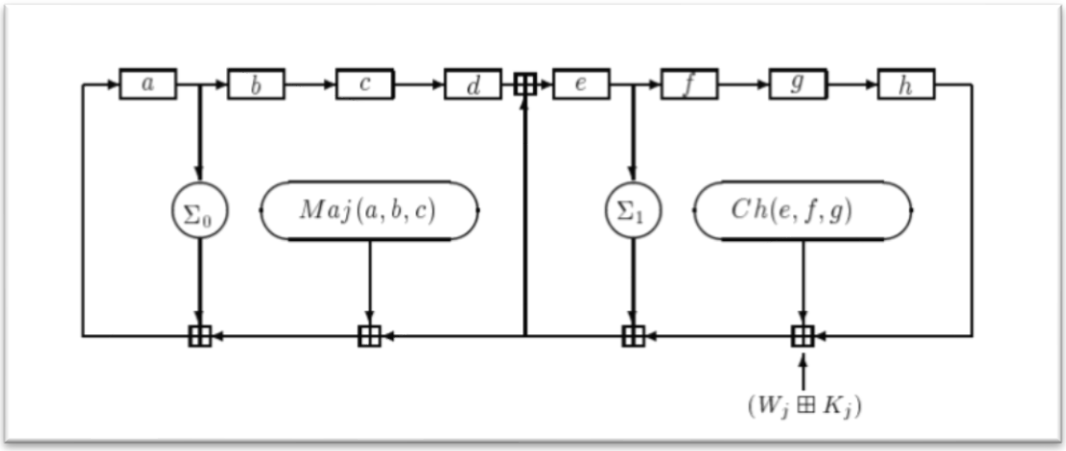

Ilustración 2. Diagrama del algoritmo SHA-512

2.5 Qué es el patrón MVC

El MVC o Modelo-Vista-Controlador es un patrón utilizado en el desarrollo de software que se basa en : **modelos**, **controladores** y **vistas**, separa la lógica de la aplicación de la lógica de su **vista**.

Los modelos

"Se encargan de los datos generalmente, pero no obligatoriamente, consultando la base de datos. Actualizaciones, consultas, búsquedas, etc. Todo esto va aquí, en el modelo" [MVC, 2015].

Los controladores

Se encargan de supervisar, es decir, de llevar a cabo las órdenes del usuario, buscar los datos en el modelo y finamente enviarlos a la **vista**.

Las vistas

"Son la representación visual de los datos, todo lo que tenga que ver con la interfaz gráfica va aquí. Ni el modelo ni el **controlador** se preocupan de cómo se verán los datos, esa responsabilidad es únicamente de la **vista**" [MVC, 2015].

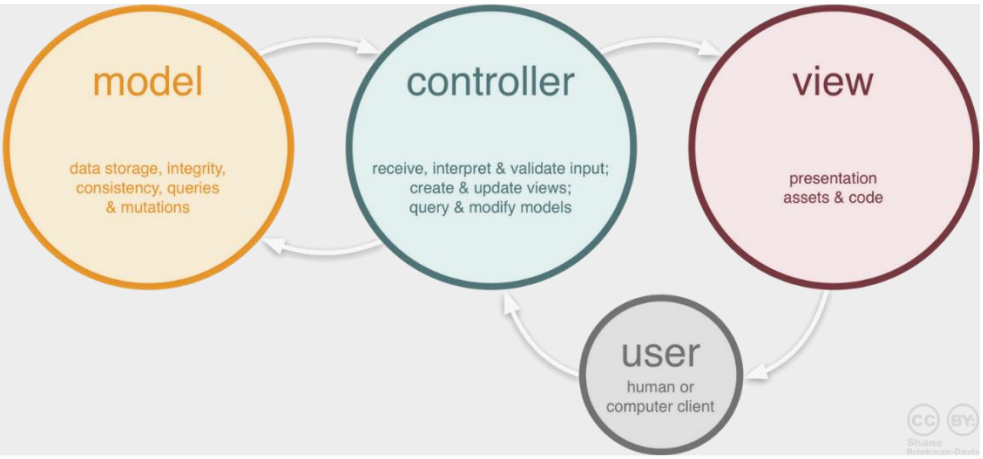

Ilustración 3. Resumen del Patrón MVC

2.6 Qué es HTML y CSS

Definiéndolo en forma sencilla: "HTML es el lenguaje que se utiliza para crear las páginas web a las que se accede mediante Internet" [Desarrollo de Aplicaciones Web, 2018]. Más concretamente, HTML es el lenguaje con el que se "escriben" la mayoría de las páginas web.

Las páginas web son creadas con el lenguaje HTML como los programas que autogeneran páginas en HTML y los navegadores web como, por ejemplo, Google Chrome. Pese a que HTML es un lenguaje de programación utilizado por ordenadores y navegadores web, es sencillo de entender y manejar por las personas. En realidad, HTML son las siglas de *HyperText Markup Language,* que podría ser traducido como *Lenguaje de Formato de Documentos para Hipertexto*[5].

Por su parte, Cascading Style Sheets (*CSS*), traducido literalmente al español, como *Hojas de estilo en cascada*, es un lenguaje para la especificar cómo los documentos se presentan a los usuarios. Definiéndolo de una forma sencilla, "CSS es el lenguaje de programación que se encarga de darle estilo a una página web HTML" [Desarrollo de Aplicaciones Web, 2018].

[5] Vía http://www.um.es/docencia/barzana/DAWEB/Introduccion-a-html-y-css.html

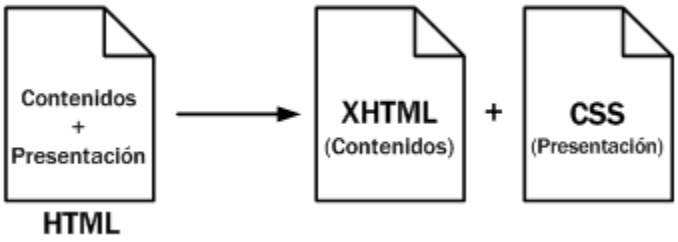

Ilustración 4. Esquema de uso HTML y CSS

2.7 Qué es JavaScript

"**JavaScript** sirve principalmente para mejorar la gestión de la interfaz cliente/servidor. Un script **JavaScript** insertado en un documento HTML permite reconocer y tratar localmente, es decir, en el cliente, los eventos generados por el usuario. Estos eventos pueden ser el recorrido del propio documento HTML o la gestión de un formulario" [G. A. Diaz, 2016].

Por ejemplo, cuando en una web se inserta un botón para que realice una acción en concreto, el código que extendió la funcionalidad del botón viene dada por código **JavaScript**.

Ilustración 5. Logos corporativos HTLM, CSS y JS

Estos lenguajes, como se comprende revisados los apartados anteriores, ofrecen una muy buena sinergia entre ellos ya que los tres trabajan a nivel del cliente, con la ayuda de su navegador web.

2.8 Qué es Bootstrap

"Bootstrap es un *framework*[6] desarrollado y liberado por Twitter que tiene como objetivo facilitar el diseño web. Permite crear de forma sencilla webs de diseño adaptable, es decir, que se ajusten a cualquier dispositivo y tamaño de pantalla y siempre se vean igual de bien.

[6] Es un esquema (un esqueleto, un patrón) para el desarrollo y/o la implementación de una aplicación.

Es un código abierto, por lo que lo podemos usar de forma gratuita y sin restricciones" [H. H. Taboada, 2018].

En este proyecto ha sido utilizada la versión Bootstrap 3.

Ilustración 6. Logo corporativo Bootstrap

El uso de este *framework* ha permitido mejorar el aspecto visual de las **vistas** de forma sencilla ya que ofrece infinidad de posibilidades.

3 CONFIGURACIÓN Y PUESTA EN MARCHA DEL ENTORNO

3.1 Descarga e instalación del software

A continuación, se detalla el software que ha sido necesario emplear para desarrollar este proyecto.

En este caso concreto, ha sido desarrollado con el IDE[7] **NetBeans 8.2**, que permite desarrollar rápida y fácilmente aplicaciones Java de escritorio, móviles y web, así como aplicaciones *HTML5* con *HTML*, *JavaScript* y *CSS*. Es gratuita y de código abierto y tiene una gran comunidad de usuarios y desarrolladores en todo el mundo[8].

Ilustración 7. Logo corporativo del IDE NetBeans

Como el proyecto desarrolla una aplicación web con la utilización de código Java, también es necesario instalar **JDK** (*Java Development Kit*), que en este caso ha sido utilizada la versión **JDK Standard Edition 8.**

Java es un lenguaje de programación desarrollado por la multinacional *Oracle Corporation* y el **JDK** es necesario ya que se encarga de compilar el código desarrollado en el IDE **NetBeans.**

[7] Entorno de desarrollo interactivo que consiste en un editor de código fuente, herramientas de construcción automáticas y un depurador. Vía https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado
[8] Vía https://netbeans.org/features/index.html

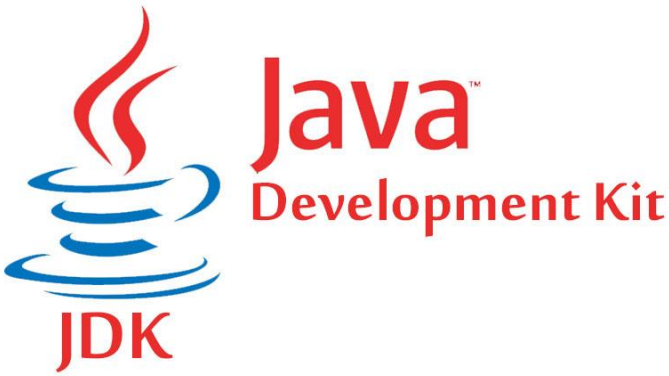

Ilustración 8. Logo corporativo de JDK

Al tratarse de una aplicación web, también requiere de un servidor, y para ello se utiliza el "**GlassFish server**, que es un servidor de aplicaciones de software libre desarrollado por *Sun Microsystems*, una compañía adquirida por *Oracle Corporation* (Java)" [N. I. F. Suarez, 2011]. En este proyecto en concreto se ha utilizado la versión **GlassFish 4.1.1**[9].

Ilustración 9. Logo corporativo de GlassFish Server

Respecto a la base de datos empleada en el proyecto ha sido implementada en **MySQL.** "**MYSQL** es un sistema de gestión de bases de datos relacional desarrollado bajo licencia dual: licencia pública general y licencia comercial propiedad de *Oracle Corporation*, considerada como la base de datos de código abierto más popular del mundo" [Wikipedia, 2018]. Para la creación de la base de datos es necesario tanto el motor de base de datos C*ommunity Edition* como **MySQL Workbech 6.3** para la creación de las tablas[10].

[9] Vía https://es.wikipedia.org/wiki/GlassFish
[10] Vía https://es.wikipedia.org/wiki/MySQL

Ilustración 10. Logo corporativo de MySQL

Para que el servidor web desarrollado en Java acceda a la base de datos es necesario el driver correspondiente que en este caso se trata del **mysql-connector-java5.1.46**.

Ilustración 11. Logo corporativo de Google

Por último, hemos empleado la librería de Google de código abierto *diffutils-1.2.1.jar* con la que realizamos las comparaciones entre las diferentes versiones de los archivos **JavaScript** que se van a explicar en los apartados siguientes.

3.2 Diagramas

En este apartado se analiza de forma general el comportamiento del programa a través del desglose y relación de sus componentes y funciones en tres diagramas representados a continuación.

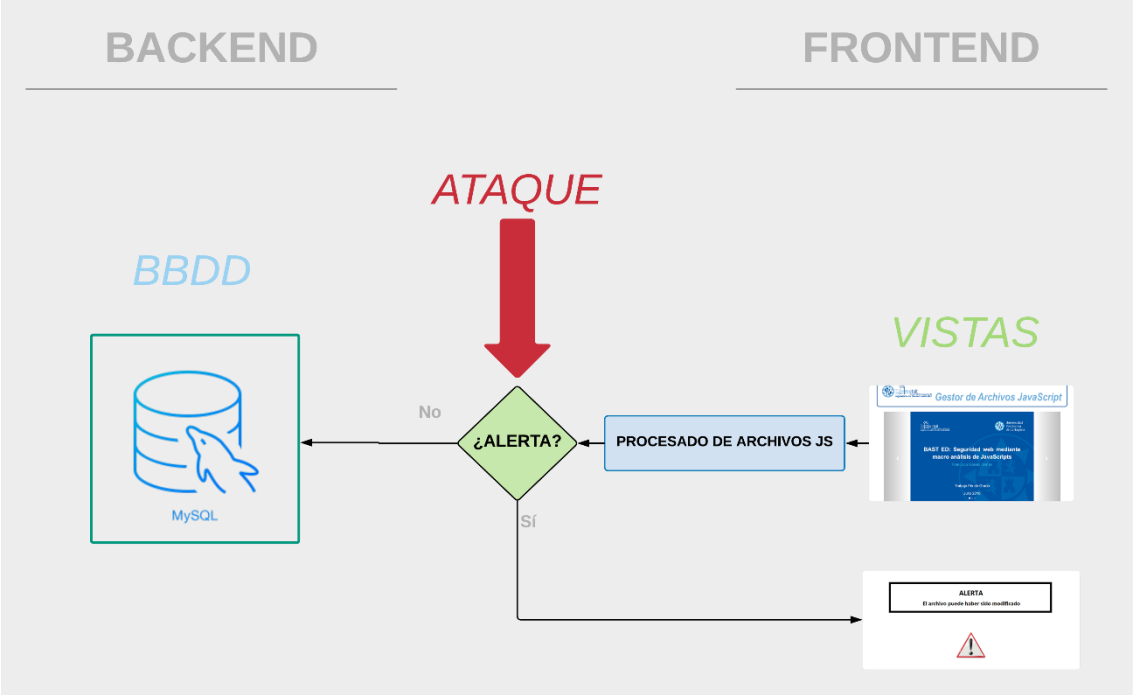

Ilustración 12. Diagrama general del software

Este primer diagrama general describe el comportamiento del software en cuestión. Este está conformado por dos capas o facetas que podemos ver representadas como *back-end* y *front-end*. El término *back-end* hace referencia a la capa de acceso a los datos, en la cual hemos implementado la base de datos (BBDD) donde se almacenan los archivos **JavaScripts** (JS) que el usuario carga desde la interfaz *front-end*. El *front-end*, por su parte, se definiría como aquella capa visible del programa o parte del software en la que se interacciona con el usuario.

Según queda representado en la ilustración, el funcionamiento del software desarrollado implica el siguiente proceso: en primer lugar, el usuario accede a un formulario a través de la interfaz desde la vista principal. Una vez lanzado el proyecto, a través de esta interfaz, el usuario podría cargar archivos exclusivamente de tipo **JavaScript**. En segundo lugar, ya cargado el archivo **JavaScript** en el código, el software realiza una serie de diversas comprobaciones detalladas más adelante para verificar si el archivo ha sido modificado durante la carga. En última instancia, si el archivo hubiese sido modificado, el usuario recibe como respuesta la vista archivo modificado que muestra como mensaje que el archivo ha sido modificado durante la carga. Si por lo contrario el archivo ha sido subido con éxito, este se almacería en la BBDD.

A continuación, vamos a analizar en detalle los componentes y funciones específicos de cada una de las facetas del software: *back-end* y *front-end*, que interactúan en el proceso descrito anteriormente.

Back-end

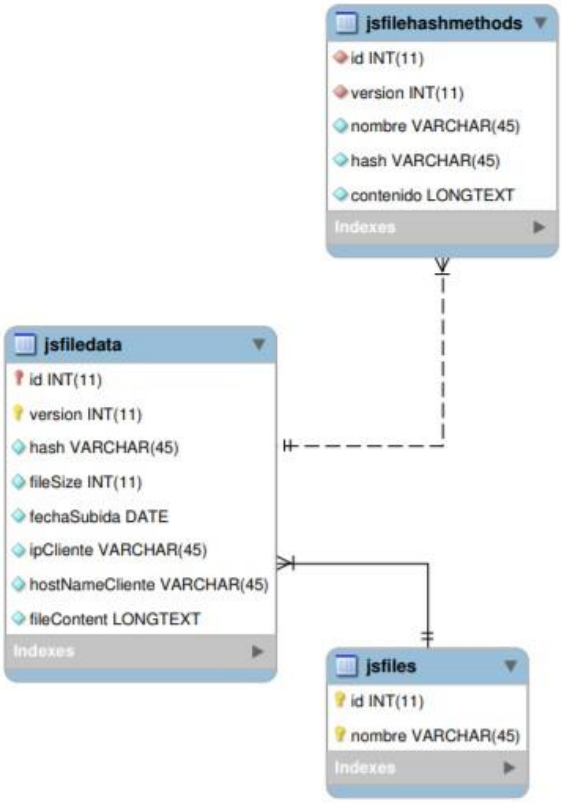

Ilustración 13. Diagrama representativo del *back-end*

El *back-end*, como podemos ver en el diagrama general, es en sí la BBDD, que a su vez consta de tres tablas relacionada. La tabla **jsfiles,** únicamente tiene dos campos: el *id* que es auto-incremental y el *nombre* del archivo en cuestión. Estos conforman la clave primaria de la tabla. Sin embargo, en el planteamiento del proyecto se contempla la posibilidad de subir diferentes versiones de un mismo archivo, por lo que es necesario otra tabla: la **jsfiledata**. En esta tabla el *id* del archivo es clave externa de **jsfiles** y la *versión* es clave primaria, para controlar que no se puedan introducir versiones iguales. El resto de los campos contenidos en esta tabla ofrecen información del archivo en cuestión como: el **hash**, el tamaño o la fecha de subida.

Por último y no menos importante, encontramos la tabla **jsfilehashmethods**, necesaria para almacenar los diferentes métodos de los diferentes archivos **JavaScript**. Como clave externa

posee tanto el *id* como la *versión* y, además, otros campos con información relevante como el *nombre*, el **hash** y el *contenido* del método.

Respecto a la relación entre las tablas, es bastante intuitiva. Puesto que un archivo puede tener varias versiones, la relación entre las tablas **jsfiles** y **jsfiledata** es de una a varias dependiendo del número de versiones de un mismo archivo. Al igual que entre las tablas **jsfiledata** y **jsfilehashmethods**, ya que una versión concreta de un archivo concreto puede poseer uno o varios métodos.

Front-end

Vistas	*Modelo*	*Controladores*	*Helpers*
Index.jsp	DataBaseConexion.java	AJAXFileSelected.java	ArchivoSeleccionado.java
	JSFile.java	AJAXManager.java	ComparacionResultante.java
archivoJSHackeado.jsp	JSFileHashMethod.java	AjaxComparadorDeVersiones.java	ComparadorDeVersiones.java
	JSFileData.java	UploadManagaer.java	Hash.java
versionOriginal.html	Modelo.java	AJAXContenidoVersiones.java	VersionYFechaSubida.java
versionRevisada.html			Versiones.java
			ContenidoVersiones.java

Ilustración 14. Distribución del patrón MVC

A pesar de que en los apartados siguientes se va a explicar en detalle el funcionamiento del código, en resumen, el *patrón MVC* (Modelo-Vista-Controlador) responde al siguiente esquema:

Las **vistas**, compuesta por: la vista principal en la que se muestra el formulario a través del cual el usuario puede cargar los archivos **JavaScript** y responde a la **clase** *idex.jsp*. Junto a esta también hay otra **vista**, vista archivo modificado, que se muestra si se produce un ataque al archivo cuando este se está subiendo y responde a la **clase** *ArchivoJSHackeado.jsp*.

Los **controladores**, por su parte, son los diferentes **servlets** que se encargan de comunicarse con las **vistas** para tratar la información y responderle lo que se les solicita.

Respecto al **modelo**, se trata de la **clase** *modelo.java*, que junto a las **clases** derivadas: *JSFile.java, JSFileData.java, JSFileHashMethod.java y DataBaseConexion.java*, proveen al **controlador** de todo lo que necesita en su proceso de comprobación y almacenaje.

4 VISTAS

4.1 archivoJSHackeado.jsp

Ilustración 15. Apariencia Vista Archivo modificado

Funcionamiento a nivel de usuario

Cuando el archivo modificado ha sido *hackeado* al intentar subir el archivo aparece en pantalla una alerta de seguridad que avisa al usuario, como la que vemos en la ilustración 14. El **servlet** que gestiona la vista archivo modificado: *archivoJSHackeado.jsp*, es el *UploadManager.java.*

Funcionamiento del código

Es un código HTML muy simple, lo único que contiene es un título, una imagen y un gif.

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Alerta de Seguridad</title>
    </head>
    <body>
        <div class="img-responsive" style="margin-top: 80px; margin-bottom: 50px">
            <img src="imagenes/bannerAlerta.png" style=" display: block ;margin: auto"
alt=""/>
        </div>
        <div class="img-responsive" style=" margin-bottom: 30px">
            <img src="imagenes/alerta.gif" style=" display: block ;margin: auto" alt=""/>
        </div>
    </body>
</html>
```

4.2 index.jsp

Ilustración 16. Apariencia Vista Principal I

Ilustración 17. Apariencia Vista Principal II

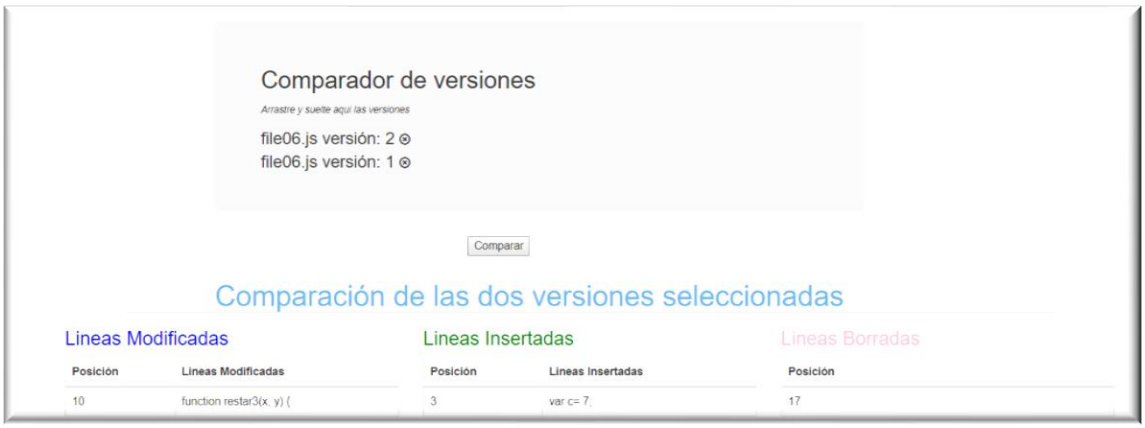

Ilustración 18. Apariencia Vista Principal III

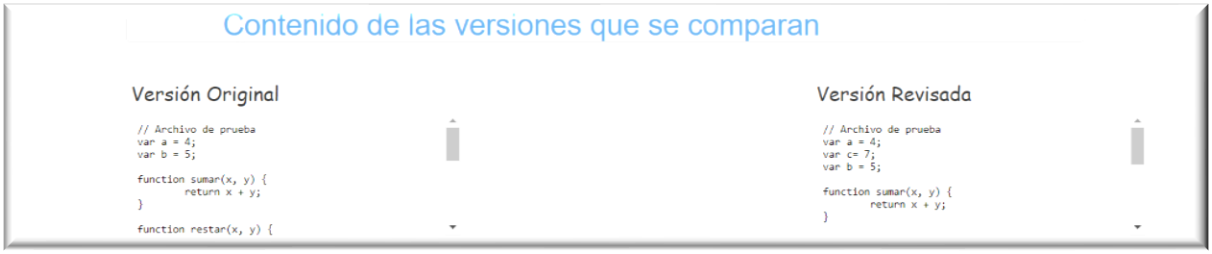

Ilustración 19. Apariencia Vista Principal IV.

Funcionamiento a nivel de usuario

Como podemos observar en las ilustraciones anteriores se muestra el aspecto de la vista principal *index.jsp*. En la ilustración 16 se muestra un pequeño carousel[11] de imágenes, las cuales, son una pequeña presentación del proyecto. En la ilustración 17 se muestra el botón que permite subir los archivos **JavaScript** a la base de datos, además de un desplegable en que se pueden seleccionar las diferentes versiones del archivo, mostrando al usuario las diferentes versiones guardadas en la base de datos del archivo seleccionado. En la ilustración 18 se muestra en el centro de la imagen un jumbotron[12] en que podemos arrastrar las diferentes versiones para posteriormente comparar con el botón que hay debajo, además en la parte de debajo de la imagen hay tres tablas que se rellenan dinámicamente con las diferencias que existen entre las versiones que se comparan. En la ilustración 19 se muestra el contenido de las versiones que se están comparando, a la izquierda el contenido de la versión original y a la derecha el contenido de la versión revisada.

[11] Se trata de un widget que permite crear una presentación de imágenes dinámica.
[12] Herramienta que provee el Bootstrap que permite presentar un contenido muy destacado.

Esta **vista** comprueba internamente que no se haya subido el mismo archivo, de tal forma que se muestra un pop-up[13] cuando se intenta subir un archivo que está subido anteriormente como podemos observar en la ilustración 20. Si el archivo posee un nombre con el que no se haya subido ningún archivo anteriormente, se habilitará el botón "Enviar" en el formulario que a priori está desactivado.

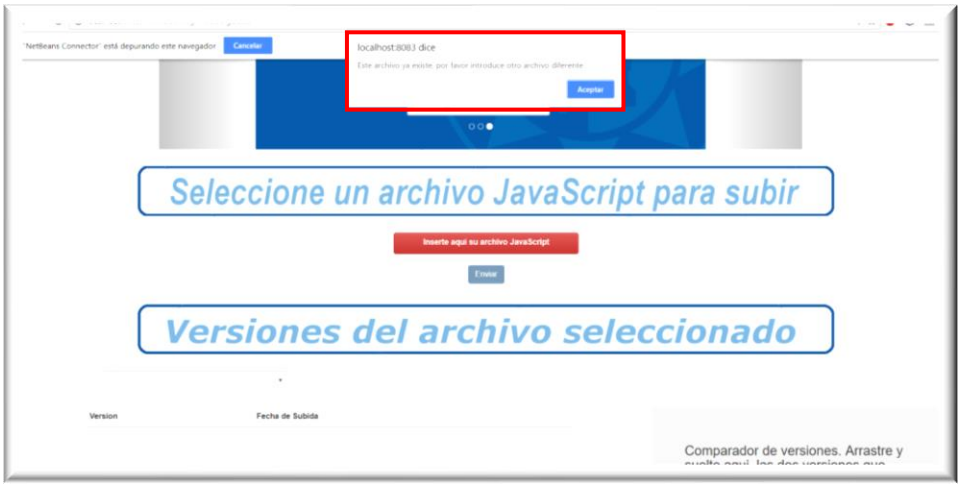

Ilustración 20. Apariencia Vista Principal IV.

Existen otras dos variantes respecto a la posibilidad de subir un mismo archivo dos veces: la primera, no permitiría cargar el archivo ya que al calcular el **hash** del archivo y comprobar que existe otra combinación exacta en *nombre* y **hash** en la base de datos, no se habilita el botón "Enviar" y se mostraría el pop-up anterior; la otra, permitiría cargar el archivo ya que, si posee el mismo *nombre*, pero es una versión diferente y, por tanto, también su **hash**, se habilitaría el botón "Enviar".

Funcionamiento del código

Como la vista principal contiene mucho contenido de código vamos a ir analizando por partes dicho código, lo que hará más fácil seguir su funcionamiento. En todo caso el código se explica de forma general y no línea por línea.

Respecto del código HTML:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
        <title>Reto Ciberseguridad y Big Data</title>
```

[13] Denota un elemento emergente que se utiliza generalmente dentro de la tecnología web.

```
        <link href="css/bootstrap.css" rel="stylesheet" type="text/css"/>
        <link href="css/bootstrap-theme.css" rel="stylesheet" type="text/css"/>
        <style>
            .btn-file {
                display: block;
                margin: auto;
                position: relative;
                overflow: hidden
            }

            .btn-file input[type=file] {
                position: absolute;
                top: 0;
                right: 0;
                min-width: 100%;
                min-height: 100%;
                font-size: 100px;
                text-align: right;
                filter: alpha(opacity=0);
                opacity: 0;
                outline: none;
                background: white;
                cursor: inherit;
                display: block;
            }

            .carrousel-pos {
                margin-bottom: 30px;
                display: block;
                margin: auto;
                width: 70%;
            }

            #clearVersion1, #clearVersion2 {
                visibility: hidden
            }

        </style>
    </head>
    <body>
        <div class="container-fluid">
            <div class="img-responsive" style="margin-top: 20px; margin-bottom: 30px">
                <img src="imagenes/bannerprueba2final.png" style="display: block; margin: auto"
alt=""/>
            </div>

            <div id="myCarousel" class="carousel slide carrousel-pos" data-ride="carousel">
                <!-- Indicators -->
                <ol class="carousel-indicators">
                    <li data-target="#myCarousel" data-slide-to="0" class="active"></li>
                    <li data-target="#myCarousel" data-slide-to="1"></li>
                    <li data-target="#myCarousel" data-slide-to="2"></li>
                </ol>

                <!-- Wrapper for slides -->
                <div class="carousel-inner">
                    <div class="item active">

                        <img src="imagenes/img1.png" alt="" style="width: 70%; display: block;
margin-left: auto; margin-right: auto">
                    </div>

                    <div class="item">
                        <img src="imagenes/img2.png" alt="" style="width: 70%; display: block;
margin-left: auto; margin-right: auto">
                    </div>

                    <div class="item">
                        <img src="imagenes/img3.png" alt="" style="width: 70%; display: block;
margin-left: auto; margin-right: auto">
                    </div>
                </div>
```

```
                <!-- Left and right controls -->
                <a class="left carousel-control" href="#myCarousel" data-slide="prev">
                    <span class="glyphicon glyphicon-chevron-left"></span>
                    <span class="sr-only">Previous</span>
                </a>
                <a class="right carousel-control" href="#myCarousel" data-slide="next">
                    <span class="glyphicon glyphicon-chevron-right"></span>
                    <span class="sr-only">Next</span>
                </a>
            </div>

            <div class="row" style="padding-left: 20px">
                <fieldset>

                    <div class="img-responsive" style="margin-top: 30px; margin-bottom: 30px">
                        <img src="imagenes/bannerprueba7final.png" style="display: block; margin:
auto" alt=""/>
                    </div>
                    <form action="uploadmanager" method="post"
                          enctype="multipart/form-data" id="form1" name="form1">
                        <div class="btn btn-danger btn-file" style="width: 20%; margin-bottom:
20px;">
                            <label for="file"> Inserte aquí su archivo JavaScript  </label>
                            <input type="file" accept="application/javascript" name="file"
id="file">
                        </div>


                        <!--Información/metadatos del archivo seleccionado-->
                        <input type="hidden" id="hashFile" name="hashFile">
                        <input type="hidden" id="fileSize" name="fileSize">


                        <!--Enviar archivo al servidor-->

                        <input type="submit" class="btn btn-primary" id="enviar" disabled
style="display: block; margin-top: 30px; margin: auto">



                    </form>
                </fieldset>
```

Para esta primera parte del código, lo primero que encontramos es el título *Reto Ciberseguridad y Big Data* a lo que le siguen dos líneas de código bastante importantes en lo que a la apariencia de la página web se refiere:

```
<link href="css/bootstrap.css" rel="stylesheet" type="text/css"/>
<link href="css/bootstrap-theme.css" rel="stylesheet" type="text/css"/>
```

Estas dos líneas nos permiten utilizar Bootstrap, tecnología explicada en un apartado anteriormente. Seguidamente encontramos unas pequeñas reglas de CSS asignadas posteriormente a diferentes componentes de la página. Dentro del body[14] encontramos, el primer banner de la página que pone "Gestor de Archivos **JavaScript**", el carousel con las tres imágenes que contiene y el código necesario para poder pasar de imágenes de forma manual. En el body también encontramos el siguiente banner de la página "Selecciones un archivo **JavaScript** para subir" además del botón que permite subir archivos "Inserte aquí su

[14] Hace referencia al cuerpo de la página.

archivo **JavaScript**" y el botón para enviar en contenido del archivo al **controlador** pertinente. Además, contiene las dos siguientes líneas:

```
<input type="hidden" id="hashFile" name="hashFile">
<input type="hidden" id="fileSize" name="fileSize">
```

Que nos permiten obtener los atributos de **hash** y tamaño del archivo sin necesidad de mostrarlo en pantalla gracias a que son de tipo "hidden".

```
                <output id="resultado"></output>

                <div class="img-responsive" style="margin-top: 30px; margin-bottom: 30px">
                    <img class="img-responsive" src="imagenes/bannerprueba6final.png" style="
display: block ;margin: auto" alt=""/>
                </div>
                <div class="row">
                    <select id="filesList" class="form-control" style="margin-left: 150px; width:
20%; margin-bottom: 30px">
                        <option></option>
                        <%
                            Modelo m;

                            m = new Modelo();

                            List<JSFile> listado;

                            listado = m.getJsFile();

                            for (JSFile e : listado) {

                        %>
                        <option value="<%=e.getId()%>"><%=e.getName()%></option>
                        <%
                            }
                        %>
                    </select>
                </div>
                <div class="row">
                    <div class="col-lg-12" style="margin: auto; width: 90%; margin-left: 70px">


                        <table class="table table-striped" style=" margin: auto; margin-bottom:
30px">
                            <thead>
                                <tr>
                                    <th>Version</th>
                                    <th>Fecha de Subida</th>
                                    <th>Hash-sha512</th>
                                </tr>
                            </thead>
                            <tbody id="tabla1">
                            </tbody>
                        </table>
                    </div>
                    <div class="col-lg-4" style="display: block; margin: auto; width: 50%;margin-
left: 350px">
                        <div class="jumbotron" ondrop="drop(event)" ondragover="allowDrop(event)">
                            <h2 title="Muestra cambios de una version inferior respecto a una mas
reciente">
                                Comparador de versiones</h2>
                            <h6 style="font-style: italic; margin: auto; margin-top: 15px; margin-
bottom: 15px">Arrastre y suelte aqui las versiones</h6>

                            <div>
                                <p id="version1" data-version="" data-jsfile="" style="display:
inline"></p>
                                <span id="clearVersion1" class="glyphicon glyphicon-remove-circle"
                                      onclick="document.getElementById('version1').innerHTML = '';
                                               this.style.visibility = 'hidden';

document.getElementById('compararVersiones').disabled = true" aria-hidden="true">

                                </span>
                            </div>
```

```
                            <div>
                                <p id="version2" data-version="" data-jsfile="" style="display:
inline"></p>
                                <span id="clearVersion2" class="glyphicon glyphicon-remove-circle"
                                      onclick="document.getElementById('version2').innerHTML = '';
                                               this.style.visibility = 'hidden';

document.getElementById('compararVersiones').disabled = true" aria-hidden="true">

                                </span>
                            </div>

                        </div>
                        <!-- Se habilita solamente si hay dos versiones para comparar -->
                        <input type="button" id="compararVersiones" value="Comparar" disabled
style="margin-bottom: 30px; margin-left: 320px">
                    </div>


                </div>

            </div>

            <div class="row">
                <div class="panel panel-default" style="width: 70%; margin: auto">
                    <h1  class="panel-primary" style="display: block; margin: auto; margin-top: 30
px; width: 80%;color: #66afe9">Comparación de las dos versiones seleccionadas</h1>
                </div>
                <div class="row" style="width: 80%; margin-top: 40px; margin: auto">
                    <div class="col-lg-4">
                        <h3 style="color: blue">Lineas Modificadas</h3>
                        <div id="lineasModificadas">
                            <table class="table table-bordered">
                                <thead>
                                    <tr>
                                        <th>Posición</th>
                                        <th>Lineas Modificadas</th>
                                    </tr>
                                </thead>
                                <tbody id="lineasModificadasTbl">
                                </tbody>
                            </table>

                        </div>
                    </div>
                    <div class="col-lg-4">
                        <h3 style="color: green">Lineas Insertadas</h3>
                        <div id="lineasInsertadas">
                            <table class="table table-bordered">
                                <thead>
                                    <tr>
                                        <th>Posición</th>
                                        <th>Lineas Insertadas</th>
                                    </tr>
                                </thead>
                                <tbody id="lineasInsertadasTbl">
                                </tbody>
                            </table>
                        </div>
                    </div>
                    <div class="col-lg-4">
                        <h3 style="color: pink">Lineas Borradas</h3>
                        <div id="lineasBorradas">
                            <table class="table table-bordered">
                                <thead>
                                    <tr>
                                        <th>Posición</th>
                                    </tr>
                                </thead>
                                <tbody id="lineasBorradasTbl">
                                </tbody>
                            </table>
                        </div>
                    </div>
                </div>
            </div>

        </div>
```

```
        <div class="col-lg-12">
            <div class="panel panel-default" sandbox style="width: 70%; margin: auto; margin-
bottom: 30px">
                <h1  class="panel-primary" sandbox style="display: block; margin: auto; margin-
top: 30 px; width: 80%;color: #66afe9">Contenido de las versiones que se comparan</h1>
            </div>
            <div class="col-lg-6">
                <h3 style="font-family: fantasy; font-family: cursive; display: block; margin-
left: 30%">Versión Original</h3>
            <iframe id="if1"  src="versionOriginal.html"  style="width: 50%; display: block;
margin-left: 30%; margin-bottom: 30px"  scrolling="yes"  frameborder="0" style="margin-bottom:
30px">
            </iframe>
            </div>

            <div class="col-lg-6">
            <h3 style="font-family: fantasy; font-family: cursive; display: block; margin-left:
30%">Versión Revisada</h3>
            <iframe  id="if2" src="versionRevisada.html"  style="width: 50%; display: block;
margin-left: 30%; margin-bottom: 30px" scrolling="yes"  frameborder="0" style="margin-bottom:
30px">
            </iframe>
            </div>
```

Para esta parte del código **HTML**, primeramente, encontramos el siguiente banner "Versiones del archivo seleccionado" y seguidamente está el código necesario para el desplegable que nos permite seleccionar los diferentes archivos que se encuentran guardados en la base de datos junto con la tabla que nos permitirá mostrar, gracias a la ayuda en un script[15], las diferentes versiones existentes para el archivo seleccionado.

Después encontramos el código necesario para el jumbotron donde se van a arrastrar las diferentes versiones para posteriormente ser comparadas junto con el botón que permite comparar las versiones que se habilitara solo cuando estén seleccionadas dos versiones de un mismo archivo, como es lógico.

Seguidamente, encontramos el código necesario para las tres tablas que se encuentran al final de la página, que como en el caso de la anterior tabla, se rellenaran dinámicamente con el uso de un script que explicaremos más adelante.

Para finalizar, encontramos el código necesario para los dos iframes en los que se va a mostrar el contenido de las versiones que se están comparando. Gracias al atributo sandbox para los iframes, se va a poder mostrar el contenido de los archivos sin que el archivo pueda modificar la web.

En cuanto al código **JavaScript**, como en el caso del código **HTML** se va a explicar el código por partes.

```
// Comprobar el navegador tiene soporte para File API
            if (window.File && window.FileReader && window.FileList && window.Blob) {
                // Great success! All the File APIs are supported.
                // alert('The File APIs are fully supported in this browser.');
            } else {
                alert('The File APIs are not fully supported in this browser.');
            }
```

[15] Hace referencia al código JavaScript.

```
            document.getElementById("file").addEventListener("change", function () {

                var input = event.target;
                var reader = new FileReader();
                var resumen = "";   // hash del archivo
                var file = input.files[0];
                var fileSize = file.size;



                document.getElementById("fileSize").value = fileSize;


                // Cuando termina de almacenarse en memoria el contenido del archivo.
                // Similar a la tecnologia de AJAX
                reader.onload = function () {
                    var text = reader.result;

                    sha512(text).then(function (digest) {

                        console.log(digest);
                        resumen = digest;
                        // Guardamos el hash en el campo "hashFile", oculto, en el formulario
                        // con el objetivo de compararlo con el obtenido cuando el archivo llegue
                        // y sea recibido en el servidor.
                        document.getElementById("hashFile").value = resumen;

                        // Se envia por AJAX el nombre del archivo seleccionado y
                        // el hash para comprobar si existe en la base de datos.
                        // En caso de existir no se habilita el boton de enviar
                        // pues se estaria subiendo el mismo archivo.
                        var archivoSeleccionado = new Object();


                        // En el atributo values del input de tipe file tendriamos
                        // la ruta completa, por lo que utilizando una expresion
                        // regular eliminamos todo menos el nombre
                        archivoSeleccionado.nombre =
document.getElementById("file").value.replace(/^.*[\\\/]/, '');
                        archivoSeleccionado.hash = resumen;

                        // Se convierte el objeto anterior a una cadena (String) en
                        // formato JSON para poder ser enviada al servidor por AJAX
                        var archivoSeleccionadoStringJSON = JSON.stringify(archivoSeleccionado);

                        var xmlhttp = new XMLHttpRequest();

                        xmlhttp.onreadystatechange = function () {
                            if (this.readyState === 4 && this.status === 200) {
                                // No será recibido un String JSON procedente del
                                // servidor, solamente la respuesta de que exista
                                // o no existe el archivo javascript que se intenta
                                // subir
                                var respuesta = this.responseText;

                                console.log(respuesta);


                                if (respuesta === "existe") {
                                    alert("Este archivo ya existe, por favor introduce otro
archivo diferente")
                                    document.getElementById("enviar").disabled = true;
                                } else {
                                    document.getElementById("enviar").disabled = false;
                                }
                            }
                        };

                        xmlhttp.open("POST", "ajaxmanager", true);
                        xmlhttp.setRequestHeader("Content-type", "application/x-www-form-
urlencoded");
```

```
                    xmlhttp.send("q=" + archivoSeleccionadoStringJSON);


                });


            };

            // Al archivo se almacena en memoria de forma asincronica
            reader.readAsText(file);

        });
```

A pesar de que el propio código ya está comentado, se va a realizar una pequeña explicación general del mismo aquí. Lo primero es comprobar que el navegador soporte la API[16] de *file*. Por su parte, el script se encarga de asignar el valor a las propiedades *fileSize* y *hashFile*, además de controlar el estado de *file*. El estado del *file* se controla mediante el evento *change*. Cuando esto ocurre se calcula el **hash** del archivo a partir de una biblioteca de criptografía de **JavaScript** que hemos añadido, como se puede observar en el código sha512**(**text**).**then**(function (**digest**).**

Para comprobar si el archivo existe, es decir, si ha sido registrado en la base de datos previamente, es necesario convertir el objeto *archivoSeleccionado* a una cadena de caracteres mediante ***Json*** con la sentencia "var archivoSeleccionadoStringJSON = JSON.stringify(archivoSeleccionado)", para poder enviar el objeto al **servlet**. Después, el **servlet** *AJAXManager.java* se encarga de hacer las comprobaciones en la base de datos con el *nombre* del archivo y el *hash*, respondiendo posteriormente a "exite = true o false".

```
        // Manejador de evento onchange en la lista de seleccion de archivos
        document.getElementById("filesList").addEventListener("change", function () {


            var index = this.selectedIndex;

            var idFileSelected = this.options[index].value;
            var nombreFileSelected = this.options[index].text;


            // Se envia el id del archivo seleccionado por AJAX al Servlet
            // AJAXFileSelected que devolvera un listado de las versiones de
            // dicho archivo para renderizarlas en la tabla
            var objIdFileSelected = {
                id: idFileSelected,
                name: nombreFileSelected
            };

            var objIdFileSelectedStringJSON = JSON.stringify(objIdFileSelected);

            var http = new XMLHttpRequest();


            http.onreadystatechange = function () {
                if (this.readyState === 4 && this.status === 200) {
                    var objRespuesta = JSON.parse(this.responseText);

                    console.log(this.responseText);


                    // Renderizar la tabla a partir de aqui
                    var tableContent = '';
```

[16] Es la interfaz de programación de aplicaciones, abreviada del inglés: Application Programming Interface

```
                            // Agregamos un atributo id a cada fila para utilizar en
                            // la implementacion de Drag and Drop HTML5 API, en un primer
                            // momento, aunque despues el id no lo utilizamos, en su lugar
                            // agregamos la version y el nombre del archivo como atributos
                            // de datos, utilizando el atributo global de HTML5 data-*
                            var id = 0;
                            var fileSelected = document.getElementById("filesList")

.options[document.getElementById("filesList").selectedIndex].text;

                            for (x in objRespuesta) {
                                id++;
                                tableContent += '<tr draggable="true" id=\"fila' + id + '\"' + ' '
+
                                        'data-version=\"' + objRespuesta[x].version + '\" ' +
                                        'data-fileSelected=\"' + fileSelected + '\" ' +
                                        'ondragstart=\"drag(event)\"' + '>' +
                                        '<td>' + objRespuesta[x].version + '</td>' +
                                        '<td>' + objRespuesta[x].fechaSubida.day + ' - ' +
                                        objRespuesta[x].fechaSubida.month + ' - ' +
                                        objRespuesta[x].fechaSubida.year + '</td>' +
                                        '<td>' + objRespuesta[x].hash + '</td></tr>';
                            }

                            document.getElementById("tabla1").innerHTML = tableContent;

                            // Deshabilitar el boton para Comparar y limpiar el Comparador
                            document.getElementById("compararVersiones").setAttribute("disabled",
true);

                            document.getElementById("version1").innerHTML = "";
                            document.getElementById("version2").innerHTML = "";

                            document.getElementById("clearVersion1").style.visibility = "hidden";
                            document.getElementById("clearVersion2").style.visibility = "hidden";
                        }
                    };

                    http.open("POST", "ajaxfileselected", true);
                    http.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                    http.send("p=" + objIdFileSelectedStringJSON);
                });
```

Para este fragmento de código, encontramos el código necesario para el manejador del evento *onchange* de la lista de selección de archivos. Se hace una petición al **controlador** *AJAXFileSelected.java* que proporciona las diferentes versiones del archivo seleccionado tras una consulta a la base de datos.

También se encuentra el código para rellenar de forma dinámica la tabla que nos proporciona la fecha de subida, la versión y el hash del archivo que se ha seleccionado en el desplegable.

```
// Drag and Drop functions
            function allowDrop(ev) {
                ev.preventDefault();
            }


            function drag(ev) {
                var version = ev.target.getAttribute("data-version");
                var fileSelected = ev.target.getAttribute("data-fileSelected");
                ev.dataTransfer.setData("version", version);
                ev.dataTransfer.setData("fileSelected", fileSelected);
            }

            function drop(ev) {
                ev.preventDefault();
                var version = ev.dataTransfer.getData("version");
                var fileSelected = ev.dataTransfer.getData("fileSelected");

                var p1 = document.getElementById("version1");
```

```
                var p2 = document.getElementById("version2");

                if (p1.innerHTML === "" && p2.innerHTML === "" ||
                        (p1.innerHTML !== "" && p2.innerHTML !== "") ||
                        (p2.innerHTML !== "")) {
                    p1.setAttribute("data-version", version);
                    p1.setAttribute("data-jsfile", fileSelected);
                    p1.innerHTML = fileSelected + " versión: " + version + " ";
                    document.getElementById("clearVersion1").style.visibility = "visible";
                } else if (p1.innerHTML !== "") {
                    p2.setAttribute("data-version", version);
                    p2.setAttribute("data-jsfile", fileSelected);
                    p2.innerHTML = fileSelected + " versión: " + version + " ";
                    document.getElementById("clearVersion2").style.visibility = "visible";
                }


                // Habilitar el boton para Comparar, si ya hay dos versiones
                // seleccionadas
                if (p1.innerHTML !== "" && p2.innerHTML !== "") {
                    document.getElementById("compararVersiones").removeAttribute("disabled");
                } else {
                    document.getElementById("compararVersiones").setAttribute("disabled", true);
                }


            }
```

En este fragmento del código se muestra el script necesario para que se pueda hacer drap/drop[17] de las diferentes versiones hacia el jumbotron comentado anteriormente, además de que aparezcan las versiones pertinentes en el mismo jumbotron. Por último, si se han seleccionado dos versiones "**if (**p1.innerHTML **!==** "" **&&** p2.innerHTML **!==** ""**)"** se habilitará el botón para comparar las versiones.

```
 document.getElementById("compararVersiones").addEventListener("click", function () {
                // Consideramos el ordinal menor de las dos versiones que se van a
                // comparar como la versión original y el ordinal mayor como la
                // versión revisada
                var version1 = document.getElementById("version1").getAttribute("data-version");
                var version2 = document.getElementById("version2").getAttribute("data-version");

                // El nombre del archivo se puede recuperar de cualquiera de las dos versiones,
                // pues el exactamente el mismo.
                var versionFileName = document.getElementById("version1").getAttribute("data-
jsfile");

                var versionOriginal = version1 > version2 ? version2 : version1;
                var versionRevisada = version1 < version2 ? version2 : version1;

                // TODO. Borrar para producción (Dejar en fase de desarrollo)
                console.log("FileName: " + versionFileName);
                console.log("Original: " + versionOriginal);
                console.log("Revisada: " + versionRevisada);

                // Original
                var original = versionFileName + ";" + versionOriginal;
                var revisada = versionFileName + ";" + versionRevisada;

                var objVersiones = {
                    original: original,
                    revisada: revisada
                };

                var objVersionesStringJSON = JSON.stringify(objVersiones);
```

[17] Términos que hacen referencia a arrastrar y soltar.

```
                var xmlhttp = new XMLHttpRequest();

                xmlhttp.onreadystatechange = function () {
                    if (this.readyState === 4 && this.status === 200) {
                        // Se recibe la respuesta en JSON
                        console.log(this.responseText); //TODO. Borrar al final, si se quiere

                        // Convertir a un objeto de javascript el resultado recibido
                        // de la comparacion de las dos versiones
                        var objResults = JSON.parse(this.responseText);

                        var cambios = [];
                        var inserciones = [];
                        var borrados = [];

                        cambios = objResults.modificaciones;
                        inserciones = objResults.inserciones;
                        borrados = objResults.eliminaciones;

                        if (cambios.length > 0) {
                            var filasModificaciones = '';

                            for (i = 0; i < cambios.length; i++) {
                                filasModificaciones += '<tr><td>'
                                var pos = cambios[i].position + 1;
                                filasModificaciones += pos + '</td>';
                                var lines = cambios[i].lines;
                                var lineasModificadas = '';
                                for (j = 0; j < lines.length; j++) {
                                    lineasModificadas += lines[j].toString() + '<br>';
                                }
                                filasModificaciones += '<td>' + lineasModificadas + '</td>';
                                filasModificaciones += '</tr>';
                            }

                            document.getElementById("lineasModificadasTbl").innerHTML =
filasModificaciones;
                        }

                        if (inserciones.length > 0) {
                            var filasInserciones = '';

                            for (i = 0; i < inserciones.length; i++) {
                                filasInserciones += '<tr><td>'
                                var pos = inserciones[i].position + 1;
                                filasInserciones += pos + '</td>';
                                var lines = inserciones[i].lines;
                                var lineasInsertadas = '';
                                for (j = 0; j < lines.length; j++) {
                                    lineasInsertadas += lines[j].toString() + '<br>';
                                }
                                filasInserciones += '<td>' + lineasInsertadas + '</td>';
                                filasInserciones += '</tr>';
                            }

                            document.getElementById("lineasInsertadasTbl").innerHTML =
filasInserciones;
                        }

                        if (borrados.length > 0) {
                            var filasBorradas = '';

                            for (i = 0; i < borrados.length; i++) {
                                filasBorradas += '<tr><td>'
                                var pos = borrados[i].position + 1;
                                filasBorradas += pos + '</td></tr>';
                            }

                            document.getElementById("lineasBorradasTbl").innerHTML =
filasBorradas;
                        }
                    }
```

```
                };


                xmlhttp.open("POST", "ajaxComparadorDeVersiones", true);
                xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                xmlhttp.send("versiones=" + objVersionesStringJSON);
            });
```

En este fragmento del código nos encontramos con el código necesario para, primeramente, atribuir cual es la versión original, es decir, la de menor orden ordinal y la versión revisada. Una vez fijado esto, con el uso del **controlador** *ajaxComparadorDeVersiones.java*, consultamos las diferencias existentes entre las dos versiones que le pasamos al **controlador**:

```
var objVersionesStringJSON = JSON.stringify(objVersiones);
var xmlhttp = new XMLHttpRequest();
```

El **controlador** nos dará información de las líneas que han cambiado, las que se han insertado nuevas y las que han sido borradas. Una vez obtenido esto, hay que rellenar dinámicamente las tablas que se encuentran al final de la página y que muestras las diferencias de las versiones, es decir, si se han borrado, insertado o se han cambiado alguna línea entre las versiones.

```
// Otro manejador de envento click del boton que compara las dos versiones del mismo archivo.
                // Pero ahora lo realizamos para mostrar en pantalla el contenido de las dos
versiones que
                // se estan comparando.

                document.getElementById("compararVersiones").addEventListener("click", function ()
{
                    // Consideramos el ordinal menor de las dos versiones que se van a
                    // comparar como la versión original y el ordinal mayor como la
                    // versión revisada
                    var version1 = document.getElementById("version1").getAttribute("data-
version");
                    var version2 = document.getElementById("version2").getAttribute("data-
version");

                    // El nombre del archivo se puede recuperar de cualquiera de las dos
versiones,
                    // pues el exactamente el mismo.
                    var versionFileName = document.getElementById("version1").getAttribute("data-
jsfile");

                    var versionOriginal = version1 > version2 ? version2 : version1;
                    var versionRevisada = version1 < version2 ? version2 : version1;

                    // TODO. Borrar para producción (Dejar en fase de desarrollo)
                    console.log("FileName: " + versionFileName);
                    console.log("Original: " + versionOriginal);
                    console.log("Revisada: " + versionRevisada);

                    // Original
                    var original = versionFileName + ";" + versionOriginal;
                    var revisada = versionFileName + ";" + versionRevisada;

                    var objVersiones = {
                        original: original,
                        revisada: revisada
                    };

                    var objVersionesStringJSON = JSON.stringify(objVersiones);

                    var xmlhttp = new XMLHttpRequest();

                    xmlhttp.onreadystatechange = function () {
                        if (this.readyState === 4 && this.status === 200) {
                            console.log(this.responseText);
```

```
                            //Convertir el objeto recibido y pintar los datos
                            var objResults = JSON.parse(this.responseText);


                            contenidoVersionOriginal = objResults.original;
                            contenidoVersionRevisada = objResults.revisada;



document.getElementById("if1").contentWindow.setTextoIframe(contenidoVersionOriginal);

document.getElementById("if2").contentWindow.setTextoIframe(contenidoVersionRevisada);
                        }
                    };

                    xmlhttp.open("POST", "ajaxContenidoVersiones", true);
                    xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
                    xmlhttp.send("versiones=" + objVersionesStringJSON);
                });
```

En este fragmento de código representa otro manejador de evento click del botón que compara las dos versiones del mismo archivo, como el fragmento anterior. Pero para este caso se realiza para mostrar en los iframes que hay en la parte inferior de la página web y que nos muestran el contenido de las dos versiones que se están comparando.

Para ello hace uso del **controlador** *AJAXContenidoVersiones.java*, el cual, proporciona el contenido de los archivos.

```
    /* Para la encriptación SHA512 */
            function sha512(str) {
                // We transform the string into an arraybuffer.
                var buffer = new TextEncoder("utf-8").encode(str);
                return crypto.subtle.digest("SHA-512", buffer).then(function (hash) {
                    return hex(hash);
                });
            }

            function hex(buffer) {
                var hexCodes = [];
                var view = new DataView(buffer);
                for (var i = 0; i < view.byteLength; i += 4) {
                    var value = view.getUint32(i);

                    var stringValue = value.toString(16);
                    var padding = '00000000';
                    var paddedValue = (padding + stringValue).slice(-padding.length);
                    hexCodes.push(paddedValue);
                }

                // Join all the hex strings into one
                return hexCodes.join("");
            }

        </script>
        <script src="js/jquery1.12.4.js" type="text/javascript"></script>
        <script src="js/bootstrap.js" type="text/javascript"></script>
```

Finalmente, para este último fragmento del código tenemos el código necesario para el cálculo del hash sha512 a nivel del cliente. Y las dos últimas líneas:

```
<script src="js/jquery1.12.4.js" type="text/javascript"></script>
<script src="js/bootstrap.js" type="text/javascript"></script>
```

Son necesarias para el uso de Bootstrap 3, utilizado en la vista principal.

4.3 versionOriginal.html

```
    <body>
      <pre id="ContenidoVersionOriginal">
      </pre>

      <script>
          function setTextoIframe(texto){
              document.getElementById("ContenidoVersionOriginal").innerHTML = texto;

          };
      </script>
  </body>
```

Se trata de una vista auxiliar utilizada para poder mostrar el contenido de la versión original en el iframe (es un elemento HTML que permite insertar o incrustar un documento HTML dentro de un documento HTML principal) acondicionado para ello. Se utiliza un iframe para mostrar el contenido de la versión para que no pueda interactuar con la pagina web, de tal forma que, si se hubiese guardado un archivo malicioso en la base de datos, no afectaría a la web.

4.4 versionRevisada.html

```
  <body>
      <pre id="ContenidoVersionRevisada">
      </pre>

      <script>
          function setTextoIframe(texto){
              document.getElementById("ContenidoVersionRevisada").innerHTML = texto;

          };
      </script>
  </body>
```

Se trata de una vista auxiliar utilizada para poder mostrar el contenido de la versión revisada en el iframe acondicionado para ello. Se utiliza un iframe para mostrar el contenido de la versión para que no pueda interactuar con la pagina web, de tal forma que, si se hubiese guardado un archivo malicioso en la base de datos, no afectaría a la web.

5 MODELO

5.1 DataBaseConexion.java

Esta se trata de una **clase** extensa en cuanto al código se refiere, por lo que se va a describir su comportamiento separándolo en diferentes fragmentos respecto del total del código.

```
public class DataBaseConexion {
    private Connection conn;
    private final String usuario;
    private final String password;

    public DataBaseConexion(String usuario, String password) {
        this.usuario = usuario;
        this.password = password;
    }


    // Metodo que realiza la conexion a la base de datos.
    public Connection getConexion() throws ClassNotFoundException, SQLException {
        Properties propiedades;
        propiedades = new Properties();

        propiedades.put("user", this.usuario);
        propiedades.put("password", this.password);

        conn = null;

        String urlConection;    // Conexion a MySQL
        urlConection = "jdbc:mysql://localhost:3306/reto_ciberseguridad_big_data";

        try {
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(urlConection, propiedades);
        } catch (SQLException ex) {
            Logger.getLogger(DataBaseConexion.class.getName()).log(Level.SEVERE, null, ex);
        }

        return conn;
    }
```

En este primer fragmento se observa contenido el constructor por defecto de la **clase** *DataBaseConexion.java* y el método *getConexion*. Este se encarga de establecer la conexión con la base de datos, siendo necesario incluir el driver de esta. Como nuestra base de datos es *mysql*, nuestro driver será, por tanto, el siguiente:

"*jdbc:mysql://localhost:3306/reto_ciberseguridad_big_data"*

El constructor, por su parte, se encarga de inicializar los valores de las variables *usuario* y *password*, que son las credenciales pertinentes que nos proporcionarían el acceso a la base de datos.

Aunque en esta **clase** el método principal es *getConexion*, existen otra serie de métodos adicionales:

```
public ResultSet getFileName(String fileName) throws SQLException {
        ResultSet rs;
        rs = null;
```

```
        String query;
        query = "select nombre from jsfiles where nombre = ? ";

        PreparedStatement stmt = null;

        try {
            conn = getConexion();

            stmt = conn.prepareStatement(query);
            stmt.setString(1, fileName);
            rs = stmt.executeQuery();

        } catch (ClassNotFoundException ex) {
            Logger.getLogger(DataBaseConexion.class.getName()).log(Level.SEVERE, null, ex);
        }

        return rs;
    }
```

El método *getFileName*, que recibe como parámetro de entrada un *String* y devuelve un objeto de tipo *ResultSet*. Como se puede apreciar en el código, este método realiza una *query*[18] pasando el nombre del archivo que recibe el método como parámetro de entrada y devolviendo el archivo como parámetro de salida del método.

```
public Long setNewJSFileRecord(JSFile jsFile, JSFileData jsFileData,
            List<JSFileHashMethod> jsFileHashMethods) throws SQLException, ClassNotFoundException {

        // Primero insertar en la tabla jsfiles el nombre del archivo recibido
        // y recuperar el id generado.
        String queryInsert1;
        queryInsert1 = "insert into jsfiles (nombre) values (?)";

        Long lastInsertedId = 0L;

        PreparedStatement stmt1 = null;

        try {
            conn = getConexion();
            conn.setAutoCommit(false);

            // Por si hay algun fallo deshacer todo desde aqui
            Savepoint save1 = conn.setSavepoint();

            // Para recuperar el last inserted id
            stmt1 = conn.prepareStatement(queryInsert1, Statement.RETURN_GENERATED_KEYS);

            stmt1.setString(1, jsFile.getName());
            int affectedRows = stmt1.executeUpdate();

            if(affectedRows == 0) {
                throw new SQLException("Fallo al insertar registro en la tabla jsfiles");
            } else {
                ResultSet generatedKeys = stmt1.getGeneratedKeys();
                if(generatedKeys.next()) {
                    lastInsertedId = generatedKeys.getLong(1);
                }
            }

            // Insertar en la tabla jsfiledata con el id recuperado de la tabla jsfiles
            int version = 1; // Este método se invoca cuando el archivo se recibe por primera vez

            String queryInsert2;
            queryInsert2 = "insert into jsfiledata (id, version, hash, fileSize, "
                    + "     fechaSubida, ipCliente, "
                    + "     hostNameCliente, fileContent ) values (?,?,?,?,?,?,?,?)";

            PreparedStatement stmt2 = null;
```

[18] Consulta a la base de datos.

```
            stmt2 = conn.prepareStatement(queryInsert2);
            stmt2.setLong(1, lastInsertedId);
            stmt2.setInt(2, version);
            stmt2.setString(3, jsFileData.getHash());
            stmt2.setString(4, jsFileData.getFileSize());
            stmt2.setDate(5, Date.valueOf(jsFileData.getFechaSubida()));
            stmt2.setString(6, jsFileData.getIpCliente());
            stmt2.setString(7, jsFileData.getHostNameCliente());
            stmt2.setString(8, jsFileData.getFileContentString());

            affectedRows = stmt2.executeUpdate();

            // Insertar registro en la tabla jsfilehashmethods
            String queryInsert3;
            queryInsert3 = "insert into jsfilehashmethods (id, version, nombre,"
                    + " hash, contenido) values (?,?,?,?,?)";


            for(JSFileHashMethod metodo: jsFileHashMethods) {
                PreparedStatement stmt3 = null;

                stmt3 = conn.prepareStatement(queryInsert3);
                stmt3.setLong(1, lastInsertedId);
                stmt3.setInt(2, version);
                stmt3.setString(3, metodo.getNombre());
                stmt3.setString(4, metodo.getHash());
                stmt3.setString(5, metodo.getContenido());

                affectedRows = stmt3.executeUpdate();
            }

            conn.commit();
        } catch (SQLException ex) {
            System.out.println(ex.getMessage());
        } finally {
            conn.setAutoCommit(true);
            stmt1.close();
        }

        return lastInsertedId;
    }
```

El método *setJSFileRecord*, que recibe como parámetros un objeto de tipo *JsFile*, *JSFileData* y *List<JSFileHashMethods>* y devuelve un L*ong*. Este método se encarga principalmente de insertar un nuevo archivo **JavaScript** en la base de datos.

Estos objetos que se reciben como parámetros de entrada son instancias de **clases** que se verán detalladamente en los siguientes apartados y que facilitarán comprender cómo funciona el método.

En continuación con el código, el método inserta, en primer lugar, en la tabla **jsfiles** el nombre del archivo recibido mediante "queryInsert1" y recupera el *id* generado con "smt1". En segundo lugar, inserta en la tabla **jsfiledata** la "queryInsert2" a partir del *id* obtenido anteriormente en la tabla **jsfiles**. Una vez rellenadas estas dos tablas, solo faltaría por rellenar la tabla **jsfilehashmethods** mediante la "quetyInsert3". Por último, se recibe como parámetro de salida del método el *id* del último archivo seleccionado.

Cabría destacar que, al tratar con bases de datos, en el código se ha tenido en cuenta que varios usuarios puedan subir archivos desde diferentes dispositivos compartiendo una misma base de datos. De esta forma es necesario establecer un S*avepoint*[19] para evitar que si se

[19] Punto de recuperación de información.

produce algún fallo a mitad de ejecución no se produzca ningún conflicto en la base de datos. Y, por último, se añade un C*ommit*[20] que nos permita guardar los cambios al final del método.

```
 public Boolean existeArchivoSeleccionado(ArchivoSeleccionado as) throws ClassNotFoundException,
SQLException {
        Boolean existe = Boolean.TRUE;

        String query =  "select id, hash \n" +
                        "from jsfiledata \n" +
                        "where id = (select id from jsfiles where nombre = ?) and hash = ?";
        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
        stmt.setString(1, as.getNombre());
        stmt.setString(2, as.getHash());
        ResultSet rs = stmt.executeQuery();
        if(!rs.next()) {
            existe = Boolean.FALSE;
        }

        return existe;
    }
```

El método *existeArchivoSeleccionado*, que recibe como parámetro un objeto de la **clase** ArchivoSeleccionado y devuelve un booleano, se encarga de realizar una consulta en la base de datos y devuelve un booleano dependiendo de si el archivo pasado como parámetro de entrada existe o no.

La *query*, como se puede apreciar en el código, se aplica tanto al nombre como al **hash** del archivo en cuestión. El *nombre* y el **hash** se obtienen a través de los métodos propios de la **clase** *ArchivoSeleccionado.*

```
public int getId(String nombreArchivo) throws ClassNotFoundException, SQLException {
        int id = 0;
        String query = "select id from jsfiles where nombre = ?";

        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
        stmt.setString(1, nombreArchivo);
        ResultSet rs = stmt.executeQuery();
        if(rs.next()) {
            id = rs.getInt("id");
        }
        return id;
    }
```

El método *getId* recibe como parámetro de entrada un *String* y devuelve un entero. Este método se encarga de obtener el *id* haciendo la correspondiente consulta en la base de datos. Para ello, es necesario el nombre del archivo que recibe como parámetro de entrada.

```
  public int getCurrentVersion(int id) throws ClassNotFoundException, SQLException {
        int currentVersion = 0;
        String query = "select max(version) as currentVersion from jsfiledata where id = ?";

        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
```

[20] Confirmación de un conjunto de cambios.

```
        stmt.setInt(1, id);
        ResultSet rs = stmt.executeQuery();
        if(rs.next()) {
            currentVersion = rs.getInt("currentVersion");
        }

        return currentVersion;
    }
```

El método *getCurrentVersion* recibe y devuelve un entero. Este método se encarga de obtener la versión actual del archivo correspondiente. Para ello, realiza una consulta en la base de datos a partir del *id* que recibe como parámetro de entrada y devuelve un entero con el valor de la versión actual.

```
public void setJSFileData(JSFileData jsFileData) throws ClassNotFoundException, SQLException {
            String query;
            query = "insert into jsfiledata (id, version, hash, fileSize, "
                    + "     fechaSubida, ipCliente, "
                    + "     hostNameCliente, fileContent ) values (?,?,?,?,?,?,?,?)";

            PreparedStatement stmt = null;

            conn = getConexion();

            stmt = conn.prepareStatement(query);
            stmt.setLong(1, jsFileData.getId());
            stmt.setInt(2, jsFileData.getVersion());
            stmt.setString(3, jsFileData.getHash());
            stmt.setString(4, jsFileData.getFileSize());
            stmt.setDate(5, Date.valueOf(jsFileData.getFechaSubida()));
            stmt.setString(6, jsFileData.getIpCliente());
            stmt.setString(7, jsFileData.getHostNameCliente());
            stmt.setString(8, jsFileData.getFileContentString());

            int affectedRows = stmt.executeUpdate();

    }
```

El método *setFileData*, que recibe como parámetro de entrada un objeto de tipo *JSFileData* y no devuelve nada. Este método se encarga de insertar una nueva versión del archivo seleccionado a través de una consulta a la base de datos. Esta *query* se aplica a varios parámetros como *id*, *versión*, **hash** y otros, obtenidos a través de los propios métodos de la **clase** jsFileData. Por último, se ejecuta un *executeUpdate* que permita establecer el nuevo *FileData*.

```
public      void      setJsFileHashMethods(List<JSFileHashMethod>      jsFileHashMethods)      throws
ClassNotFoundException, SQLException {
            // Insertar registro en la tabla jsfilehashmethods
            int affectedRows = 0;

            String queryInsert4;
            queryInsert4 = "insert into jsfilehashmethods (id, version, nombre,"
                    + " hash, contenido) values (?,?,?,?,?)";

            PreparedStatement stmt4;
            conn = getConexion();

            for(JSFileHashMethod metodo: jsFileHashMethods) {
                stmt4 = null;

                stmt4 = conn.prepareStatement(queryInsert4);
                stmt4.setLong(1, metodo.getId());
                stmt4.setInt(2, metodo.getVersion());
                stmt4.setString(3, metodo.getNombre());
                stmt4.setString(4, metodo.getHash());
```

```
                stmt4.setString(5, metodo.getContenido());

                affectedRows = stmt4.executeUpdate();
            }
```

El método *setJSFileHashMethods* recibe como parámetro de entrada una lista de objetos de tipo *JSFileHashMethod* (*List<JSFileHashMethod>*) y no devuelve nada. Este método se encarga de introducir los registros en la tabla ***jsfilehashmethods*** de la base de datos y, para ello, se ejecuta una consulta de los parámetros *id*, *versión*, **hash** y otros.

```
  // Recupera los registros de la tabla jsfiles
    public ResultSet getRegistrosJsFiles() throws SQLException {
        ResultSet rs;
        rs = null;

        String query;
        query = "select * from jsfiles";

        PreparedStatement stmt = null;

        try {
            conn = getConexion();

            stmt = conn.prepareStatement(query);

            rs = stmt.executeQuery();

        } catch (ClassNotFoundException ex) {
            Logger.getLogger(DataBaseConexion.class.getName()).log(Level.SEVERE, null, ex);
        }

        return rs;
    }
```

El método *getRegistrosJsFiles* no recibe nada como parámetro de entrada y devuelve un objeto de tipo *ResultSet*. Este método se encarga de obtener todos los registros de la tabla **jsfiles** de la base de datos y, para ello, se ejecuta un “select * from jsfiles”.

```
    // Recupera version y fecha de subida del archivo
     public ResultSet getVersionFechaSubida(String id) throws ClassNotFoundException, SQLException
{
        String query = "select version, fechaSubida, hash\n"
                + "from jsfiledata\n"
                + "where id = ? order by version desc";

        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
        stmt.setString(1, id);

        ResultSet rs = stmt.executeQuery();

        return rs;
    }
```

El método *getVersionFechaSubida* recibe un entero como parámetro de entrada y devuelve un objeto de tipo *ResultSet*. Este método se encarga de recuperar la versión, la fecha de subida y el hash del archivo y, para ello, se ejecuta un *select* en la tabla **jsfiledata** para el id introducido como parámetro de entrada del método.

```
// Recupera el id de un archivo dado el nombre, desde la tabla
    // jsfiles
    public int getIdFromJSFile(String fileName) throws SQLException, ClassNotFoundException {
        String query = "select id from jsfiles where nombre = ?";

        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
        stmt.setString(1, fileName);

        ResultSet rs = stmt.executeQuery();

        return rs.next() ? rs.getInt("id") : 0;
    }
```

El método *getIdFromJSFile* recibe como parámetro de entrada un string correspondiente al nombre del archivo y devuelve un entero correspondiente al id. Este método se encarga de recuperar el id del archivo dado el nombre desde la tabla **jsfiles** de la base de datos y, para ello, se realiza un *select* a partir del nombre recibido como parámetro de entrada al método.

```
// Recupera el contenido de un archivo (fileContent) de la tabla jsfiledata
    // entrando con id y version
    public String getFileContent(int id, int version) throws SQLException, ClassNotFoundException
{
        String query = "select fileContent from jsfiledata where id = ? and version = ?";

        PreparedStatement stmt = null;
        conn = getConexion();
        stmt = conn.prepareStatement(query);
        stmt.setInt(1, id);
        stmt.setInt(2, version);

        ResultSet rs = stmt.executeQuery();

        return rs.next() ? rs.getString("fileContent") : "";
    }
```

El método *getFileContent* recibe como parámetros de entrada dos enteros correspondientes al id y la versión del archivo y devuelve un string correspondiente al contenido del archivo. Este método se encarga de recuperar el contenido de un archivo de la tabla **jsfiledata** de la base de datos y, para ello, se realiza un *select* dado el id y la versión del archivo obtenidos como parámetros de entrada al método.

5.2 JSFile.java

```
public class JSFile {

    private final String name;
    private final String id;

    public String getId() {
        return id;
    }

    public JSFile(String id, String name) {
        this.id = id;
        this.name = name;

    }
```

```
    public String getName() {
        return name;
    }
```

La **clase** *JsFile.java* está compuesta solamente de un constructor por defecto y los métodos *getName y getId*. *JsFile.java*. Contiene únicamente los métodos get ya que queremos que los atributos id y name sean final por lo que los métodos set no tendrían sentido.

5.3 JSFileData.java

```
public class JSFileData {
    private int id;
    private int version;
    private String hash;
    private String fileSize;
    private final String fileContentString;
    private LocalDate fechaSubida;
    private String ipCliente;
    private String hostNameCliente;

    public JSFileData(int id, int version, String hash, String fileSize, String fileContentString,
            LocalDate fechaSubida, String ipCliente, String hostNameCliente) {
        this.id = id;
        this.version = version;
        this.hash = hash;
        this.fileSize = fileSize;
        this.fileContentString = fileContentString;
        this.fechaSubida = fechaSubida;
        this.ipCliente = ipCliente;
        this.hostNameCliente = hostNameCliente;
    }

    public int getId() {
        return id;
    }

    public String getFileContentString() {
        return fileContentString;
    }

    public void setId(int id) {
        this.id = id;
    }

    public int getVersion() {
        return version;
    }

    public void setVersion(int version) {
        this.version = version;
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public String getFileSize() {
        return fileSize;
    }

    public void setFileSize(String fileSize) {
        this.fileSize = fileSize;
    }

    public LocalDate getFechaSubida() {
```

```
        return fechaSubida;
    }

    public void setFechaSubida(LocalDate fechaSubida) {
        this.fechaSubida = fechaSubida;
    }

    public String getIpCliente() {
        return ipCliente;
    }

    public void setIpCliente(String ipCliente) {
        this.ipCliente = ipCliente;
    }

    public String getHostNameCliente() {
        return hostNameCliente;
    }

    public void setHostNameCliente(String hostNameCliente) {
        this.hostNameCliente = hostNameCliente;
    }

}
```

La **clase** *JSFileData* contiene un constructor por defecto y los siguientes métodos ***getters*** y ***setters***:

- *getId, setId*
- *getFileContentString*
- *getVersion, setVersion*
- *getHash, setHash*
- *getFileSize, setFileSize*
- *getFechaSubida, setFechaSubida*
- *getIpCliente, setIpCliente*
- *getHostNameCliente, setHostNameCliente*

Cabe destacar que no está contemplado el método *setFileContentString* ya que la propiedad fileContentString es final.

5.4 JSFileHashMethod.java

```
public class JSFileHashMethod {
    private final int id;
    private final int version;
    private final String nombre;
    private final String hash;
    private final String contenido;

    public JSFileHashMethod(int id, int version, String nombre, String hash, String contenido) {
        this.id = id;
        this.version = version;
        this.nombre = nombre;
        this.hash = hash;
        this.contenido = contenido;
    }

    public int getId() {
        return id;
    }

    public int getVersion() {
```

```
        return version;
    }

    public String getNombre() {
        return nombre;
    }

    public String getHash() {
        return hash;
    }

    public String getContenido() {
        return contenido;
    }

}
```

La **clase** *JSFileHashMethod* contiene un constructor por defecto y los siguientes métodos **getters**:

- *getId*
- *getVersion*
- *getNombre*
- *getHash*
- *getContenido*

La **clase** *JSFileHashMethod*, al igual que pasaba con la **clase** anterior, no posee los correspondientes métodos **setters** ya que las propiedades están establecidas como final porque no queremos modificar los archivos una vez subidos.

5.5 Modelo.java

En este apartado se analiza el código por partes al ser más extenso.

La **clase** *Modelo.java* como su propio nombre indica es el **modelo** del *patrón MVC* y posee los, en resumen, los siguientes métodos principales:

```
public class Modelo {
    private JSFile jsFile;
    private JSFileData jsFileData;
    private List<JSFileHashMethod> jsFileHashMethods;

    public void setJsFile(JSFile jsFile) {
        this.jsFile = jsFile;
    }

    public List<JSFile> getJsFile() throws SQLException {
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");

        List<JSFile> listadoFicheros;

        listadoFicheros = new ArrayList<>();

        ResultSet rs = dbconn.getRegistrosJsFiles();

        while(rs.next()) {
            listadoFicheros.add(new JSFile(rs.getString("id"), rs.getString("nombre")));
    }
```

```
        return listadoFicheros;
    }
    public JSFileData getJsFileData() {
        return jsFileData;
    }

    public void setJsFileData(JSFileData jsFileData) {
        this.jsFileData = jsFileData;
    }

    public List<JSFileHashMethod> getJsFileHashMethods() {
        return jsFileHashMethods;
    }

    public void setJsFileHashMethods(List<JSFileHashMethod> jsFileHashMethods) throws
ClassNotFoundException, SQLException {
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        dbconn.setJsFileHashMethods(jsFileHashMethods);
    }
```

- *setJsFile, getJsFile*
- *getJsFileData*
- *setJsFileData*
- *getJsFileHashMethods, setJsFileHashMethods*

Estos son bastante descriptivos según su nombre, siendo *setFileHashMethods* el único que habría que destacar por comportamiento. Este realiza una conexión a la base de datos introduciendo el usuario y contraseña para posteriormente rellenar la tabla **jsfilehashmethods**.

Además, esta **clase** Modelo.java posee otra serie de métodos auxiliares como:

```
public Boolean jsFileExists(String jsFileName) throws SQLException {
        Boolean existe = Boolean.TRUE;
        DataBaseConexion dbconn;
        ResultSet rs;

        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        rs = dbconn.getFileName(jsFileName);

        if(!rs.next()) {
            existe = Boolean.FALSE;
        }

        return existe;
    }
```

El método *jsFileExists* recibe como parámetro de entrada un String y devuelve un Boolean. Este método se encarga de averiguar si el correspondiente nombre de archivo pasado como parámetro de entrada existe o no devolviendo un booleano en base a ello. Para ello, el método *jsFileExists* realiza una conexión a la base de datos con las correspondientes credenciales, comprobando si efectivamente existe.

```
 public Long setNewJSFileRecord(JSFile jsFile, JSFileData jsFileData,
            List<JSFileHashMethod> jsFileHashMethods ) throws SQLException, ClassNotFoundException
{
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        return dbconn.setNewJSFileRecord(jsFile, jsFileData, jsFileHashMethods);
```

```
    }
```

El método *setNewJSFileRecord* que recibe como parámetro de entrada un objeto de tipo *JSFile*, un objeto de tipo *JSFileData* y una lista de objetos de tipo *JSFIleHashMethod* (*List<JSFileHashMethod>*) y devuelve un L*ong*. Este método se encarga de insertar un registro del archivo recibido en cada una de las tres tablas de la base de datos, devolviendo el *id* del archivo. Para ello, el método *setNewJSFileRecord* realiza una conexión a la base de datos con las correspondientes credenciales y pasa los argumentos de entrada.

```
    public int getIdArchivoJSExistente(String nombreArchivoJS) throws ClassNotFoundException,
SQLException {
        int id = 0;
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        id = dbconn.getId(nombreArchivoJS);
        return id;
    }
```

El método *gerIdArchivoJSExistente* que recibe como parámetro de entrada un String y devuelve un entero. Este método se encarga de recuperar el *id* del *nombre* del archivo recibido. Para ello el método *getIdArchivoExistente* realiza una conexión a la base de datos con las credenciales correspondientes y pasa el nombre del archivo JS que recibe como parámetro de entrada.

```
public int getNextVersion(int id) throws ClassNotFoundException, SQLException {
        int nextVersion = 0;
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        nextVersion = dbconn.getCurrentVersion(id);

        return ++nextVersion;
    }
```

El método *getNextVersion* recibe y devuelve un entero. Este método se encarga de recuperar la versión más alta para un *id* en concreto, esto nos sirve para poder insertar una versión posterior del mismo archivo cada vez. Para ello el método *getNextVersion* realiza una conexión a la base de datos con las credenciales correspondientes y pasa el *id* recibido como parámetro de entrada.

```
    public void setNuevaVersion(JSFileData jsfiledata) throws ClassNotFoundException,
SQLException {
        DataBaseConexion dbconn;
        dbconn = new DataBaseConexion("galveniano", "Temp2018$$");
        dbconn.setJSFileData(jsfiledata);
    }
```

El método *setNuevaVersion* recibe como parámetro de entrada un objeto de tipo *JSFileData* y no devuelve nada. Este método nos sirve para insertar una nueva versión del archivo recibido en la tabla **jsfiledata**. Para ello, el método *setNuevaVersion* realiza una conexión a la base de datos con las credenciales correspondientes y se establece una nueva versión pasándole el parámetro de entrada.

6 CONTROLADORES

A continuación, se detalla el funcionamiento del código según cada una de las **clases** y **servlets** citados anteriormente.

Cabe destacar que no está incluido el código completo, sino el más relevante.

6.1 AJAXManager.java

```
@WebServlet(name = "AJAXManager", urlPatterns = {"/ajaxmanager"})
public class AJAXManager extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, ClassNotFoundException, SQLException {
        response.setContentType("Content-Type: application/json; charset=UTF-8");

        String existe = "no existe";

        Gson gson = new Gson();

        ArchivoSeleccionado archivoSeleccionado =        gson.fromJson(request.getParameter("q"),
ArchivoSeleccionado.class);

        DataBaseConexion db;

        db = new DataBaseConexion("galveniano", "Temp2018$$");

        if(db.existeArchivoSeleccionado(archivoSeleccionado)) {
            existe = "existe";
        }

        try (PrintWriter out = response.getWriter()) {

            out.print(existe);
        }
    }
}
```

AJAXManager.java es un **servlet** que se encarga de comprobar si el archivo **JavaScript** existe o no en la base de datos y responde a la **vista** *index.jsp* habilitando el botón "subir archivo" o no, sugiriendo al usuario que elija otro archivo diferente en caso de ya existir este en la base de datos. Este es necesario ya que nos interesa subir archivos que o bien no existen, o si existen, tengan un **hash** diferente y, por tanto, sea una nueva versión de este.

Para realizar esta tarea inicializamos la variable "existe" a "no existe" y creamos un objeto **Gson** a partir del parámetro enviado por **Json** desde el *index.jsp*. Creamos un objeto de tipo *ArchivoSeleccionado* pasándole el archivo subido mediante el método *getParameter*. Una vez realizado esto, creamos una conexión a la base de datos y a través del método *existeArchivoSeleccionado* de la **clase** *DataBaseConexion.java* comprobamos si existe o no. Por último, enviamos el resultado a la **vista** *index.jsp* mediante la instrucción "out.print (existe)".

Cabe destacar que no se ha utilizado el método tradicional para la recepción y envío de información a los **servlet** ("*doPost*"), ya que utilizando los objetos *request* y *response* no es necesario especificarle el método de recepción de datos.

6.2 UploadManager.java

En este apartado se analiza el código por partes ya que este es más extenso.

UploadManager.java es el **controlador** que se encarga principalmente de comprobar si el archivo ha sido modificado y almacenarlo en la base de datos si este no ha sufrido cambios.

```
@WebServlet(name = "UploadManager", urlPatterns = {"/uploadmanager"})
@MultipartConfig
public class UploadManager extends HttpServlet {


    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, SQLException, ClassNotFoundException {
        response.setContentType("text/html;charset=UTF-8");

        // Instanciar el modelo de datos
        Modelo m;
        m = new Modelo();

        // Hash recibido, calculado mediante javascript
        String hashRecibido;

        hashRecibido = request.getParameter("hashFile");

        // Hash calculado en el servidor (java) para el archivo recibido
        String hashCalculado = "";


        // Recuperar el archivo JS recibido desde el formulario web del cliente
        Part filePart;

        // Para obtener el nombre del fichero recibido, en el caso de haber sido
        // desde el navegador Microsoft Internet Explorer. Este navegador, de
        // forma incorrecta, envia la ruta completa del archivo conjuntamente
        // con el nombre, en lugar de solamente enviar el nombre del archivo
        String fileName;

        // Recuperar el contenido del fichero
        InputStream fileContent;

        // Tamaño en bytes del archivo recibido
        String fileSize;
        fileSize = request.getParameter("fileSize");

        String clientHostName;
        String clientIP;

        int currentID = 1;
        int currentVersion = 1;

        clientIP = request.getRemoteAddr();
        clientHostName = request.getRemoteHost();

        filePart = request.getPart("file");
        fileName = Paths.get(filePart.getSubmittedFileName()).getFileName().toString();
        fileContent = filePart.getInputStream();
        // Para posicionar el puntero al inicio del archivo recibido
        fileContent.close();

        fileContent = filePart.getInputStream();

        try {
            MessageDigest messageDigest;
            messageDigest = MessageDigest.getInstance("SHA-512");
```

```
            // leer el fichero, cuyo contenido esta almacenado en
            // la variable fileContent, byte a byte
            byte[] buffer = new byte[1];
            int fin_archivo = -1;
            int caracter;

            caracter = fileContent.read(buffer);

            while (caracter != fin_archivo) {
                // Pasa texto claro (caracter) a la función resumen
                messageDigest.update(buffer);
                caracter = fileContent.read(buffer);
            }

            // Generamos el resumen MD5, es decir, el hash
            byte[] resumen = messageDigest.digest();

            // Pasamos el resumen a hexadecimal
            String s = "";
            int totalResumen;

            totalResumen = resumen.length;

            for (int i = 0; i < totalResumen; i++) {
                s += Integer.toHexString((resumen[i] >> 4) & 0xf);
                s += Integer.toHexString((resumen[i]) & 0xf);
            }

            hashCalculado = s;

        } catch (NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
        }
```

De acuerdo con el código en este primer fragmento se inicializa el modelo de datos y las variables: *hashRecibido* (obtenido por **JavaScript**), *fileSize*, *clientIP* y *clientHostName*. A continuación, se crea una instancia *filePart* que nos va a permitir trabajar el fichero por partes. Este objeto *filePart* lee el archivo byte a byte para calcular su correspondiente **hash**. Por último, habría que convertir a hexadecimal el valor del **hash**, que se guarda en la variable *hashCalculado* (obtenido en el **servlet** mediante **Java**).

```
// Convertir el archivo recibido a String
        // para guardarlo en un campo de la tabla jsfiledata
        // en la base de datos
        fileContent.close();
        fileContent = filePart.getInputStream();


        String fileContentString;
        fileContentString = new BufferedReader(new InputStreamReader(fileContent))
                .lines().collect(Collectors.joining("\n"));

        // Compara el hash calculado en Java con el recibido, calculado
        // mediante javascript, si son iguales comprobar si el archivo recibido
        // ya esta registrado en la tabla correspondiente de la base de datos.

        if (!hashRecibido.equals(hashCalculado)) {
            // ALERTA DE SEGURIDAD!!!! No son iguales el hash calculado y
            // el recibido
            request.getRequestDispatcher("archivoJSHackeado.jsp").forward(request, response);
        } else {
            /* Guardamos el archivo en la Base de Datos, en sus tablas
             correspondientes y regresamos a la pagina principal */

            /* Extraer funciones y métodos del archivo recibido,
            calcularles el hash y insertar un registro en la tabla
            jsfilehashmethods por cada función/método, asociado al id
            y a la versión en curso */

            List<String> metodos; // Array de funciones y métodos
```

```
String REGEX = "function"; // Patrón o expresion regular a buscar

String INPUT = fileContentString; // Archivo recibido, convertido a String

Pattern p = Pattern.compile(REGEX);

// Array del contenido del archivo recibido, separado por el token
// function
String[] itemsArray = p.split(INPUT);

// Eliminar del array anterior los elementos que no contengan el
// caracter {
List<String> itemsList = Arrays.asList(itemsArray);

// Listado que solo contiene los metodos del archivo recibido
metodos = itemsList.stream()
        .filter(i -> i.contains("{"))
        .collect(Collectors.toList());

// Creamos el listado de JSFileHashMethod
List<JSFileHashMethod> listadoDeMetodos;


Pattern patron = Pattern.compile("\\{");

// Comprobar si el nombre del archivo recibido ya esta registrado
// en la tabla jsfiles
if (!m.jsFileExists(fileName)) {


    listadoDeMetodos = new ArrayList<>();

    // Recorremos la lista de metodos para obtener
    // listadoDeMetodos
    for (String s : metodos) {
        JSFileHashMethod j;
        String nombre = null;
        String hash = null;
        String contenido = null;

        // Obtener el nombre del metodo
        // En realidad hemos obtenido la declaracion, es decir,
        // el nombre y los parámetros formales que recibe
        String[] temp;
        temp = patron.split(s);
        nombre = temp[0].trim();

        // Calcular el hash del metodo
        hash = Hash.sha512(s);

        // Metodo completo, declaracion y cuerpo.
        contenido = s;

        j = new JSFileHashMethod(0, 0, nombre, hash, contenido);
        listadoDeMetodos.add(j);
    }

    currentID = (int) (long) m.setNewJSFileRecord(new JSFile(fileName),
            new JSFileData(0, 0, hashCalculado, fileSize,
                    fileContentString, LocalDate.now(),
                    clientIP, clientHostName),
            listadoDeMetodos);

    currentVersion = 1; // Porque es la primera vez que se recibe el
    // archivo


} else {
    // Si ya esta registrado en la tabla jsfiles, recuperar el id
    // y comprobar si el hash esta en la tabla jsfiledata, de ser
    // asi no guardamos el archivo porque se trata de una version
    // del mismo archivo JS que se ha intentado subir.
    // Esta comprobacion ya esta implementada a traves de AJAX,
    // es decir, que no tenemos que hacer nada al respecto porque si
    // se activa el boton de enviar es porque el archivo con su hash
    // correspondiente no existe en la tabla jsfiledata.
    // Por tanto, lo que hay que hacer en este momento es insertar
    // el archivo recibido como una nueva version.
```

```
                // Recuperar el id del nombre del archivo recibido, en la tabla
                // jsfiles
                int id = m.getIdArchivoJSExistente(fileName);
                currentID = id;

                // Recuperar la version con la cual será registrado el archivo
                int version = m.getNextVersion(id);
                currentVersion = version;

                // Registrar la nueva version
                m.setNuevaVersion(new JSFileData(id, version, hashCalculado,
                        fileSize, fileContentString, LocalDate.now(), clientIP, clientIP));

                listadoDeMetodos = new ArrayList<>();

                // Recorremos la lista de metodos para obtener
                // listadoDeMetodos
                for (String s : metodos) {
                    JSFileHashMethod j;
                    String nombre = null;
                    String hash = null;
                    String contenido = null;

                    // Obtener el nombre del metodo
                    // En realidad hemos obtenido la declaracion, es decir,
                    // el nombre y los parámetros formales que recibe
                    String[] temp;
                    temp = patron.split(s);
                    nombre = temp[0].trim();

                    // Calcular el hash del metodo
                    hash = Hash.sha512(s);

                    // Metodo completo, declaracion y cuerpo.
                    contenido = s;

                    j = new JSFileHashMethod(currentID, currentVersion, nombre, hash, contenido);
                    listadoDeMetodos.add(j);
                }

                // Registramos los metodos en la tabla jsfilehashmethods
                m.setJsFileHashMethods(listadoDeMetodos);


            }

            request.getRequestDispatcher("index.jsp").forward(request, response);
        }

    }
}
```

Esta parte del código es la más extensa y complicada del **servlet**, por lo que se comenta la idea general. El código que se muestra va bien documentado para que sea posible ir entendiendo el código a la vez que se va leyendo.

A continuación, lo primero de todo es rellenar la variable *fileContent*, que posee el contenido del fichero para más adelante registrarlo en la base de datos. Seguidamente hacemos la comprobación más importante del código, que es ver si coincide el *hashRecibido* con el *hashCalculado*. De esta forma comprobamos si el archivo ha sido modificado y, en caso de serlo, redirigimos hacia la vista archivo modificado: *ArchivoJSHackeado.jsp*.

Una vez realizada esta comprobación, sería necesario comprobar si el archivo está ya registrado y verificar si se trata de una nueva versión. En todo caso, sea o no una nueva versión, para poder registrar el *nombre*, *hash* y *contenido* de los métodos de los archivos

JavaScript debemos crear un patrón con una lista de métodos para posteriormente recorrerla y calcular su *hash*. La única diferencia existente entre el caso de que sea un archivo subido por primera vez y una nueva versión es que será necesario obtener previamente la *currentVersion*.

6.3 AJAXFileSelected.java

```
@WebServlet(name = "AJAXFileSelected", urlPatterns = {"/ajaxfileselected"})
public class AJAXFileSelected extends HttpServlet {


    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, ClassNotFoundException, SQLException {
        response.setContentType("Content-Type: application/json; charset=UTF-8");

        Gson gson = new Gson();

        JSFile archivo = gson.fromJson(request.getParameter("p"), JSFile.class);

        DataBaseConexion db;

        db = new DataBaseConexion("galveniano", "Temp2018$$");

        ResultSet rs = db.getVersionFechaSubida(archivo.getId());

        List<VersionYFechaSubida> listadoVersiones;

        listadoVersiones = new ArrayList<>();

        while(rs.next()) {
            listadoVersiones.add(new VersionYFechaSubida(rs.getString("version"),
rs.getDate("fechaSubida").toLocalDate()));
        }

        // Convertir listado de versiones a JSON para enviarla de respuesta
        // gson.toJson(listadoVersiones);


        try (PrintWriter out = response.getWriter()) {

            out.print(gson.toJson(listadoVersiones));
        }
    }
```

Este **controlador** recibe de la vista principal el archivo seleccionado del que se desean mostrar las diferentes versiones de este que están guardadas en la base datos. Para ello *AJAXFileSelected.java* hace una conexión con la base de datos para, a través del *id* del archivo, obtener una lista de las versiones correspondientes a dicho archivo.

Este listado de versiones proporciona como su propio nombre indica contiene las diferentes versiones junto con la fecha de subida.

6.4 AJAXComparadorDeVersiones.java

```
@WebServlet(name = "AjaxComparadorDeVersiones", urlPatterns = {"/ajaxComparadorDeVersiones"})
public class AJAXComparadorDeVersiones extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, SQLException, ClassNotFoundException {
        response.setContentType("Content-Type: application/json; charset=UTF-8");
```

```
        // Recibe las dos versiones que se van a comparar
        // en formato JSON
        Gson gson = new Gson();

        Versiones versiones = gson.fromJson(request.getParameter("versiones"), Versiones.class);


        String[] arrFileNameVersionOriginal = versiones.getOriginal().split(";");
        String[] arrFileNameVersionRevisada = versiones.getRevisada().split(";");

        // Recuperamos de la base de datos los archivos que se van a comparar
        DataBaseConexion db;

        db = new DataBaseConexion("galveniano", "Temp2018$$");

        // Recuperamos el archivo original (version con ordinal menor)
        String fileNameVersionOriginal = arrFileNameVersionOriginal[0];
        int versionOriginal;
        versionOriginal = Integer.parseInt(arrFileNameVersionOriginal[1]);

        // Recuperamos el archivo revisado (version con ordinal mayor)
        String fileNameVersionRevisada = arrFileNameVersionRevisada[0];
        int versionRevisada;
        versionRevisada = Integer.parseInt(arrFileNameVersionRevisada[1]);

        // Primero: Recuperar el id del archivo, de la tabla jsfiles
        int id = db.getIdFromJSFile(fileNameVersionOriginal);

        // Segundo: Recuperamos el contenido de las versiones correspondientes,
        // de la tabla jsfiledata; entrando con el id y el numero de version
        // recuperamos el fileContent
        String originalFileContent = db.getFileContent(id, versionOriginal);
        String revisadaFileContent = db.getFileContent(id, versionRevisada);

        List<String> original = Arrays.asList(originalFileContent.split("\n"));
        List<String> revisada = Arrays.asList(revisadaFileContent.split("\n"));

        // Guardamos el contenido en disco, en un archivo cuyo contenido se borra inicialmente,
        // (Finalmente asi no se implemento de esta forma)
        // o mejor aun, los procesamos directamente para ver sus diferencias
        ComparadorDeVersiones cmpVersiones;

        cmpVersiones = new ComparadorDeVersiones(original, revisada);

        // Lista de cambios, o modificaciones, en la version revisada a partir de la original
        final List<Chunk> changesFromOriginal;
        changesFromOriginal = cmpVersiones.getChangesFromOriginal();

        // Listado de lineas nuevas añadidas a la version revisada
        final List<Chunk> insertsFromOriginal;
        insertsFromOriginal = cmpVersiones.getInsertsFromOriginal();

        // Listado de lineas borradas en la version revisada a partir de la original
        final List<Chunk> deletesFromOriginal;
        deletesFromOriginal = cmpVersiones.getDeletesFromOriginal();

        ComparacionResultante cmpResults = new ComparacionResultante(changesFromOriginal,
                insertsFromOriginal, deletesFromOriginal);

        try (PrintWriter out = response.getWriter()) {
            out.print(gson.toJson(cmpResults));
        }
    }
```

El **controlador** *AJAXComparadorDeVersiones.java* tiene como objetivo fundamental retornar a la vista principal las diferencias entre los archivos. Para conseguir este objetivo primeramente se conecta a la base de datos para obtener el id (existente en la tabla **jsfiles**) correspondiente a cada una de las versiones que vamos a comparar. Una vez conseguidos cada uno de los id, obtenemos de la base de datos y a partir de los id de las correspondientes versiones el contenido de cada una de las versiones.

Seguidamente de obtener el contenido de las versiones, realizamos un *Split* por salto de línea en cada una de las versiones para después compararlos. La comparación se realiza con la ayuda de la librería *diffutils-1.2.1.jar* creada por **Google**. Por lo que nosotros trabajamos con objetos de tipo *Chunk*[21] que nos permiten ir realizando la comparación línea a línea.

Por último, a la hora de comparar realizamos tres comparaciones diferentes: líneas modificadas, líneas insertadas y líneas borradas.

6.5 AJAXContenidoVersiones.java

```
@WebServlet(name = "AJAXContenidoVersiones", urlPatterns = {"/ajaxContenidoVersiones"})
public class AJAXContenidoVersiones extends HttpServlet {

    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException, SQLException, ClassNotFoundException {
       response.setContentType("Content-Type: application/json; charset=UTF-8");
       Gson gson = new Gson();

        Versiones versiones = gson.fromJson(request.getParameter("versiones"), Versiones.class);

        // Creamos los ficheros en disco, uno por cada una de las versiones recibidas
        String[] arrFileNameVersionOriginal = versiones.getOriginal().split(";");
        String[] arrFileNameVersionRevisada = versiones.getRevisada().split(";");
         // Recuperamos de la base de datos los archivos que se van a comparar
        DataBaseConexion db;

        db = new DataBaseConexion("galveniano", "Temp2018$$");


        // Recuperamos el archivo original (version con ordinal menor)
        String fileNameVersionOriginal = arrFileNameVersionOriginal[0];
        int versionOriginal;
        versionOriginal = Integer.parseInt(arrFileNameVersionOriginal[1]);
             // Recuperamos el archivo revisado (version con ordinal mayor)
        String fileNameVersionRevisada = arrFileNameVersionRevisada[0];
        int versionRevisada;
        versionRevisada = Integer.parseInt(arrFileNameVersionRevisada[1]);

        // Primero: Recuperar el id del archivo, de la tabla jsfiles
        int id = db.getIdFromJSFile(fileNameVersionOriginal);

        // Segundo: Recuperamos el contenido de las versiones correspondientes,
        // de la tabla jsfiledata; entrando con el id y el numero de version
        // recuperamos el fileContent
        String originalFileContent = db.getFileContent(id, versionOriginal);
        String revisadaFileContent = db.getFileContent(id, versionRevisada);

        ContenidoVersiones contenido;
        contenido = new ContenidoVersiones(originalFileContent, revisadaFileContent);

        try (PrintWriter out = response.getWriter()) {
            out.print(gson.toJson(contenido));
        }

    }
```

El **controlador** *AJAXContenidoVersiones.java* tiene como objetivo fundamental retornar a la vista principal el contenido de las versiones que se están comparando. Para conseguir este objetivo primeramente se conecta a la base de datos para obtener el id (existente en la tabla **jsfiles**) correspondiente a cada una de las versiones que vamos a comparar. Una vez

[21] Contiene la información sobre la parte del texto involucrada en el proceso de diferenciación.

conseguidos cada uno de los id, obtenemos de la base de datos y a partir de los id de las correspondientes versiones el contenido de cada una de las versiones.

Una vez obtenido el contenido de las versiones, creamos una instancia de la clase *ContenidoVersiones* pasándole como parámetros el contenido de la versión original y el contenido de la versión revisada.

7 HELPERS

7.1 Hash.java

```
public class Hash {

    /* Retorna un hash a partir de un tipo y un texto */
    public static String getHash(String txt, String hashType) {
        try {
            java.security.MessageDigest md = java.security.MessageDigest
                    .getInstance(hashType);
            byte[] array = md.digest(txt.getBytes());
            StringBuffer sb = new StringBuffer();
            for (int i = 0; i < array.length; ++i) {
                sb.append(Integer.toHexString((array[i] & 0xFF) | 0x100)
                        .substring(1, 3));
            }
            return sb.toString();
        } catch (java.security.NoSuchAlgorithmException e) {
            System.out.println(e.getMessage());
        }
        return null;
    }

    /* Retorna un hash MD5 a partir de un texto */
    public static String md5(String txt) {
        return Hash.getHash(txt, "MD5");
    }

    /* Retorna un hash SHA1 a partir de un texto */
    public static String sha1(String txt) {
        return Hash.getHash(txt, "SHA1");
    }

     /* Retorna un hash SHA512  a partir de un texto */
    public static String sha512(String txt) {
        return Hash.getHash(txt, "SHA-512");
    }
}
```

La **clase** *Hash.java* nos proporciona el **hash** del archivo a nivel de servidor, es decir, con **Java**. En un principio el proyecto estaba pensado para el calculo del **hash** a través del algoritmo *MD5* pero este método es un poco obsoleto por lo hemos provisto a esta **clase** con otros dos métodos, el SHA1 y el más fiable SHA512 que es el que se ha utilizado en el proyecto.

7.2 ArchivoSeleccionado.java

```
package helpers;

public class ArchivoSeleccionado {
    private final String nombre;
    private final String hash;

    public ArchivoSeleccionado(String nombre, String hash) {
        this.nombre = nombre;
        this.hash = hash;
    }

    public String getNombre() {
        return nombre;
```

```
    }

    public String getHash() {
        return hash;
    }

}
```

La **clase** *ArchivoSeleccionado.java* es una **clase** bastante sencilla y nos sirve para poder crear instancias, es decir, objetos de dicha **clase**.

ArchivoSeleccionado.java está compuesto de un constructor por defecto y los métodos *getNombre* y *getHash*. No posee los métodos **setters** ya que el *nombre* y el **hash** se inicializan al crear la instancia y posteriormente no nos interesa cambiarlos.

7.3 ComparacionResultante.java

```
public class ComparacionResultante {
    private final List<Chunk> modificaciones;
    private final List<Chunk> inserciones;
    private final List<Chunk> eliminaciones;

    public ComparacionResultante(List<Chunk> modificaciones, List<Chunk> inserciones, List<Chunk>
eliminaciones) {
        this.modificaciones = modificaciones;
        this.inserciones = inserciones;
        this.eliminaciones = eliminaciones;
    }

}
```

La **clase** auxiliar *ComparacionResultante.java* contiene un constructor común, esta **clase** nos sirve para poder agrupar las comparaciones comentadas en el **modelo** *AJAXComparadorDeVersiones.java* en un único objeto esta **clase**.

7.4 ComparacionDeVersiones.java

```
public class ComparadorDeVersiones {
    private final List<String> original;
    private final List<String> revisada;

    public ComparadorDeVersiones(List<String> original, List<String> revisada) {
        this.original = original;
        this.revisada = revisada;
    }

    public List<Chunk> getChangesFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.CHANGE);
    }

    public List<Chunk> getInsertsFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.INSERT);
    }

    public List<Chunk> getDeletesFromOriginal() throws IOException {
        return getChunksByType(Delta.TYPE.DELETE);
    }

    private List<Chunk> getChunksByType(Delta.TYPE type) throws IOException {
```

```
        final List<Chunk> listOfChanges = new ArrayList<Chunk>();
        final List<Delta> deltas = getDeltas();
        for (Delta delta : deltas) {
            if (delta.getType() == type) {
                listOfChanges.add(delta.getRevised());
            }
        }
        return listOfChanges;
    }

    private List<Delta> getDeltas() throws IOException {


        final Patch patch = DiffUtils.diff(original, revisada);

        return patch.getDeltas();
    }

}
```

La **clase** auxiliar *ComparadorDeVersiones.java* posee un constructor común además de los *métodos*:

- *getChangesFromOriginal*, este *método* retorna una lista de *Chunk* con los correspondientes cambios entre las versiones.

- *getInsertsFromOriginal*, este *método* retorna una lista de *Chunk* con las correspondientes líneas insertadas entre las versiones.

- getDeletesFromOriginal, este *método* retorna una lista de *Chunk* con la posición de inicio de las líneas eliminadas entre versiones.

Es importante destacar que la versión original hace referencia a la de menor orden ordinal y la versión revisada como la de mayor orden ordinal, es decir, para el caso de comparar la versión 2 con la 1, la primera versión correspondería con la versión original y la segunda como la versión revisada.

7.5 VersionYFechaSubida.java

```
public class VersionYFechaSubida {
    private final String version;
    private final LocalDate fechaSubida;

    public VersionYFechaSubida(String version, LocalDate fechaSubida) {
        this.version = version;
        this.fechaSubida = fechaSubida;
    }

    public String getVersion() {
        return version;
    }

    public LocalDate getFechaSubida() {
        return fechaSubida;
    }
```

La **clase** auxiliar *VersionYFechaSubida.java* contiene un constructor común además de dos *métodos getVersion, getFechaSubida* para obtener la versión y la fecha de subida del archivo respectivamente.

Esta **clase** es necesaria para mostrar en la vista principal para cada uno de los archivos que hay subidos sus diferentes versiones y la fecha de estas.

7.6 Versiones.java

```
public class Versiones {
    private final String original;
    private final String revisada;

    public Versiones(String original, String revisada) {
        this.original = original;
        this.revisada = revisada;
    }

    public String getOriginal() {
        return original;
    }

    public String getRevisada() {
        return revisada;
    }
```

La **clase** auxiliar *Versiones.java* contiene un constructor común además de dos *métodos*: *getOriginal*, *getRevisada* que obtiene la versión original y la versión revisada respectivamente.

7.7 ContenidoVersiones.java

```
public class ContenidoVersiones {
    private final String original;
    private final String revisada;

    public ContenidoVersiones(String original, String revisada) {
        this.original = original;
        this.revisada = revisada;
    }

    public String getOriginal() {
        return original;
    }

    public String getRevisada() {
        return revisada;
    }


}
```

La **clase** auxiliar *ContenidoVersiones.java* contiene un constructor por defecto además de dos métodos *getOriginal*, *getRevisada* para obtener el contenido de la versión original y revisada respectivamente.

Esta **clase** es necesaria para mostrar en la vista principal el contenido de las versiones cuando se están comparando.

8 CONCLUSIONES

8.1 Trabajo Realizado y líneas futuras

El proyecto permite al usuario el control de los ficheros que carga en la web y se almacenan en la base de datos mientras *viaja* hacia el servidor. Además, tiene una serie de funcionalidades secundarias, como el control de versiones o la capacidad de comparar las distintas versiones.

Este proyecto es una prueba de concepto, es decir, es un punto de partida donde poder desarrollar un proyecto a nivel profesional. En el mundo del desarrollo de software hay infinidad de posibilidades acerca de donde reconstruir el proyecto.

Para mí, la línea fundamental futura de desarrollo es la de la creación de una extensión de navegador web que albergase la capacidad para acceder a las librerías **JavaScript** y que permitiese realizar el análisis web de forma recurrente, puesto que de forma manual ya se plantea el modo en el presente proyecto.

El desarrollo de la extensión para el navegador web es la línea principal, como he comentado anteriormente, pero también sería interesante trabajar en dotarle de tanta funcionalidad como se quisiese y fuera posible.

9 BIBLIOGRAFÍA

https://gplsi.dlsi.ua.es/~slujan/materiales/pi-cliente2-muestra.pdf
http://mmc.geofisica.unam.mx/cursos/mcst-2007-II/Java/Java%20desde%20Cero.pdf
http://panamahitek.com/que-son-las-clases-en-java/
http://www.losteatinos.es/servlets/servlet.html
https://www.genbetadev.com/seguridad-informatica/que-son-y-para-que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales
https://www.genbetadev.com/seguridad-informatica/que-son-y-para-que-sirven-los-hash-funciones-de-resumen-y-firmas-digitales)
https://codigofacilito.com/articulos/mvc-model-view-controller-explicado
http://www.um.es/docencia/barzana/DAWEB/Introduccion-a-html-y-css.html
http://www.ulpgc.es/otros/tutoriales/JavaScript/
https://puntoabierto.net/blog/que-es-bootstrap-y-cuales-son-sus-ventajas
https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado
https://netbeans.org/features/index.html
https://es.wikipedia.org/wiki/GlassFish
https://es.wikipedia.org/wiki/MySQL

10 ANEXOS

10.1 Códigos ficheros JavaScript

file01.js

```
// Archivo de prueba
var a = 4;
var b = 5;
var c = 8;

function restar(x, y) {
        return x - y;
}

function multiplicar(x, y) {
    return x*y;
}

function dividir(x, y) {
    return x/y;
}

function compra(x,c) {
        return 0;
}

var persona = {
    nombre: 'Francisco'
};
```

file02.js

```
// Archivo de prueba 2
var a = 4;
var b = 5;

function sumar(x, y) {
        return x + y;
}

function restar(x, y) {
        return x - y;
}

function restar2(x, y) {
        return x - y;
}
```

file03.js

```
// Archivo de prueba
var a = 4;
var b = 5;

function sumar(x, y) {
        return x + y;
}

function restar(x, y) {
        return x - y;
}

function multiplicar(x, y) {
    return x*y;
}

function dividir(x, y) {
    return x/y;
}
```

file04.js

```
// Archivo de prueba
var a = 4;
var b = 5;

function sumar(x, y) {
    return x + y;
}

function restar(x, y) {
    return x - y;
}

function multiplicar(x, y) {
    return x*y;
}

function dividir(x, y) {
    return x/y;
}
```

file05.js

```
// Archivo de prueba
var a = 10;
var b = 60;

function sumar(x, y) {
        return x + y;
}


function multiplicar(x, y) {
    return x*y;
}
```

file06.js

```
// Archivo de prueba
var a = 4;
var c= 7;
var b = 5;

function sumar(x, y) {
        return x + y;
}

function restar3(x, y) {
        return x - y;
}

function multiplicar(x, y) {
    return x*y;
}
```