



GRADO EN INGENIERÍA INFORMÁTICA DE GESTIÓN Y SISTEMAS DE
INFORMACIÓN

TRABAJO FIN DE GRADO

2015 / 2016

PORTAL DE BÚSQUEDA DE VULNERABILIDADES WEB

MEMORIA DEL PROYECTO

DATOS DE LA ALUMNA O DEL ALUMNO

NOMBRE: JONATHAN

APELLIDOS: CASTRO GONZÁLEZ

FDO.:

FECHA: 8 DE FEBRERO DE 2016

DATOS DEL DIRECTOR O DE LA DIRECTORA

NOMBRE: MIKEL

APELLIDOS: VILLAMAÑE GIRONÉS

DEPARTAMENTO: LENGUAJES Y SISTEMAS INFORMÁTICOS

FDO.:

FECHA: 8 DE FEBRERO DE 2016

Índice

1	Introducción	11
1.1	Origen del proyecto	12
1.2	Motivaciones para la elección del proyecto	12
2	Planteamiento inicial	13
2.1	Descripción	13
2.2	Objetivos	14
2.3	Vulnerabilidades a explotar	14
2.3.1	Inyección SQL	15
2.3.2	<i>Cross-Site Request Forgery</i>	17
2.4	Definiciones, acrónimos, y abreviaturas	17
2.5	Herramientas	19
2.6	Arquitectura	22
2.6.1	Cliente	22
2.6.2	Servidor	22
2.7	Alcance	25
2.7.1	Estructura de Descomposición del Trabajo	25
2.8	Planificación temporal	46
2.8.1	Diagrama Gantt	47
2.9	Evaluación económica	49
2.9.1	Mano de obra	49
2.9.2	Software	49
2.9.3	Hardware	49
2.9.4	Gastos indirectos	51
2.9.5	Gastos totales	52
2.9.6	Contexto de negocio	52
2.10	Riesgos	54
2.10.1	Problemas con Python	55
2.10.2	“Quedarse en blanco”	55
2.10.3	Problemas con Raspberry Pi	56
2.10.4	Enfermedad o lesión	57
2.10.5	Inutilización del equipo informático	58
2.10.6	Pérdida de la conexión a Internet	59
3	Antecedentes	61
3.1	PunkSPIDER	61
3.2	Metasploit	63
3.3	OWASP Zed Attack Proxy Project	63

4	Captura de requisitos	65
4.1	Requisitos funcionales	65
4.2	Requisitos no funcionales	66
4.3	Jerarquía de actores	66
4.4	Casos de uso	67
4.4.1	<i>Front-end</i>	67
4.4.2	<i>Back-end</i>	68
4.5	Modelo de dominio	70
4.5.1	<i>Front-end</i>	70
4.5.2	<i>Back-end</i>	71
5	Análisis y diseño	73
5.1	Diagrama de base de datos	73
5.1.1	API	73
5.1.2	Servidor de búsqueda	75
5.2	Diagrama de clases del servidor de búsqueda	76
5.3	Esquema REST de la API	78
6	Elección de lenguajes y tecnologías	81
6.1	Python	81
6.1.1	Diferencias entre Python y Java	82
6.1.2	Elección del IDE	82
6.2	Base de datos del servidor de búsqueda	84
6.3	Blog y wiki	84
6.4	<i>Frameworks</i> para el <i>front-end</i>	85
6.4.1	Motor	85
6.4.2	Interfaz	87
6.5	API	87
6.6	Resumen	88
7	Desarrollo	91
7.1	<i>Back-end</i>	91
7.1.1	Base de datos general	92
7.1.2	Servidor de búsquedas	95
7.2	API	108
7.2.1	Estructura de directorios	108
7.2.2	Base de datos no relacional	109
7.2.3	Enrutamiento de las llamadas	110
7.2.4	Poblar datos y agrupación por proceso	111
7.2.5	Llamadas entre API y <i>back-end</i>	112
7.2.6	Gestión de librerías	113

7.3	Interfaz gráfica del <i>front-end</i>	113
7.3.1	Prototipo a papel	114
7.3.2	Prototipo a ordenador	115
7.3.3	Primera evolución de la Interfaz gráfica	118
7.3.4	Segunda evolución de la Interfaz gráfica	120
7.3.5	Interfaz gráfica definitiva	122
7.4	<i>Front-end</i>	123
7.4.1	Estructura de directorios	123
7.4.2	Gestión de librerías	125
7.5	Instalación de módulos	125
7.6	Blog y wiki	127
8	Legislación	129
9	Pruebas	131
9.1	<i>Front-end</i> y API	131
9.2	<i>Back-end</i>	137
9.2.1	Base de datos general	137
9.2.2	Núcleo del servidor de búsqueda	138
9.2.3	Módulo de <i>webcrawling</i>	138
9.2.4	Módulo de inyección SQL	139
9.2.5	Módulo de CSRF	139
9.2.6	Adaptador de la base de datos del servidor de búsqueda	140
10	Conclusiones	141
10.1	Cambios realizados durante el desarrollo	141
10.2	Planificación final	142
10.2.1	Reevaluación económica	144
10.3	Resultado final	145
10.4	Líneas futuras	145
10.5	Reflexión personal	147
	Bibliografía	149
A	Anexo I: Casos de uso extendidos	153
A.1	<i>Front-end</i>	153
A.1.1	Registrarse	153
A.1.2	Iniciar sesión	155
A.1.3	Configurar cuenta	157
A.1.4	Comprobar vulnerabilidades	160
A.1.5	Ver resultados	162

A.2	<i>Back-end</i>	164
A.2.1	Núcleo	164
A.2.2	Módulo de inyección SQL	165
A.2.3	Módulo de CSRF	166
B	Anexo II: Diagramas de secuencia	167
C	Anexo III: Manual de instalación	171
C.1	Base de datos	171
C.2	<i>Back-end</i>	173
C.3	<i>Front-end</i> y API	173
C.3.1	Raspberry Pi	174
C.4	Blog y wiki	176
D	Anexo IV: Manual de usuario	177
D.1	Registro o conexión	177
D.2	Actualizar perfil	178
D.3	Actualizar contraseña	179
D.4	Realizar una nueva búsqueda	180
D.5	Ver estado de la búsqueda	180
D.6	Ver resultados	181

Índice de figuras

1	Arquitectura deseada.	22
2	Arquitectura que se realizará.	24
3	Primera parte del diagrama EDT.	27
4	Segunda parte del diagrama EDT.	28
5	Diagrama Gantt.	47
6	Diagrama Gantt.	48
7	Jerarquía de actores.	66
8	Casos de uso del <i>front-end</i>	67
9	Casos de uso del <i>back-end</i>	68
10	Casos de uso del módulo de inyecciones SQL.	69
11	Casos de uso del módulo CSRF.	70
12	Modelo de dominio del <i>front-end</i>	70
13	Modelo de dominio del <i>back-end</i>	72
14	Diagrama de base de datos que tendrá la API.	74
15	Diagrama de base de datos del servidor de búsqueda.	75
16	Diagrama de clases del servidor de búsqueda.	76
17	Esquema REST de la API.	79
18	Tecnologías que se van a utilizar.	88
19	Diagrama de flujo del proceso.	91
20	Datos para crear un nuevo servidor de base de datos.	92
21	Datos para crear una nueva secuencia.	93
22	Añadiendo secuencia a una columna.	93
23	Diagrama de flujo del proceso de Inyección SQL.	104
24	Interfaz de la pantalla principal.	115
25	Interfaz del <i>login</i> /registro.	115
26	Interfaz de la pantalla principal.	116
27	Interfaz del <i>login</i> /registro.	116
28	Interfaz de una búsqueda.	118
29	Primera evolución de la interfaz de la página principal.	119
30	Primera evolución de la interfaz del <i>login</i> /registro.	119
31	Segunda evolución de la interfaz de la página principal.	121
32	Segunda evolución de la interfaz del <i>login</i> /registro.	121
33	Página principal final.	122
34	Formulario de registro final.	123
35	Estructura final de los ficheros del <i>front-end</i>	124
36	Aviso sobre el uso de las <i>cookies</i>	130
37	Diagrama de clases final del servidor de búsqueda.	142
38	Tabla comparativa de la duración inicial y real del proyecto.	143

39	Pantalla de registro o conexión.	154
40	Pantalla de perfil de usuario.	154
41	Pantalla de registro con un error.	154
42	Pantalla de conexión con error.	156
43	Pantalla de perfil con los datos actualizados.	157
44	Pantalla de perfil con error.	157
45	Pantalla de cambio de contraseña.	159
46	Pantalla de cambio de contraseña con los datos actualizados. .	159
47	Pantalla de cambio de contraseña con error.	159
48	Pantalla de búsqueda.	161
49	Pantalla de información sobre búsqueda activa.	161
50	Pantalla de búsqueda con error.	161
51	Pantalla de resultados.	163
52	Pantalla de resultados sin búsquedas.	163
53	Diagrama de secuencia sobre el estado del proceso.	167
54	Diagrama de secuencia sobre el <i>webcrawling</i>	168
55	Diagrama de secuencia sobre el módulo de inyección SQL. . .	169
56	Datos para crear un nuevo servidor de base de datos.	172
57	Pantalla para restaurar los datos de la base de datos.	172
58	Datos a introducir en el adaptador.	173
59	Página principal de Krashr.	177
60	Extremo superior derecho del <i>front-end</i>	178
61	Formulario de registro con aviso.	178
62	Pantalla de perfil de usuario.	179
63	Campo para insertar la contraseña al actualizar el perfil. . . .	179
64	Submenú del <i>front-end</i>	179
65	Pantalla de búsqueda con aviso de conexión.	180
66	Pantalla de búsqueda con error.	180
67	Pantalla de información sobre búsqueda activa.	181
68	Pantalla de información sin búsqueda activa.	181
69	Pantalla de resultados.	181

Índice de tablas

1	Tareas de las que se compone el trabajo.	29
2	Tareas de las que se compone el trabajo.	30
3	Hardware usado en el desarrollo.	50
4	Amortizaciones de las herramientas hardware.	50
5	Potencia de las herramientas hardware.	51
6	Gasto total.	52
7	Pros y contras de Eclipse.	83
8	Pros y contras de PyCharm.	83
9	Pros y contras de AngularJS.	86
10	Pros y contras de ReactJS.	86
11	Pruebas de la página principal.	132
12	Pruebas de la página de conexión/registro.	133
13	Pruebas de la página de la cuenta personal.	134
14	Pruebas de la página de cambio de contraseña.	134
15	Pruebas de la página de búsqueda.	135
16	Pruebas de la página de resultados.	136
17	Pruebas de la base de datos general.	137
18	Pruebas del núcleo del servidor.	138
19	Pruebas del módulo de <i>webcrawling</i>	138
20	Pruebas del módulo de inyección SQL.	139
21	Pruebas del módulo de CSRF.	139
22	Pruebas del adaptador de la base de datos.	140
23	Gasto final total.	145
24	Caso de uso extendido: registrarse.	153
25	Caso de uso extendido: iniciar sesión.	155
26	Caso de uso extendido: configurar cuenta.	158
27	Caso de uso extendido: comprobar vulnerabilidades.	160
28	Caso de uso extendido: ver resultados.	162
29	Caso de uso extendido: gestionar petición (núcleo).	164
30	Caso de uso extendido: buscar inyecciones SQL.	165
31	Caso de uso extendido: buscar errores de CSRF.	166

1. Introducción

Ninguna aplicación es perfecta. Cuando un desarrollador, o grupo de desarrolladores, se vanagloria de haber realizado una aplicación segura, con más ímpetu tratará el resto de buscar fallos en ella.

Estos fallos, o vulnerabilidades, salen caros. No hay problema si el que ha encontrado estos fallos no quiere hacer daño a la empresa desarrolladora, o desarrollador, que ha lanzado la aplicación. Sin embargo, ¿y si el que encuentra un fallo es un ciberdelincuente dispuesto a sacar partido de ellos y buscar beneficio?

La seguridad informática es un campo al que no se le presta la suficiente atención. Se han visto en los últimos años casos de grandes compañías tecnológicas que se han visto envueltas en robos de datos masivos a causa de una mala gestión de su seguridad. En algunos casos, sus máximos responsables dimitieron (p ej. Ashley Madison [1]); en otros, las empresas comunicaron a sus usuarios que tomaran las máximas precauciones (p. ej. Sony [2]) porque no podían hacer nada con los datos que les habían sido extraídos.

La mayor parte de las ocasiones, los datos sustraídos son de los propios usuarios que utilizan los servicios de estas grandes compañías y, en el peor de los casos, desconocen que sus datos están siendo usados, subastados y/o vendidos a lo largo y ancho de Internet.

La seguridad informática se trata de una disciplina que se ocupa de diseñar normas, procedimientos, métodos y técnicas destinados a conseguir un sistema de información seguro y confiable. Además, la seguridad informática comprende todo aquello que valore y signifique un riesgo, sea software o hardware, si la información llega a manos de otras personas.

El ámbito de la seguridad es muy amplio, por lo que este Trabajo de Fin de Grado (TFG) estará enfocado a tratar de buscar vulnerabilidades en sistemas web y ayudar a sus administradores y desarrolladores a corregirlos.

Se ha escogido esta sección porque las páginas web son una de las entradas más usadas por delincuentes en busca de la sustracción, relativamente sencilla, de datos.

1.1. Origen del proyecto

La idea de la realización de este proyecto surge durante la realización de un laboratorio de la asignatura llamada “Sistemas de Gestión de Seguridad de Sistemas de Información” (SGSSI) impartida por mi Director de proyecto, Mikel Villamañe.

En este laboratorio se debía realizar una página web de manera grupal. Sin embargo, lo interesante del mismo se trataba de intercambiar la página web realizada con otro grupo, con el objetivo de buscar una serie de vulnerabilidades vistas en clase. En ese preciso instante surgió la semilla que ha ido germinando hasta llegar a lo realizado en este trabajo.

1.2. Motivaciones para la elección del proyecto

La primera motivación de la realización de este proyecto ha sido ver cómo mis conocimientos adquiridos durante la carrera me permiten ejecutar correctamente este trabajo, en todos y cada uno de los aspectos del mismo.

La segunda motivación, más importante incluso que la primera, ha sido la elaboración de este trabajo pensando en un posible negocio que montar sobre la plataforma realizada.

La tercera motivación ha sido la de tratar de ayudar a otros desarrolladores a madurar unos conceptos de seguridad mínimos que hagan que sus páginas web sean más seguras.

Por último, la elección del uso de lenguajes diferentes a los estudiados y manejados durante el transcurso de la carrera, además de la realización de los análisis y diseño de una página web realizada mediante tecnologías modernas supone un nuevo paso que puede acabar en la posibilidad de obtener un empleo que antes se veía más complicado.

2. Planteamiento inicial

La finalidad de este capítulo es indicar cuáles son los objetivos y el alcance del proyecto; así como realizar una estimación económica y temporal del mismo.

A su vez, se lleva a cabo la gestión de riesgos, la especificación de la arquitectura de las aplicaciones a realizar y, por último, la descripción de las herramientas con las cuales se llevará a cabo su desarrollo.

2.1. Descripción

El proyecto consiste en la realización de un portal de búsqueda de vulnerabilidades web, llamado Krashr.

El objetivo de este portal será el de buscar si una página web introducida por un usuario contiene algún tipo de vulnerabilidad explotable, además de tratar de ayudar a este usuario a arreglar las vulnerabilidades encontradas.

Adicionalmente, enfocado a la ayuda de los futuros usuarios y mejora de la plataforma, se realizarán otras dos partes que complementarán al portal de búsquedas:

- Una wiki en la que cualquier usuario registrado en el sistema pueda escribir información sobre cualquier vulnerabilidad o temas relacionados con la seguridad web.
- Un blog en el que el/los administrador/es podrá/n escribir acerca de noticias relacionadas con este mundo y en el que los usuarios tendrán la opción de poder opinar.

Por lo tanto, el público al que irá dirigido este proyecto debería tener unos conocimientos mínimos en desarrollo de aplicaciones y cierto interés en el mundo de la seguridad. Por ello, puede haber varios aspectos (p. ej. lenguaje técnico con posibles tecnicismos en inglés por no tener su correspondiente traducción al castellano) no orientados a usuarios sin estos conocimientos mínimos.

Llegados a este punto, pueden surgir dos cuestiones:

1. ¿Crear una aplicación de escritorio o crear un portal web?
2. ¿Hacer que un desarrollador suba su página web a “producción” aún conociendo que ésta puede ser vulnerable?

La respuesta a la primera pregunta está más enfocada al alcance que puede tener el proyecto. Sería más sencillo realizar una aplicación de escritorio, pero significaría tener que realizar una aplicación que fuera multiplataforma y, además, actualizable. Por lo tanto, es más útil realizar una aplicación web en la que cualquiera puede acceder si tiene acceso a Internet y que, por lo tanto, toda actualización realizada sobre la misma es transparente para el usuario.

La respuesta a la segunda pregunta no es sencilla, pero es entendible. La idea de Krashr es la realización de “ataques” de manera controlada a páginas web estando éstas ya en línea, es decir, en fase de desarrollo pero en un entorno real. De realizar ataques en entornos controlados o locales es posible que, a la hora del despliegue de la página web a analizar, se cambien configuraciones; de nada serviría encontrar fallos y que, tras el despliegue, surjan otros no previstos. Por lo tanto, toda página web que se quiera analizar deberá estar subida a Internet. Se recomienda no subirla con los datos finales de “producción”.

2.2. Objetivos

El principal objetivo será realizar completamente la aplicación web en la manera en la que se detallará en los apartados siguientes, detallando en todo momento los problemas encontrados y todos los cambios que se hayan realizado. Por supuesto, el desarrollo se ha de tratar de completar dentro de los límites de tiempo propuestos.

A falta de preparar los requisitos, el servidor deberá ser capaz de realizar un *webcrawling* a partir de la URL especificada por el usuario y buscar, en todo el árbol de páginas web obtenido, las vulnerabilidades de inyección SQL y *Cross-Site Request Forgery*.

Otro punto importante es que la interfaz de la aplicación sea lo más intuitiva posible, consiguiendo que el tiempo de adaptación del usuario a la misma sea prácticamente inexistente. Se realizarán diferentes prototipos en los que potenciales usuarios darán su opinión acerca de la evolución de los mismos, mediante encuestas y/o entrevistas.

2.3. Vulnerabilidades a explotar

Antes de comenzar con las explicaciones sobre este proyecto, conviene comentar acerca de las vulnerabilidades que se van a intentar explotar en las

páginas web introducidas por los usuarios.

Como bien se ha comentado en la subsección anterior, la que comenta los objetivos, este proyecto tratará de buscar dos tipos de vulnerabilidades: inyecciones SQL y *Cross-Site Request Forgery*.

Se ha decidido la explotación de estas dos vulnerabilidades por ser sencillas de encontrar, además de ser, a priori, bastante dañinas.

Se usará el proyecto OWASP¹ (Open Web Application Security Project) para documentar todo lo relacionado con las vulnerabilidades, por ser un proyecto de código abierto dedicado a determinar y combatir las causas que hacen que el software sea inseguro.

Cuentan con guías y documentos acerca de seguridad de aplicaciones, entre ellos, el famoso OWASP TOP 10², que se trata de un documento de los diez riesgos de seguridad más importantes en aplicaciones web según la organización OWASP que se publica y actualiza cada tres años.

2.3.1. Inyección SQL

La inyección SQL³ es un método de infiltración de código intruso en una aplicación en el nivel de validación de las entradas para realizar operaciones sobre una base de datos.

Lleva, por lo menos, desde 2010, siendo el ataque más realizado según el OWASP Top 10. Además, es un ataque **fácil** de ejecutar, con una “detectabilidad” **media** y con un impacto **severo**.

Una inyección exitosa no sólo puede leer, o modificar, el contenido de una base de datos, también puede llegar a ejecutar operaciones sobre el propio gestor de base de datos o hasta en el sistema operativo en el que corre el mismo gestor.

Además, este tipo de ataques afecta a cualquier gestor de bases de datos SQL, pudiendo estar ejecutándose en cualquier sistema operativo.

Los errores de inyección SQL injection ocurren cuando:

- Los datos entran al programa desde una fuente no confiable.
- Los datos se usan para construir sentencias SQL dinámicamente.

¹Acerca de OWASP: https://www.owasp.org/index.php/Main_Page.

²https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project.

³Más información en: https://www.owasp.org/index.php/SQL_Injection.

Las principales consecuencias son:

- Confidencialidad: las bases de datos SQL, generalmente, almacenan datos sensibles.
- Autenticación: si se usan comando débiles para comprobar usuarios y contraseñas, es posible conectarse al sistema como otro usuario sin tener que conocer su contraseña.
- Autorización: si la información de autorización está almacenada en una base de datos SQL, es posible que se pueda cambiar dicha información a través de un ataque de inyección SQL.
- Integridad: al igual que es posible leer información sensible, también se puede modificar o borrar dicha información.

Las inyecciones SQL se han convertido en el principal problema en páginas web que usan bases de datos SQL. Es un fallo que se detecta muy fácilmente y que se puede explotar, también, de manera muy sencilla. Por lo tanto, cualquier aplicación que contenga una base de datos puede ser sujeto de este tipo de ataques.

Existen varios tipos de inyecciones SQL. Este proyecto buscará tres de estos tipos:

1. Intento de autenticación. El objetivo es acceder como administrador de la aplicación web sin conocer las credenciales del mismo, aprovechando los formularios de conexión.
2. Inyección basada en errores. Se trata de realizar búsquedas en direcciones con parámetros hasta que la aplicación muestre un error del sistema gestor de base de datos.
3. *Blind SQL injection*. Se trata de otro método similar al de la inyección basada en errores. Se pretende enviar una petición a la base de datos de la aplicación para que tarde la cantidad de tiempo que se quiera. La web será vulnerable si tarda ese tiempo que se le ha especificado.

Durante el desarrollo del módulo de inyección SQL se explicará de manera más detallada la detección de los errores que permiten estos ataques y cómo evitarlos.

2.3.2. *Cross-Site Request Forgery*

La falsificación de petición en sitios cruzados⁴, más conocido por sus siglas en inglés CSRF (*Cross-Site Request Forgery*), se trata de un ataque cuyo objetivo se trata de conseguir que una víctima, que tiene una sesión con su cuenta en un sitio web, haga acciones involuntarias en esa sesión al abrir un enlace o cargar un contenido externo.

Aunque se trata de un ataque que en el OWASP Top 10 de 2010 se encontraba en quinta posición y que en la versión de 2013 ha bajado a la octava, se ha seleccionado por ser un ataque con una ejecución **media**, pero con una “detectabilidad” **fácil**, además de tener un impacto **moderado**.

Este tipo de ataques se centra no tanto en el robo de datos, puesto que el atacante no puede ver la respuesta a la petición manipulada, sino en el cambio de estado de los mismos. Dependiendo del nivel de acceso de la víctima, el atacante podría desde cambiar datos de la cuenta de la víctima hasta tomar control de la aplicación web.

Para la mayoría de servicios web, las peticiones incluyen las credenciales del usuario autenticado, que pueden ser a través de *cookies*, IP, credenciales de dominio, y de otras maneras. Por lo tanto, si la aplicación web no cuenta con las medidas para contrarrestar CSRF, la propia aplicación no tendría manera de distinguir entre las peticiones legítimas de la víctima y las que se realizan a través de un ataque.

Durante el desarrollo del módulo de CSRF se explicará la manera de detectar los errores que permiten estos ataques y cómo evitarlos.

2.4. Definiciones, acrónimos, y abreviaturas

A continuación, se expondrán y explicarán términos relacionados con el trabajo a realizar.

API Una API, acrónimo creado a partir del inglés *Application Programming Interface* (que se traduce como Interfaz de Programación de Aplicaciones), es un conjunto de funciones y procedimientos que cumplen una o muchas funciones con el fin de ser utilizadas por otro software.

⁴Más información en: <https://www.owasp.org/index.php/CSRF>.

Back-end En el desarrollo web, el *back-end* es la parte que procesa la entrada desde el *front-end*. Es la parte encargada de la manipulación de los datos, interacción con la base de datos y generación de plantillas a servir al *front-end*.

Exploit Un *exploit* es una secuencia de comandos utilizada con el fin de aprovechar una vulnerabilidad en la seguridad de un sistema con el propósito de conseguir un comportamiento no deseado del mismo.

Firewall Programa informático, o componente *hardware*, que comprueba la información procedente de Internet o de una red y que, a partir de ella, bloquea o permite el paso de ésta al equipo en cuestión en función de la configuración realizada.

Framework Entorno o ambiente de trabajo para desarrollo; dependiendo del lenguaje normalmente integra componentes que facilitan el desarrollo de aplicaciones.

Front-end En el desarrollo web, es la parte responsable de recolectar los datos de entrada del usuario y ajustarlos a las especificaciones que demanda el *back-end* para poder procesarlos. Además, otra de sus funciones es la de recibir y exponer al usuario, de una forma amigable para él, los mensajes y datos recibidos desde el *back-end*.

HTML5 HTML5, acrónimo inglés de *HyperText Markup Language* versión 5, es la quinta revisión importante del lenguaje básico de la *World Wide Web* HTML.

IDE IDE, del acrónimo inglés *Integrated Development Environment* (traducido como Entorno de Desarrollo Integrado), es un entorno de programación consistente en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica. Puede estar pensado para un único o varios lenguajes de programación.

JavaScript JavaScript es un lenguaje de programación que permite a los desarrolladores crear acciones en sus páginas web.

Node.js Servidor que trabaja con lenguaje JavaScript, cuyo núcleo en su arquitectura cuenta con el motor JavaScript V8 desarrollado por Google para su navegador Chrome.

NoSQL Las bases de datos NoSQL buscan resolver los problemas de agilidad, escalabilidad y rendimiento que las bases de datos relacionales no fueron diseñadas para abordar. Son sistemas de almacenamiento de información que no cumplen con el esquema entidad-relación.

Pentesting *Pentesting*, o prueba de penetración, es un ataque a un sistema informático con la intención de encontrar sus debilidades en seguridad y todo lo que podría tener acceso a este sistema, su funcionalidad y datos.

Proxy Se trata de un servidor (que puede ser tanto *hardware* como *software*), que hace de intermediario entre la comunicación de otras dos máquinas.

Python Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, usa tipado dinámico y es multiplataforma.

SQL Del inglés *Structured Query Language* (que se traduce como Lenguaje de Consulta Estructurado), identifica a un tipo de lenguaje vinculado con la gestión de bases de datos de carácter relacional que permite la especificación de distintas clases de operaciones entre éstas.

URL Acrónimo de *Uniform Resource Locator* (traducido en español como Localizador Uniforme de Recursos), permite al navegador encontrar una dirección o sitio web en Internet.

2.5. Herramientas

En esta sección, se explicarán las herramientas que se usarán durante el desarrollo del proyecto, además del uso que se le dará a cada una de ellas. Cabe mencionar que, durante el transcurso del trabajo, es posible que se tengan que usar otras herramientas no previstas.

Dia Editor de diagramas gratuito, de código abierto y multiplataforma. Se tratará de usar esta herramienta para realizar diagramas siempre que se pueda. Es posible que la herramienta no sea lo suficiente potente y sea necesario usar *Visual Paradigm*.

Dropbox Es un servicio de alojamiento de archivos multiplataforma en la nube. Se usará para ir subiendo los documentos finales de cada apartado o, si es necesario, la subida de nueva documentación o ficheros para poder realizar la aplicación. Además, es una buena opción para realizar copias de seguridad de pequeños ficheros y documentos.

GitLab Es una aplicación de código abierto que permite administrar repositorios en *git* mediante una interfaz web. A diferencia de *GitHub*, se usará esta otra herramienta porque permite el uso de repositorios privados de manera gratuita.

Google Chrome Navegador web desarrollado por Google. Se usará este navegador para realizar pruebas sobre el portal de vulnerabilidades.

LaTeX Es un programa de creación de textos, especialmente orientado para cubrir las necesidades de los técnicos. Se usará este editor para crear toda la documentación necesaria.

Microsoft Internet Explorer Navegador web desarrollado por Microsoft para el sistema operativo Microsoft Windows. Se usará este navegador para realizar pruebas sobre el portal de vulnerabilidades.

Microsoft Project/Visio Es un software que permite a un administrador gestionar sus proyectos. Se usará esta herramienta para realizar toda la organización y planificación del proyecto, además de para realizar ciertos diagramas necesarios para la realización de la memoria.

Mozilla Firefox Navegador web libre, de código abierto y multiplataforma, desarrollado por la Fundación Mozilla. Se usará este navegador para realizar pruebas sobre el portal de vulnerabilidades.

Notepad++/GEdit Se trata de programas *software* que permiten crear ficheros de texto y que resaltan las palabras clave de multitud de lenguajes de programación. Se usarán estas dos herramientas para editar ficheros PHP, de ser necesarios, o para editar ficheros JavaScript.

Raspberry Pi Ordenador de placa reducida (SBC) de bajo coste desarrollado en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. Se usará este aparato con el fin de subir el *front-end* y la API para realizar pruebas en un entorno más real que la de una máquina virtual.

VirtualBox Software de virtualización de sistemas para arquitecturas x86 y amd64. Se usará esta herramienta para la creación de las máquinas virtuales necesarias para poder realizar el proyecto.

Visual Paradigm Se trata de una herramienta que permite realizar diagramas UML. Se usará esta herramienta para realizar los diagramas necesarios para poder realizar la implementación en código del trabajo, siempre y cuando no se puedan realizar estos diagramas usando la herramienta *Dia*.

2.6. Arquitectura

En este apartado se tratará de explicar cómo se quiere establecer la aplicación de una forma breve y sencilla. Para tratar de explicarlo de la mejor manera, la Figura 1 contempla la idea inicial de la arquitectura a montar.

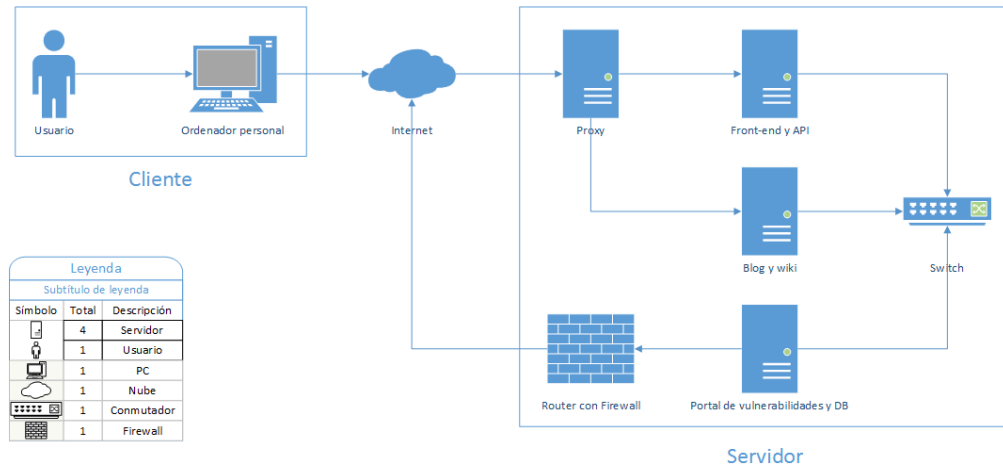


Figura 1: Arquitectura deseada.

2.6.1. Cliente

El cliente será el navegador por el que los usuarios accedan a la interfaz gráfica de la aplicación, además de al contenido del blog y de la wiki.

El uso del navegador será poco más que el de mostrar, de manera agradable, la información recibida por el servidor. Aunque es posible que se realicen ciertas funciones, no serán nada más que para aliviar de tareas al propio servidor.

2.6.2. Servidor

En un principio, se está teniendo en cuenta como si el proyecto estuviera siendo desarrollado por una empresa y éste acabara en producción.

Una red local de tres servidores serían los que llevaran el peso del alojamiento y manejo de los datos, además de ser, uno de ellos, el responsable de la búsqueda propia de vulnerabilidades. Existirían otros dos servidores responsables de tareas de seguridad, enrutamiento y serían los únicos con acceso a Internet.

Por lo tanto, los tres servidores en red local sería los que contengan la aplicación en sí, y estarían divididos en:

1. *Front-end* y API. Se encargaría de alojar el *front-end* y la API, junto con su base de datos no relacional. Respondería al cliente con el recurso pedido, además de ser quien realice las peticiones al servidor con la aplicación de búsqueda.
2. Wiki y blog. Se trataría de un servidor web encargado de alojar el blog y la wiki y responder a las consultas que se hagan sobre ellos.
3. Portal de vulnerabilidades web. Se alojaría la base de datos general y la aplicación realizada en Python. Se encargaría de la búsqueda de vulnerabilidades de una página web enviada desde la API y del guardado de todos los datos necesarios para el correcto funcionamiento de la aplicación en la base de datos.

Los dos servidores con acceso a Internet serían el *proxy* y el *router* con *firewall*. La idea es que los tres servidores que alojan los datos no tengan acceso a Internet, con el objetivo de tratar de minimizar los riesgos a la hora de recibir un ataque no deseado.

1. El *proxy* sería la primera respuesta de un cliente, enviando la petición al servidor que haga falta, es decir, hacia la web/API o el blog/wiki. En caso de ser algún ataque o de una petición incorrecta, esta petición sería rechazada.
2. El *router firewall* se encargaría de enviar las peticiones de búsqueda del portal y de recibir las respuestas de búsqueda y enviarlas de vuelta al propio portal. Únicamente dejaría entrar toda comunicación que el portal haya empezado, rechazando toda otra actividad.

Como bien se ha escrito anteriormente, el diseño sería el adecuado para una empresa con cierto capital y volumen alto de datos. Sin embargo, como el coste que conllevaría la puesta a punto a escala real no es manejable, la arquitectura que se realizará para el desarrollo y prueba de este TFG será el mostrado en la Figura 2.

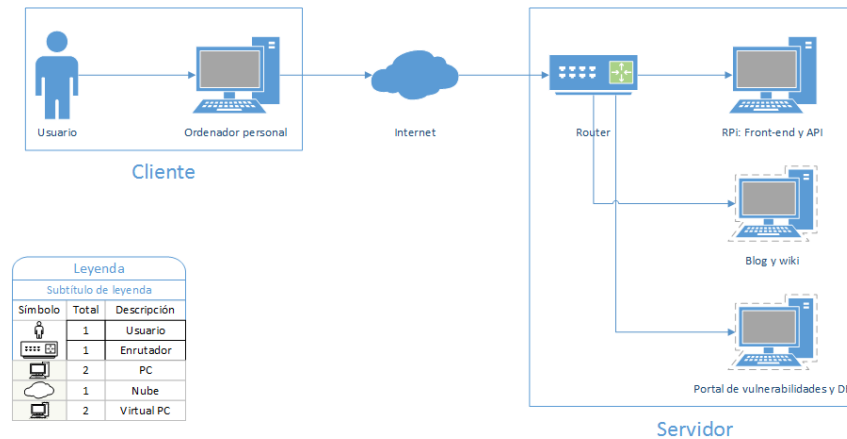


Figura 2: Arquitectura que se realizará.

Se puede observar que tanto el *proxy* como el *router* con *firewall* se han eliminado para dejar paso a un *router* convencional con conexión a Internet, haciendo el mismo de *switch* para montar una red local con los otros tres elementos de la imagen.

El servidor con el *front-end* y la API se cambia por una Raspberry Pi, lo suficientemente potente y con un bajo consumo como para poder realizar las pruebas a la web sin tener que encender el propio ordenador de sobremesa con las máquinas virtuales. Por lo tanto, si el ordenador está apagado no hay búsqueda de vulnerabilidades ni blog/wiki, sin embargo, se encenderá para realizar aquellas pruebas que requieran su uso.

Complementando a la Raspberry Pi, se encontraría el ordenador de sobremesa, mencionado en el párrafo anterior, que contendrá las, en principio –puede que el blog y la wiki se acaben instalando en el sistema principal y no en una máquina virtual–, dos máquinas virtuales con el blog/wiki y el propio portal con la base de datos, respectivamente.

El problema principal será que, al ser máquinas diferentes y, al no contar con *proxy*, se tendrán que asignar diferentes puertos a la web y al blog/wiki, para poder realizar llamadas a los diferentes recursos de los mismos.

2.7. Alcance

Una de las primeras tareas de todo proyecto, es la definición de su alcance, o dicho de otra forma, delimitar las funcionalidades a desarrollar (en el caso de un producto software) para poder cumplir con todos sus objetivos al finalizar el proyecto.

2.7.1. Estructura de Descomposición del Trabajo

La distribución se ha realizado por módulos y no como normalmente se ha trabajado hasta ahora –organización, análisis, diseño, implementación, pruebas y documentación–. Esta distribución se puede observar en el diagrama EDT (Estructura de Descomposición del Trabajo), que se muestran en las Figuras 3 y 4, por la lista extensa de tareas que componen este TFG.

Se ha tratado de dividir el proyecto en pequeños “trozos” en los que cada uno tiene varias tareas a realizar. Existirá la posibilidad de realizar un módulo antes que otro, aunque éste segundo aparezca después en el Gantt, siempre y cuando, no tenga ninguna dependencia y ésta se haya realizado. Por ejemplo, se puede realizar el *front-end* y API antes que la propia aplicación en Python; sin embargo, no se podría realizar el blog y la wiki antes que el *front-end* y API, puesto que es necesaria la previa implementación de la base de datos no relacional para poder continuar.

Otra ventaja de esta manera de distribución es la de poder realizar las diferentes etapas a la hora del desarrollo (análisis y diseño) de cada módulo. Por lo tanto, los diagramas serán más sencillo de realizar y de entender, incluso facilitará el trabajo de modificación de existir algún fallo. Además, de atragantarse un módulo, siempre se puede saltar a otro para no perder tiempo.

Sin embargo, no todas son ventajas. El principal problema es que al haber más módulos, habrá más diagramas a realizar; por lo tanto, se habrá de explicar cada uno de ellos, haciendo que todo sea más pesado.

La lista de cada uno de los “trozos”, con una breve explicación, es la siguiente:

- **Organización y aprendizaje.** Consistirá en organizar el proyecto a realizar, junto con el estudio de las herramientas o lenguajes que sean necesarios.
- **Realización del *back-end* de la aplicación.** Realización de la captura de requisitos, análisis, diseño, implementación y desarrollo de las pruebas del *back-end* de la aplicación.
- **Realización del *front-end* y API de la aplicación.** Realización de la captura de requisitos, análisis, diseño, implementación y desarrollo de las pruebas tanto del *front-end*, como de la API de la aplicación.
- **Implantación.** Implantar la aplicación para que sea accesible para su uso desde el exterior.
- **Wiki y blog.** Modificación, implantación y configuración del *software* de wiki y blog.
- **Leyes.** Estudio de la legislación actual y su implantación en la aplicación.
- **Documentación.** Realizar toda la documentación requerida para explicar el completo desarrollo de la herramienta, junto con la escritura de todos los manuales necesarios para la instalación y uso de la misma.

Nota: cuando se habla de “configuración de seguridad incorrecta” se refiere a la vulnerabilidad CSRF.

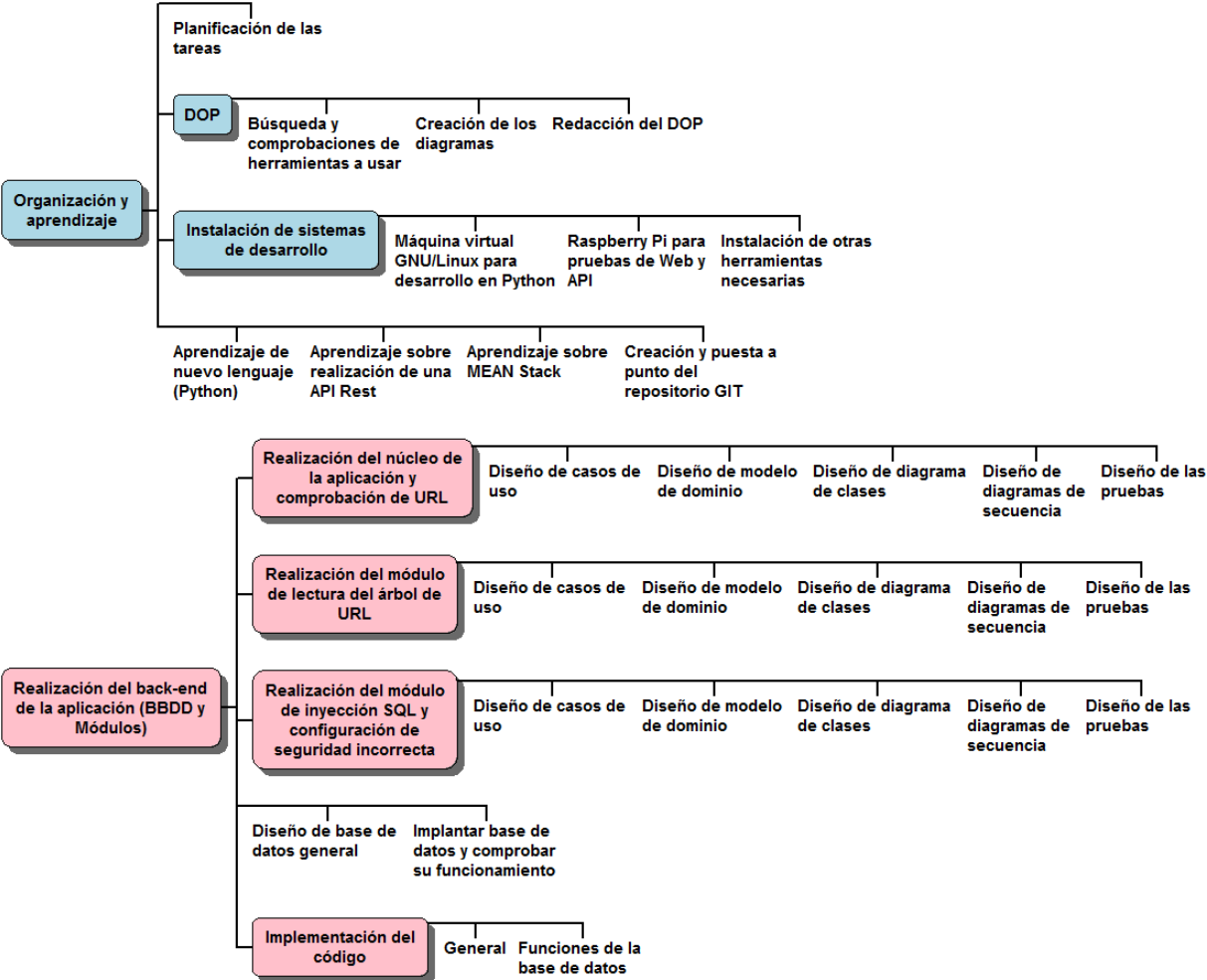


Figura 3: Primera parte del diagrama EDT.

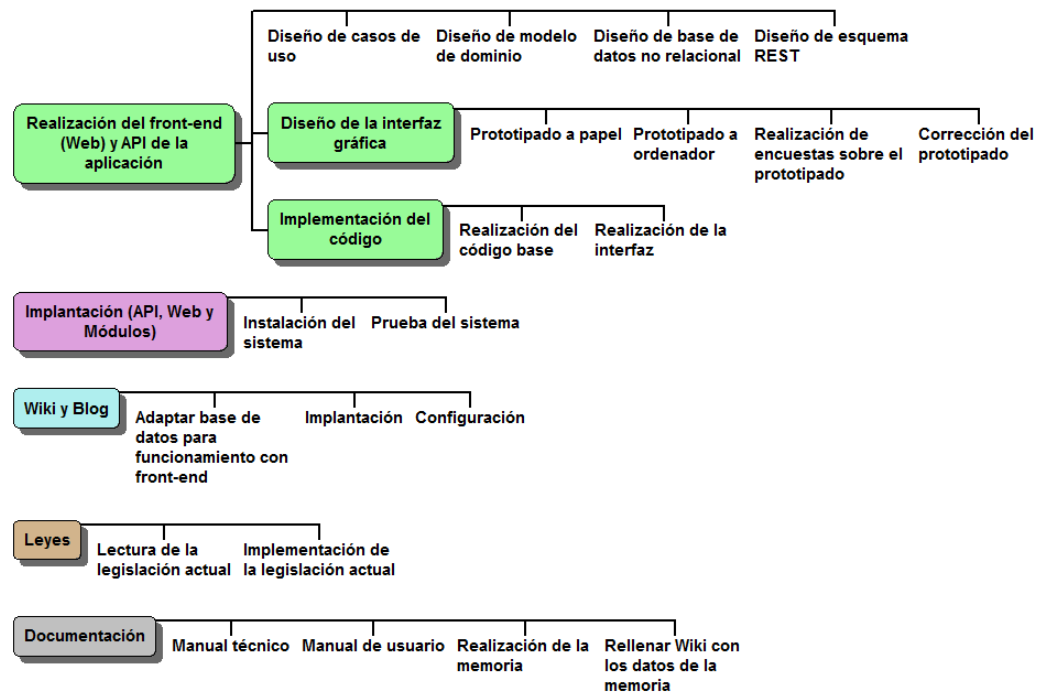


Figura 4: Segunda parte del diagrama EDT.

A continuación, se explicará cada una de las tareas de manera más detallada. Las Tablas 1 y 2 muestran un resumen de estas tareas. En ellas se pueden observar la duración de cada tarea y sus dependencias.

Id	Nombre	Predecesoras	Duración
0	Portal de búsqueda de vulnerabilidades		572 horas
1	Organización y aprendizaje		116 horas
2	Planificación de las tareas		8 horas
3	DOP	2	30 horas
4	Búsqueda y comprobaciones de herramientas a usar	2	10 horas
5	Creación de los diagramas	4	8 horas
6	Redacción del DOP	5	12 horas
7	Instalación de sistemas de desarrollo	3	18 horas
8	Máquina virtual GNU/Linux para desarrollo en Python	6	4 horas
9	Raspberry Pi para pruebas de Web y API	8	10 horas
10	Instalación de otras herramientas necesarias	9	4 horas
11	Aprendizaje de nuevo lenguaje (Python)	7;10	40 horas
12	Aprendizaje sobre realización de una API Rest	11	8 horas
13	Aprendizaje sobre MEAN Stack	12	10 horas
14	Creación y puesta a punto del repositorio GIT	13	2 horas
15	Realización del back-end de la aplicación (BBDD y Módulos)	1	229 horas
16	Realización del núcleo de la aplicación y comprobación de URL	1	28 horas
17	Diseño de casos de uso	14	6 horas
18	Diseño de modelo de dominio	17	4 horas
19	Diseño de diagrama de clases	18	3 horas
20	Diseño de diagramas de secuencia	19	10 horas
21	Diseño de las pruebas	20	5 horas
22	Realización del módulo de lectura del árbol de URL	16	20 horas
23	Diseño de casos de uso	21	4 horas
24	Diseño de modelo de dominio	23	3 horas
25	Diseño de diagrama de clases	24	2 horas
26	Diseño de diagramas de secuencia	25	6 horas
27	Diseño de las pruebas	26	5 horas
28	Realización del módulo de inyección SQL y configuración de seguridad incorrecta	22	20 horas
29	Diseño de casos de uso	27	4 horas
30	Diseño de modelo de dominio	29	3 horas
31	Diseño de diagrama de clases	30	2 horas
32	Diseño de diagramas de secuencia	31	6 horas
33	Diseño de las pruebas	32	5 horas
34	Diseño de base de datos general	18;24;30;33	6 horas
35	Implantar base de datos y comprobar su funcionamiento	34	5 horas
36	Implementación del código	18;24;30;35	150 horas
37	General	35	110 horas
38	Funciones de la base de datos	37	40 horas

Tabla 1: Tareas de las que se compone el trabajo.

Id	Nombre	Predecesoras	Duración
39	Realización del front-end (Web) y API de la aplicación	15	100 horas
40	Diseño de casos de uso	38	8 horas
41	Diseño de modelo de dominio	40	4 horas
42	Diseño de base de datos no relacional	41	3 horas
43	Diseño de esquema REST	42	5 horas
44	Diseño de la interfaz gráfica	43	30 horas
45	Prototipado a papel	43	2 horas
46	Prototipado a ordenador	45	12 horas
47	Realización de encuestas sobre el prototipado	46	10 horas
48	Corrección del prototipado	47	6 horas
49	Implementación del código	42;43;44	50 horas
50	Realización del código base	42;43;44;47	40 horas
51	Realización de la interfaz	47;50	10 horas
52	Implantación (API, Web y Módulos)	15;39	6 horas
53	Instalación del sistema	51;15;39	2 horas
54	Prueba del sistema	53	4 horas
55	Wiki y Blog	15;39;52	13 horas
56	Adaptar base de datos para funcionamiento con front-end	41;50;54	8 horas
57	Implantación	56	1 hora
58	Configuración	57	4 horas
59	Leyes	15;39;52;55	42 horas
60	Lectura de la legislación actual	58	32 horas
61	Implementación de la legislación actual	60	10 horas
62	Documentación	15;39;52;55;59	66 horas
63	Manual técnico	15;39;52;55;59;61	8 horas
64	Manual de usuario	39;55;63	8 horas
65	Realización de la memoria	1;15;39;52;55;59;63;64	40 horas
66	Rellenar Wiki con los datos de la memoria	65	10 horas

Tabla 2: Tareas de las que se compone el trabajo.

2.7.1.1. Organización y aprendizaje

Tareas que permitirán organizar el trabajo a realizar y aprender lo necesario para poder empezar a desarrollarlo.

Planificación de las tareas
Paquete de trabajo: Organización y aprendizaje. Duración: 8 horas.
Descripción: planificar las tareas que se tienen que realizar para alcanzar el objetivo de este trabajo de fin de grado. Salidas/Entregables: tareas a realizar. Recursos necesarios: software planificador de proyectos.

2.7.1.1.1. DOP

La realización del DOP puede parecer una tarea sencilla, sin embargo, no solamente se trata de escribir. La búsqueda de las herramientas a usar, además de pensar en el tiempo para preparar los diagramas es una importante suma a la hora de preparar este DOP.

Búsqueda y comprobaciones de herramientas a usar

Paquete de trabajo: DOP.

Duración: 10 horas.

Descripción: buscar información sobre las herramientas necesarias para la realización de este TFG por completo, además de comprobar que realmente dichas herramientas satisfacen por completo mis necesidades.

Entradas: tareas a realizar.

Salidas/Entregables: herramientas a usar.

Recursos necesarios: Internet, experiencia previa con herramientas.

Creación de los diagramas

Paquete de trabajo: DOP.

Duración: 8 horas.

Descripción: creación de todos los diagramas necesarios para incluir en el DOP. En esta tarea no se contemplan aquellos diagramas necesarios para realizar los análisis y diseños propios de la aplicación a crear.

Entradas: tareas a realizar, herramientas a usar.

Salidas/Entregables: diagramas para el DOP.

Recursos necesarios: herramientas necesarias para realizar los diagramas.

Redacción del DOP

Paquete de trabajo: DOP.

Duración: 12 horas.

Descripción: redacción del documento de objetivos del proyecto, añadiendo los diagramas necesarios creados anteriormente y explicadas las herramientas a usar/usadas.

Entradas: tareas a realizar, herramientas a usar, diagramas para el DOP.

Salidas/Entregables: documento de objetivos del proyecto (DOP).

Recursos necesarios: editor de textos y software planificador de proyectos.

2.7.1.1.2. Instalación de sistemas de desarrollo

Como bien se ha explicado en el apartado de arquitectura, se necesitarán de varios sistemas para realizar y desplegar este TFG, de ahí que se hayan dividido esta tarea en varias.

Máquina virtual GNU/Linux para el desarrollo en Python

Paquete de trabajo: Instalación de sistemas de desarrollo.

Duración: 4 horas.

Descripción: preparar una máquina virtual e instalar sobre ella el sistema y el software adecuado para poder programar en Python y desarrollar la Web, API y Núcleo de la aplicación.

Entradas: DOP.

Salidas/Entregables: máquina virtual.

Recursos necesarios: software virtualizador, sistemas operativo, IDE, Python y dependencias.

Raspberry Pi para pruebas de Web y API

Paquete de trabajo: Instalación de sistemas de desarrollo.

Duración: 10 horas.

Descripción: preparar una Raspberry Pi, es decir, instalar un sistema operativo y las herramientas y dependencias necesarias para realizar las pruebas sobre el *front-end* y la API.

Entradas: DOP.

Salidas/Entregables: Raspberry Pi lista para las pruebas.

Recursos necesarios: sistemas operativo, Node.js y dependencias.

Instalación de otras herramientas necesarias

Paquete de trabajo: Instalación de sistemas de desarrollo.

Duración: 4 horas.

Descripción: instalar otras herramientas para el desarrollo del TFG, como puede ser un servidor web para el blog/wiki o la base de datos.

Entradas: DOP.

Salidas/Entregables: sistemas completos y listos para el desarrollo.

Recursos necesarios: resto de herramientas necesarias y dependencias.

2.7.1.1.3. Resto de tareas de Organización y aprendizaje

Todavía quedan más tareas para completar esta primera parte y que son igualmente importantes que los dos paquetes anteriores.

Aprendizaje de nuevo lenguaje (Python)

Paquete de trabajo: Organización y aprendizaje.

Duración: 40 horas.

Descripción: búsqueda y lectura de tutoriales y realización de pequeños programas base para conseguir el aprendizaje necesario para empezar la tarea. Puede ser posible que tenga que volver a este punto durante la realización del proyecto.

Aprendizaje sobre la realización de una API *Rest*

Paquete de trabajo: Organización y aprendizaje.

Duración: 8 horas.

Descripción: búsqueda y lectura de tutoriales para la realización de una API *Rest*.

Aprendizaje sobre *MEAN stack*

Paquete de trabajo: Organización y aprendizaje.

Duración: 10 horas.

Descripción: búsqueda y lectura de tutoriales para comprender el funcionamiento de *MEAN stack*.

Creación y puesta a punto del repositorio GIT

Paquete de trabajo: Organización y aprendizaje.

Duración: 2 horas.

Descripción: crear un repositorio para tener en cualquier momento los archivos necesarios para poder programar la aplicación, además de configurar el IDE para poder usarlo.

Recursos necesarios: navegador web, IDE de desarrollo.

2.7.1.2. Realización del back-end de la aplicación (BBDD, API y Módulos)

Se he decidido partir la aplicación en *back-end* y *front-end*. El *front-end* no será más que una interfaz gráfica para que el usuario pueda usar las funcionalidades que aportará el *back-end*.

2.7.1.2.1. Realización de núcleo de la aplicación y comprobación de URL

El núcleo será la parte principal de la aplicación y será la parte que acabe llamando a los diferentes módulos.

Por otra parte, la comprobación de URL será una parte del núcleo que se encargará de la lectura de la URL introducida por el usuario y que realizará comprobaciones para aprobar, o no, su uso.

Diseño de casos de uso

Paquete de trabajo: Realización de núcleo de la aplicación y comprobación de URL.

Duración: 6 horas.

Descripción: Realizar el diseño de los casos de uso para ver las acciones de los actores de la aplicación.

Entrada: DOP.

Salidas/Entregables: Diagrama de casos de uso.

Recursos necesarios: software realizador de diagramas.

Diseño de modelo de dominio

Paquete de trabajo: Realización de núcleo de la aplicación y comprobación de URL.

Duración: 4 horas.

Descripción: realización del modelo de dominio a partir de los casos de uso.

Entrada: diagrama de casos de uso.

Salidas/Entregables: modelo de dominio.

Recursos necesarios: software realizador de diagramas.

Diseño de diagrama de clases

Paquete de trabajo: Realización de núcleo de la aplicación y comprobación de URL.

Duración: 3 horas.

Descripción: realizar el diagrama de clases para representar cada una de las clases con sus respectivos métodos y atributos.

Entrada: modelo de dominio.

Salidas/Entregables: diagrama de clases.

Recursos necesarios: software realizador de diagramas.

Diseño de diagramas de secuencia

Paquete de trabajo: Realización de núcleo de la aplicación y comprobación de URL.

Duración: 10 horas.

Descripción: realización de los diagramas de secuencia de los métodos de cada clase.

Entrada: diagrama de clases.

Salidas/Entregables: diagramas de secuencia.

Recursos necesarios: software realizador de diagramas.

Diseño de las pruebas

Paquete de trabajo: Realización de núcleo de la aplicación y comprobación de URL.

Duración: 5 horas.

Descripción: realización de las pruebas para proceder a sus comprobaciones una vez terminado este módulo.

Entrada: diagrama de secuencia.

Salidas/Entregables: diagramas de las pruebas.

Recursos necesarios: editor de texto.

2.7.1.2.2. Realización del módulo de lectura del árbol de URL

Módulo que leerá el contenido de una página mediante la URL suministrada, previamente comprobada y aprobada.

Además, este módulo será capaz de buscar el mapa web a través de la URL suministrada. De no encontrarlo, se buscarán todas las URL de las que se compone la web del usuario mediante la lectura de sus hipervínculos.

Diseño de casos de uso

Paquete de trabajo: Realización del módulo de lectura del árbol de URL.

Duración: 4 horas.

Descripción: Realizar el diseño de los casos de uso para ver las acciones de los actores de la aplicación.

Entrada: DOP.

Salidas/Entregables: Diagrama de casos de uso.

Recursos necesarios: software realizador de diagramas.

Diseño del modelo de dominio

Paquete de trabajo: Realización del módulo de lectura del árbol de URL.
Duración: 3 horas.

Descripción: realización del modelo de dominio a partir de los casos de uso.

Entrada: diagrama de casos de uso.

Salidas/Entregables: modelo de dominio.

Recursos necesarios: software realizador de diagramas.

Diseño del diagrama de clases

Paquete de trabajo: Realización del módulo de lectura del árbol de URL.
Duración: 2 horas.

Descripción: Realizar el diagrama de clases para representar cada una de las clases con sus respectivos métodos y atributos.

Entrada: modelo de dominio.

Salidas/Entregables: diagrama de clases.

Recursos necesarios: software realizador de diagramas.

Diseño de los diagramas de secuencia

Paquete de trabajo: Realización del módulo de lectura del árbol de URL.
Duración: 6 horas.

Descripción: realización de los diagramas de secuencia de los métodos de cada clase.

Entrada: diagrama de clases

Salidas/Entregables: diagrama de secuencia.

Recursos necesarios: software realizador de diagramas..

Diseño de las pruebas

Paquete de trabajo: Realización del módulo de lectura del árbol de URL.
Duración: 5 horas.

Descripción: realización de las pruebas para proceder a sus comprobaciones una vez terminado este módulo.

Entrada: diagrama de secuencia.

Salidas/Entregables: diagramas de las pruebas.

Recursos necesarios: editor de texto.

2.7.1.2.3. Realización del módulo de inyección SQL y configuración de seguridad incorrecta

Módulos que buscarán estos dos tipos de vulnerabilidades una vez leídas y conociendo el contenido de una página única.

Diseño de casos de uso

Paquete de trabajo: Realización del módulo de inyección SQL y configuración de seguridad incorrecta.

Duración: 4 horas.

Descripción: realizar el diseño de los casos de uso para ver las acciones de los actores de la aplicación.

Salidas/Entregables: Diagrama de casos de uso.

Recursos necesarios: software realizador de diagramas.

Diseño del modelo de dominio

Paquete de trabajo: Realización del módulo de inyección SQL y configuración de seguridad incorrecta.

Duración: 3 horas.

Descripción: realización del modelo de dominio a partir de los casos de uso.

Entrada: casos de uso.

Salidas/Entregables: modelo de dominio.

Recursos necesarios: software realizador de diagramas.

Diseño del diagrama de clases

Paquete de trabajo: Realización del módulo de inyección SQL y configuración de seguridad incorrecta.

Duración: 2 horas.

Descripción: realizar el diagrama de clases para representar cada una de las clases con sus respectivos métodos y atributos.

Entrada: modelo de dominio.

Salidas/Entregables: diagrama de clases.

Recursos necesarios: software realizador de diagramas.

Diseño de los diagramas de secuencia

Paquete de trabajo: Realización del módulo de inyección SQL y configuración de seguridad incorrecta.

Duración: 6 horas.

Descripción: realización de los diagramas de secuencia de los métodos de cada clase.

Entrada: Diagrama de clases.

Salidas/Entregables: Diagrama de secuencia.

Recursos necesarios: software realizador de diagramas.

Diseño de las pruebas

Paquete de trabajo: Realización del módulo de inyección SQL y configuración de seguridad incorrecta.

Duración: 5 horas.

Descripción: realización de las pruebas para proceder a sus comprobaciones una vez terminado este módulo.

Entrada: diagrama de secuencia.

Salidas/Entregables: diagramas de las pruebas.

Recursos necesarios: editor de texto.

2.7.1.2.4. Realización del resto de las tareas del *back-end*

Una vez diseñados todos los módulos, es la hora de diseñar la base de datos global de la aplicación.

Además, también es el momento de implementar cada uno de los módulos, puesto que para ello también se necesita de la base de datos.

Diseño de base de datos general

Paquete de trabajo: Realización del *back-end* de la aplicación.

Duración: 6 horas.

Descripción: realización del diseño de la base de datos que se usará en toda la aplicación.

Entradas: modelos de dominio de los cuatro subpaquetes anteriores.

Salidas/Entregables: diagrama de base de datos.

Recursos necesarios: software realizador de diagramas.

Implantar base de datos y comprobar su funcionamiento

Paquete de trabajo: Realización del *back-end* de la aplicación.

Duración: 5 horas.

Descripción: implantación de la base de datos general anteriormente diseñada y comprobar su correcto funcionamiento.

Entradas: diagrama de base de datos.

Salidas/Entregables: base de datos implementada.

Recursos necesarios: software de base de datos.

2.7.1.2.5. Implementación del código

A la hora de implementar el código necesario, se ha decidido dividirlo en dos grandes subgrupos.

El primero de ellos se trata de implementar el núcleo y todos los módulos para el funcionamiento de la aplicación; sin embargo, hace falta del segundo subgrupo.

Por lo tanto, el segundo, lo que se hará es realización de funciones en la base de datos para no tener que acceder de manera directa a los datos almacenados en la misma base de datos, con el fin de realizar una aplicación más segura.

General
Paquete de trabajo: Implementación del código. Duración: 110 horas.
Descripción: realización del núcleo y API de la aplicación. Entradas: los diagramas de clases y de secuencia de los 4 subpquetes anteriores, además de la base de datos. Salidas/Entregables: núcleo y módulos implementados. Recursos necesarios: IDE.
Funciones de la base de datos
Paquete de trabajo: Implementación del código. Duración: 40 horas.
Descripción: realización de las funciones necesarias para poder realizar peticiones a la base de datos sin tener que hacer llamadas directas a las tablas. Entradas: núcleo, módulos y base de datos implementada. Salidas/Entregables: funciones de la base de datos. Recursos necesarios: software de base de datos.

2.7.1.3. Realización del *front-end* y API de la aplicación

Esta parte será lo que vea el usuario para poder hacer uso de la aplicación en sí alojada en el servidor. Por lo tanto, se trata de desarrollar la página web.

Además, se implementará la API encargada de conectar la página web con la aplicación realizada en Python.

Diseño de casos de uso

Paquete de trabajo: Realización del *front-end* y API de la aplicación.
Duración: 8 horas.

Descripción: realizar el diseño de los casos de uso para ver las acciones de los actores de la aplicación.

Entrada: DOP.

Salidas/Entregables: diagrama de casos de uso.

Recursos necesarios: software realizador de diagramas.

Diseño de modelo de dominio

Paquete de trabajo: Realización del *front-end* y API de la aplicación.
Duración: 4 horas.

Descripción: realizar el diseño del modelo de dominio.

Entrada: diagrama de casos de uso.

Salidas/Entregables: diagrama de modelo de dominio.

Recursos necesarios: software realizador de diagramas.

Diseño de base de datos no relacional

Paquete de trabajo: Realización del *front-end* y API de la aplicación.
Duración: 3 horas.

Descripción: realizar el diseño de la base de datos no relacional que contendrá los datos necesarios de la web para su funcionamiento con el motivo de no realizar peticiones a la base de datos principal de la aplicación.

Entrada: diagrama de modelo de dominio.

Salidas/Entregables: diagrama de base de datos no relacional.

Recursos necesarios: software realizador de diagramas.

Diseño de esquema REST

Paquete de trabajo: Realización del *front-end* y API de la aplicación.
Duración: 5 horas.

Descripción: realizar el diseño del esquema de API *Rest*.

Entrada: diagramas de casos de uso y de modelo de dominio.

Salidas/Entregables: esquema REST.

Recursos necesarios: software realizador de diagramas.

2.7.1.3.1. Diseño de la interfaz gráfica

Cabe recordar que, hasta ahora, no ha hecho falta de una interfaz de usuario. Sin embargo, la página web será lo que vea el usuario y habrá que diseñarla de tal manera que cualquier usuario pueda acceder a ella y que no tenga ningún problema a la hora de usarla.

Prototipado a papel

Paquete de trabajo: Diseño de la interfaz gráfica.

Duración: 2 horas.

Descripción: diseñar la interfaz gráfica mediante un prototipo rápido a papel.

Salidas/Entregables: prototipo a papel.

Recursos necesarios: papel, lápiz, regla y mucha goma de borrar.

Prototipado a ordenador

Paquete de trabajo: Diseño de la interfaz gráfica.

Duración: 12 horas.

Descripción: traspasar el prototipo a papel al ordenador mediante una herramienta de prototipado de interfaces. Este prototipo será el que se use para realizar los distintos estudios posteriores.

Entrada: prototipo a papel.

Salidas/Entregables: prototipo a ordenador.

Recursos necesarios: software de diseño de interfaces de usuario.

Realización de encuestas sobre el prototipado

Paquete de trabajo: Diseño de la interfaz gráfica.

Duración: 10 horas.

Descripción: hablar con potenciales usuarios, mostrarles el prototipo y sacar conclusiones sobre el mismo.

Entrada: prototipo a ordenador.

Salidas/Entregables: encuestas.

Corrección del prototipo

Paquete de trabajo: Diseño de la interfaz gráfica.

Duración: 6 horas.

Descripción: corrección del prototipo a ordenador.

Entrada: prototipo a ordenador y encuestas.

Salidas/Entregables: prototipo a ordenador final.

Recursos necesarios: software de diseño de interfaces de usuario.

2.7.1.3.2. Implementación del código

Hasta aquí, se ha habra realizado los análisis y diseño de la página web y API, además del prototipo de la interfaz gráfica. Por lo tanto, lo que falta es la implementación de las mismas.

Realización del código base

Paquete de trabajo: Implementación del código.

Duración: 40 horas.

Descripción: realización del código base de la web: la API, los controladores y el modelo.

Entradas: diagramas del *front-end* y de la API.

Salidas/Entregables: API y parte del *front-end* implementados.

Recursos necesarios: IDE.

Realización de la interfaz

Paquete de trabajo: Implementación del código.

Duración: 10 horas.

Descripción: realización de la vista de la página web.

Entradas: prototipo a ordenador final y la implementación anterior.

Salidas/Entregables: API y *front-end* completamente implementados.

Recursos necesarios: IDE.

2.7.1.4. Implantación (API, Web y Módulos)

Tareas que se componen de montar el servidor, cargar la página web y realizar las pruebas de la propia interfaz web en busca de posibles fallos de diseño o de alguna vulnerabilidad explotable.

Instalación del sistema

Paquete de trabajo: Implantación.

Duración: 2 horas.

Descripción: Instalación y puesta a punto del sistema web.

Entradas: front y back-end implementados.

Recursos necesarios: acceso a un servidor web y un software FTP para la subida a la Raspberry Pi.

Precedencias: Realización de los *front-end* y *back-end*.

Pruebas del sistema

Paquete de trabajo: Implantación.

Duración: 4 horas.

Descripción: probar la página web para comprobar el correcto funcionamiento de la misma.

Recursos necesarios: navegador web.

Precedencias: Instalación del sistema.

2.7.1.5. Wiki y Blog

Montar una wiki y un blog mediante alguna de las herramientas open source disponibles ya en el mercado. Habrá que adaptarla, implantarla y configurarla.

Adaptar base de datos para funcionamiento con front-end

Paquete de trabajo: Wiki y Blog.

Duración: 8 horas.

Descripción: adaptar las bases de datos de la Wiki y Blog para que sean compatibles con la base de datos no relacional de la API.

Entrada: API y los esquemas de base de datos no relacional.

Recursos necesarios: IDE.

Precedencias: Implantación del sistema.

Implantación

Paquete de trabajo: Wiki y Blog.

Duración: 1 hora.

Descripción: implantar la Wiki y el Blog.

Recursos necesarios: Blog y Wiki modificados, software de servidor web.

Configuración

Paquete de trabajo: Wiki y Blog.

Duración: 4 horas.

Descripción: configurar la Wiki el Blog.

Entrada: Wiki y Blog implantados.

Recursos necesarios: navegador web.

2.7.1.6. Leyes

El desconocimiento de una ley no te exime de su cumplimiento. Es importante registrarse por la legislación actual para no infringir ninguna ley en vigor con la aplicación que se va a realizar en este TFG.

Lectura de la legislación actual
Paquete de trabajo: Leyes.
Duración: 32 horas.
Descripción: lectura de la legislación actual en busca de leyes que pueda que esté infringiendo.

Implementación de la legislación actual
Paquete de trabajo: Leyes.
Duración: 10 horas.
Descripción: modificar o añadir lo necesario para cumplir con la legislación actual.

2.7.1.7. Documentación

Es importante la realización de los manuales necesarios. Se realizarán dos manuales; el primero contará la puesta a punto del servidor y la instalación propia de la aplicación; y un segundo manual explicará la interfaz de usuario.

Manual técnico
Paquete de trabajo: Documentación.
Duración: 8 horas.
Descripción: redactar un escrito para explicar el mantenimiento de la web y su instalación.
Salidas/Entregables: Manual técnico.
Recursos necesarios: software editor de textos.

Manual de usuario
Paquete de trabajo: Documentación.
Duración: 8 horas.
Descripción: redactar un escrito para los usuarios en el que se explicará cómo se usa el front-end de la aplicación.
Salidas/Entregables: Manual de usuario.
Recursos necesarios: software editor de textos.

Realización de la memoria

Paquete de trabajo: Documentación.

Duración: 40 horas.

Descripción: redactar un escrito para explicar todo lo realizado para llegar hasta completar el TFG.

Salidas/Entregables: Memoria.

Precedencias: todas y cada una de las tareas para realizar este TFG.

Recursos necesarios: software editor de textos.

Rellenar Wiki con los datos de la memoria

Paquete de trabajo: Documentación.

Duración: 10 horas.

Descripción: Redactar un escrito para los usuarios de la aplicación en el cual se explicará cómo se usa dicha aplicación.

Entrada: Memoria.

Recursos necesarios: navegador web.

2.8. Planificación temporal

Una vez se conocen las tareas a realizar, hay que calcular cómo se van a distribuir en el tiempo. El tiempo de dedicación previsto se representa mediante un diagrama de Gantt, como se puede observar en las Figuras 5 y 6.

Se ha estimado una carga de trabajo semanal de 40 horas, como si de un empleo real se tratara. Los fines de semana, en principio, no se trabajaría en el trabajo, a no ser que haga falta terminar alguna tarea pendiente. Cabe mencionar que existen días en los que no se va a poder trabajar debido a eventos y días festivos.

Por lo tanto, poniendo que el trabajo comienza el 1 de julio de 2015, éste terminaría, si se sigue el plan previsto, el 25 de noviembre de 2015.

Hablando sobre la manera de distribución de las tareas, al contar el equipo con un único componente, no se puede empezar una tarea sin haber terminado otra, de ahí que no existan tareas que se solapen.

2.8.1. Diagrama Gantt

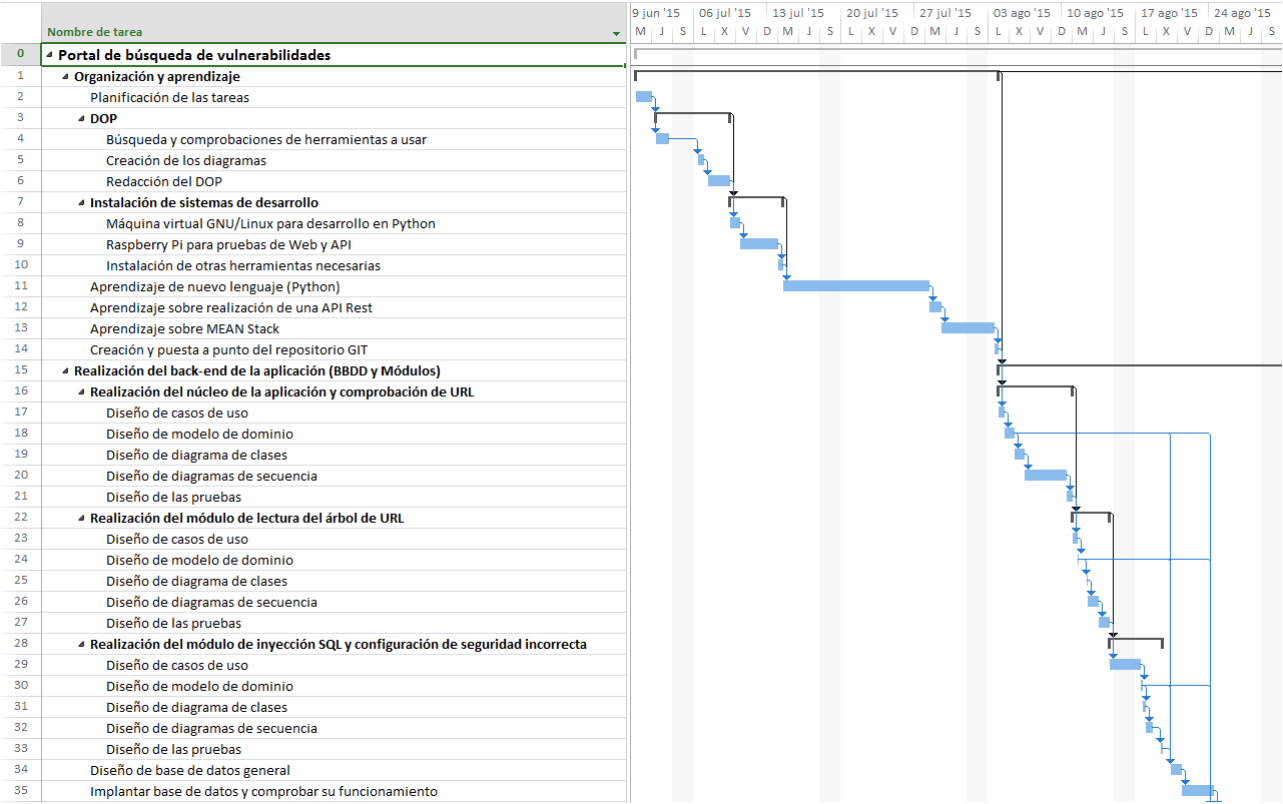


Figura 5: Diagrama Gantt.

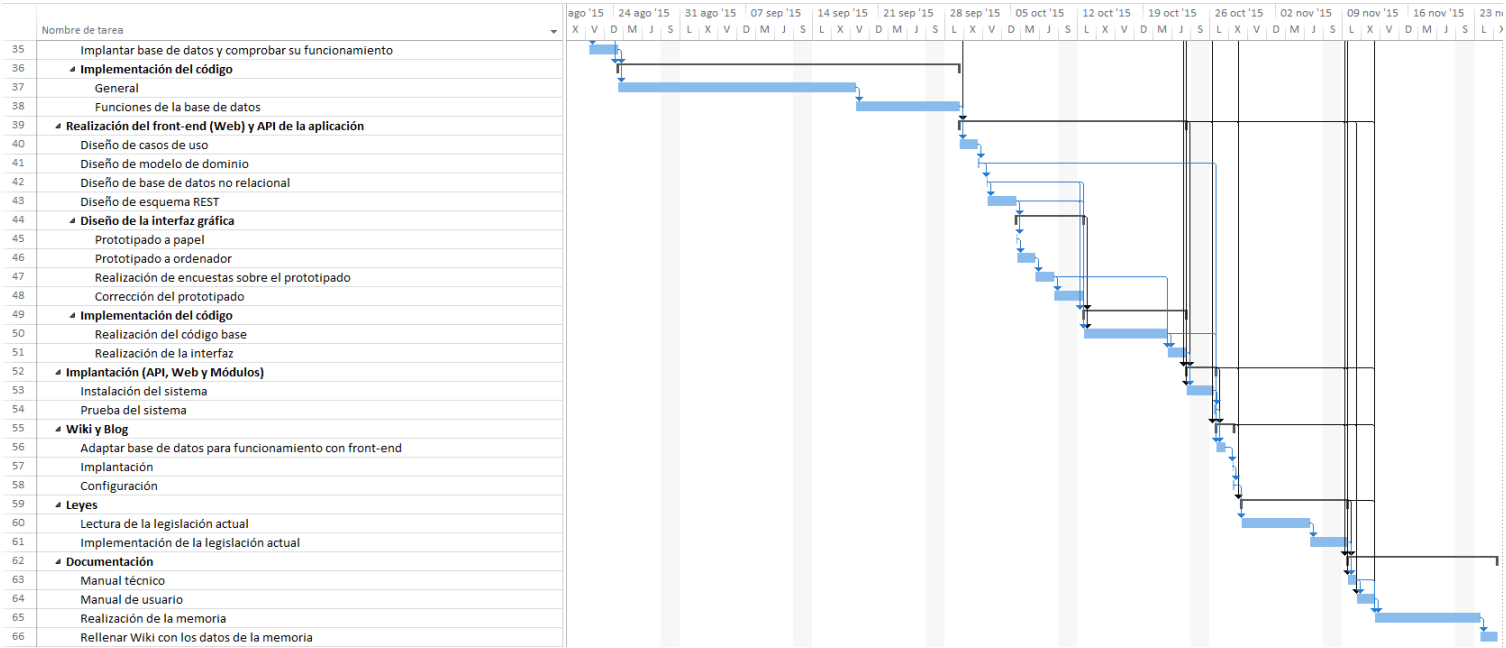


Figura 6: Diagrama Gantt.

2.9. Evaluación económica

Aunque la idea es terminar montando una empresa, es posible que este proyecto sea liberado como programa académico o vendido a una empresa mayor. Por lo tanto, la mejor manera de calcular su valor será la de proponer una cantidad de dinero adecuada por mes trabajado en el mismo, como si de una empresa se tratara, puesto que la forma en la que se va a realizar el mismo imitará dicha forma de trabajo.

Además, no es menos importante tener en mente los materiales y herramientas que se van a utilizar, puesto que son gastos asociados a la hora de realizar el propio proyecto. A su vez, también se deben contar los gastos indirectos de luz, teléfono e Internet.

2.9.1. Mano de obra

Como el trabajo realizado se trata de Analista-Programador, se han empleado como base el sueldo mínimo (establecido por el convenio de 2009) presente en las tablas publicadas para el convenio de 2009 en el BOE a fecha de 4 de abril de 2009.

Por lo tanto, el sueldo base es de 1.539,69 € (Anexo III, Tabla salarial y de plus convenio 2009 [3]) mensuales. Como el trabajo abarca casi 5 meses, el sueldo alcanza la cantidad de 7698,45 €.

2.9.2. Software

La mayoría del proyecto se ha realizado en entornos de código abierto, salvo excepciones en el que se ha usado un entorno Microsoft Windows. Sin embargo, las licencias de las herramientas de pago a utilizar durante la realización del mismo no han tenido que ser compradas, gracias a las licencias del programa de Microsoft DreamSpark y la licencia de Visual Paradigm de la universidad.

2.9.3. Hardware

Las herramientas hardware empleadas durante el desarrollo y la vida útil estimada de todos ellos son los mostrados en la Tabla 3:

Dispositivo	Vida útil estimada	Precio
Ordenador de sobremesa	36 meses	~1200 €
Ordenador portátil	60 meses	500 €
Nexus 5	36 meses	350 €
Raspberry Pi	60 meses	80 €

Tabla 3: Hardware usado en el desarrollo.

Para realizar los cálculos necesarios para hallar el gasto realizado a los diferentes dispositivos, es necesario el cálculo de la amortización mensual de cada uno de los aparatos, tal y como se ve en la siguiente fórmula:

$$\text{Amortización mensual} = \frac{\text{Precio}}{\text{Vida útil estimada}}$$

Contando con la amortización mensual de cada dispositivo, se puede calcular la amortización de cada una durante la duración de la realización del proyecto. Para ello, se seguirá la siguiente fórmula:

$$\text{Amortización del proyecto} = \frac{\text{Amortización mensual}}{\text{Duración del proyecto}}$$

Con estas dos fórmulas, se puede hallar el gasto asociado de cada herramienta durante el transcurso del proyecto. Los resultados se pueden observar en la Tabla 4.

Herramienta	A. mensual	A. durante el proyecto
Ordenador de sobremesa	~33,33 €	~166,66 €
Ordenador portátil	~8,33 €	~41,66 €
Nexus 5	~9,72 €	~48,61 €
Raspberry Pi	~1,33 €	~6,66 €
	Total	~263,59 €

Tabla 4: Amortizaciones de las herramientas hardware.

2.9.4. Gastos indirectos

Los gastos indirectos son aquellos que no están dentro de la propia actividad productiva.

De entre todos los posibles gastos, los que afectan para el desarrollo de este proyecto se detallarán a continuación.

2.9.4.1. Luz

Para concretar el gasto de luz, es necesario conocer la potencia de cada herramienta hardware. Estos datos se encuentran en la Tabla 5.

Herramienta	Potencia
Ordenador de sobremesa	máx. 620 W
Ordenador portátil	máx. 65 W
Nexus 5	aprox. 5 W
Raspberry Pi	aprox. 3 W
Total	máx. 693 W

Tabla 5: Potencia de las herramientas hardware.

En España, el precio medio del kWh es de 0,1815€⁵ en el momento de realizar este documento.

Para calcular el consumo, se contará que todos los dispositivos están encendidos en todo momento durante el desarrollo de este proyecto.

$$\text{Gasto luz diario} = \text{precio kWh} * \text{potencia total kW} * 8 \text{ horas diarias}$$

$$\text{Gasto luz diario} = 0,1815 * 0,693 * 8 = 1,006236 \text{ €}$$

$$\text{Gasto luz mensual} = 1,006236 * 22 \text{ días laborales} = 22,137192 \text{ €}$$

$$\text{Gasto luz proyecto} = 22,13719 * 5 \text{ meses} = 110,68596 \text{ €}$$

⁵Precio del kWh: <http://tarifasgasluz.com/faq/precio-kwh/espana>.

2.9.4.2. Internet

Es necesario Internet para la realización de este proyecto. Además, se debe contar la tarifa de Internet móvil, que se usará en caso de perder la conexión física.

- Internet: 50 € mensuales, lo que da un total de 250 €.
- Internet móvil: 6,90 € mensuales, lo que da un total de 34,5 €.

2.9.5. Gastos totales

Tras realizar todos los gastos, ahora es posible hallar el gasto total, que se podrá ver en la Tabla 6.

Concepto	Gasto en €
Mano de obra	7698,45 €
Hardware	263,59 €
Gastos indirectos	395,19 €
Total	8357,23 €

Tabla 6: Gasto total.

Por último, se puede observar que el gasto total se eleva a 8357,23 €. En la siguiente subsección, se expondrán diferentes maneras para tratar de recuperar, mínimo, la inversión realizada.

2.9.6. Contexto de negocio

En esta etapa tan temprana del proyecto, es posible que no se contemple con total seguridad un contexto de negocio que se pueda hacer realidad una vez terminado el mismo. Por lo tanto, es posible que, tras la terminación de este proyecto, se decida liberar todo lo realizado mediante licencia *GNU General Public License v2.0*⁶.

Sin embargo, conviene aclarar las ideas de negocio que pueden funcionar viendo, de esta manera, las maneras de poder recuperar la inversión que se va a realizar y poder obtener algún beneficio.

⁶Más en: <http://www.gnu.org/licenses/old-licenses/gpl-2.0.html>.

2.9.6.1. Modelo *freemium*

La idea bajo este modelo es la de permitir el uso de la página web a cualquier usuario registrado pero que, para que pueda optar a hacer uso de más, o todas, las opciones de la propia página web, este usuario tendrá que realizar un pago mensual.

En un principio, la opción básica gratuita consistiría en:

- Opción a comentar en el blog.
- Opción a colaborar en la wiki.
- Poder buscar las vulnerabilidades sobre inyección SQL y *Cross-Site Request Forgery* de una única URL por cada vez, es decir, el usuario no podrá buscar vulnerabilidades de una web si ya está en ejecución una búsqueda anterior.

Los usuarios de pago, podrán contar con las mismas opciones que los usuarios registrados básicos; sin embargo, podrán buscar las mismas vulnerabilidades dentro del mapa web que la propia aplicación capturará a través de la dirección escrita por el usuario y podrán realizar más de una búsqueda simultánea.

Supongamos que existen 1000 usuarios dispuestos a pagar por el uso de más opciones. Al ser un servicio nuevo, no conviene poner un precio alto, puesto que puede perjudicar a los posibles clientes. Por lo tanto, supongamos un precio de 10€ anuales.

A partir de este importe anual, se puede calcular el retorno sobre la inversión, llamado ROI por sus siglas en inglés *Return on Investment*, tras el primer año de actividad:

$$ROI = \frac{10000}{8357,23} * 100 = 119,66 \%$$

Como se puede observar en el ROI, se obtendría un buen beneficio y se recuperaría la inversión inicial realizada.

2.9.6.2. Cobro por el soporte

Otra idea se trataría de abrir todas las opciones que ofrece la página web a todos los usuarios. El negocio estaría en el cobro por dar soporte personalizado a aquellos usuarios que lo pidan.

El soporte estaría enfocado tanto en la explicación del uso de la herramienta que se realizará en este TFG, como en la enseñanza de diferentes métodos para prevenir la explotación de posibles vulnerabilidades y ayudar en la codificación para tratar de mejorar la seguridad en la aplicación web de cada uno de los posibles clientes.

Puesto que un soporte personalizado puede ser costoso, una cifra adecuada podría ser de 100€ mensuales por cliente, pudiendo dar soporte a 2 clientes mensuales como máximo.

Tras el primer año, el ROI quedaría en:

$$ROI = \frac{2400}{8357,23} * 100 = 28,72\%$$

No se recuperaría la inversión inicial realizada.

2.9.6.3. Combinación: modelo *freemium* y cobro por el soporte

Una tercera posibilidad se podría tratar de la realización de una mezcla entre las dos opciones anteriores, es decir, tener dos categorías de usuarios y dar soporte a aquellos que lo soliciten.

2.9.6.4. Liberar el código

La última opción sería la de liberar el código mediante la licencia *GNU General Public License v2.0*. Por lo tanto, no se podría recuperar la inversión, tal y como se muestra en el siguiente cálculo del ROI:

$$ROI = \frac{0}{8357,23} * 100 = 0\%$$

2.10. Riesgos

Si bien es cierto que no podemos decir que nuestra área de trabajo conlleva serios riesgos para la salud, tampoco está exenta de los mismos. A continuación, se procede a describir los riesgos que se consideran que puedan amenazar la planificación realizada.

Cabe mencionar que las jornadas laborales son de 8 horas, por lo tanto, al poner 1 día a la hora de realizar los cálculos oportunos, cada uno equivale a las 8 horas mencionadas anteriormente.

2.10.1. Problemas con Python

Al estar realizando parte de este trabajo utilizando el lenguaje de programación Python, el cual no ha sido utilizado previamente durante la carrera, es posible que, aún habiendo estudiado este lenguaje antes de comenzar a programar, se presenten problemas.

Prevención

- Prepararse correctamente para la realización de este trabajo.

Plan de contingencia

- Buscar información sobre Python.
- Encontrar ejemplos de uso acerca del problema obtenido en la versión de Python finalmente escogida.

Probabilidad

- Encontrar un problema y dar con la solución adecuada. Media: 40 %.

Impacto

- Solución no aparente. $0,4 \times 1 \text{ día} = 3 \text{ horas}$. Impacto medio.

2.10.2. “Quedarse en blanco”

Todos sabemos que llegará un momento en el que uno no sabe cómo continuar con una cierta tarea. En el caso que nos ocupa en este trabajo, esta situación se da por encontrarte frente a un problema que a simple vista no tiene una solución sencilla aparente.

Prevención

- Prepararse correctamente para la realización de este trabajo.

Plan de contingencia

- Buscar información sobre el problema en los libros sobre la temática o en Internet.
- En caso de que el problema sea fácil de solucionar, se continuará con el trabajo arreglando, primero, el problema.
- En caso de que el problema no tenga una solución aparentemente sencilla o las soluciones encontradas no arreglen el problema, se pasará a realizar otra tarea, siempre que se pueda. De no ser posible, descansar la mente e, incluso, esperar al día siguiente para continuar.

Probabilidad

- Fácil solución. No compromete.
- Solución no aparente. Baja: 15 %.

Impacto

- Fácil solución. No hay impacto.
- Solución no aparente. $0,15 \times 1 \text{ día} = 1,2 \text{ horas}$. Impacto bajo.

2.10.3. Problemas con Raspberry Pi

El uso de este dispositivo se ha pensado para realizar pruebas en un entorno más real. Sin embargo, es posible que ocurran errores por el uso de MongoDB, puesto que sus librerías no están del todo optimizadas para este dispositivo.

Prevención

- Instalar únicamente lo necesario en el dispositivo para que sea capaz de servir el portal web.

Plan de contingencia

- Buscar información sobre el problema en Internet.
- En caso de que el problema sea fácil de solucionar, se continuará con el trabajo arreglando, primero, el problema.
- En caso de que el problema no tenga una solución, se abandonará el uso del dispositivo y se dará servicio a la web usando la propia máquina virtual del núcleo en Python.

Probabilidad

- Fácil solución. Baja: 15 %.
- Solución no aparente. No compromete, estarán todas las dependencias ya instaladas.

Impacto

- Fácil solución. $0,15 \times 1 \text{ día} = 1,2 \text{ horas}$. Impacto bajo.
- Solución no aparente. No hay impacto.

2.10.4. Enfermedad o lesión

Consiste en la indisponibilidad por una causa médica, ya sea derivada del trabajo o por causas ajenas al mismo.

Prevención

- Derivadas del trabajo:
 - Mantener una postura correcta a la hora de trabajar.
 - Mantener la distancia correcta frente al monitor.
 - Realizar descansos de 10 minutos cada 2 horas.
- No derivadas del trabajo:
 - Evitar actividades deportivas de riesgo.
 - Abrigarse en invierno.

Plan de contingencia

- Acudir al médico para diagnosticar el alcance de la lesión o enfermedad.
- En caso de que el diagnóstico sea favorable, se continuará con el trabajo pero descansando más a menudo.
- En caso de que el diagnóstico sea muy desfavorable, se aplazará el tiempo el tiempo necesario para recuperarse y poder continuar con el trabajo.

Probabilidad

- Diagnóstico favorable. Baja: 15 %.
- Diagnóstico desfavorable. Muy baja: 5 %.

Impacto

- Diagnóstico favorable. $0,15 \times 3 \text{ días} = 3,6 \text{ horas}$. Impacto medio.
- Diagnóstico desfavorable. $0,05 \times 10 \text{ días} = 4 \text{ horas}$. Impacto medio.

2.10.5. Inutilización del equipo informático

Daños imprevistos, tanto de hardware como de software, debido a acciones relacionadas con el trabajo, transporte de material o agentes externos.

Prevención

- Mantener el equipo en unas condiciones ambientales adecuadas.
- Realizar una revisión mensual del equipo.
- Realizar el proyecto sobre máquinas virtuales en vez de hacerlo directamente sobre las físicas.
- Realizar copias de seguridad diarias de dichas máquinas virtuales.
- Evitar usar el equipo de trabajo para otros usos que no sean los propios del trabajo.

Plan de contingencia

- Hardware:
 - Si lo que falla es una pieza del equipo fácilmente reemplazable, como un disco duro o memoria RAM, se compra ese mismo día y se restaura el sistema.
 - Si es una avería más grave, se utilizará el portátil o un equipo disponible en la universidad hasta que se reciba el nuevo hardware o se arregle el que ya se tenía.
- Software: Se procederá a restaurar la última copia de seguridad guardada y se procederá a realizar el trabajo que se haya perdido.

Probabilidad Baja: 10 %.

Impacto $0,10 \times 1 \text{ día} = 0,8 \text{ horas}$. Impacto bajo.

2.10.6. Pérdida de la conexión a Internet

Consiste en la pérdida de conexión a Internet por parte de alguno de los componentes del grupo. Puede acarrear retrasos respecto a los plazos de entrega previstos si uno o varios miembros del equipo se ven privados de este servicio, impidiéndoles buscar información o compartir archivos con los compañeros.

Prevención

- Contratar la conexión a Internet con una compañía fiable.
- Colocar el router en un entorno alejado de posibles golpes o incidencias.

Plan de contingencia

- Intentar conectar a alguna red WiFi disponible.
- Si el paso anterior no ha funcionado, usar el móvil como router, conocido como *tethering*.
- Si todo lo anterior no ha funcionado, esperar al día siguiente y realizar el trabajo en la universidad o en un lugar adecuado para ello.

Probabilidad Muy baja: 5 %.

Impacto $0,05 \times 1 \text{ día} = 0,4 \text{ horas}$. Impacto muy bajo.

3. Antecedentes

El punto a explorar en este capítulo es el de exponer aquellas aplicaciones existentes que tienen una idea similar a la aplicación que se va a realizar. No es fácil empezar un desarrollo y encontrar aplicaciones similares a lo que se va a realizar; sin embargo, todavía es más difícil tener una idea revolucionaria que sea pionera y que sea en la que luego se base la competencia.

Si bien existen varios ejemplos, se han escogido tres aplicaciones por ser los que más se asemejan al proyecto que se va a realizar o, bien, por estar enfocadas al mismo tipo de usuarios potenciales.

3.1. PunkSPIDER

PunkSPIDER⁷ se trata de un motor de búsqueda de vulnerabilidades *open source* con una idea similar a la que se pretende conseguir con este proyecto. Está desarrollado por la empresa Hyperion Gray⁸, que se dedica a la investigación y al desarrollo en entornos de la seguridad web.

No cuenta con blog ni wiki, por lo que está enfocada a desarrolladores más veteranos o aquellos que saben lo que están buscando. Tampoco requiere registro alguno y su interfaz web está un tanto desfasada a día en el que se ha realizado esta descripción.

Las vulnerabilidades que busca son las siguientes y se comentarán aquellas que este proyecto no vaya a contemplar:

- *Blind SQL injection*.
- Inyección SQL.
- *Cross-site scripting*⁹ (XSS). Se trata de ataques que inyectan *scripts* maliciosos en páginas web.
- *Path traversal*¹⁰. Se trata de ataques que intentan el acceso a directorios superiores de la raíz de la aplicación web y que, en principio, no deberían ser accesibles.

⁷Más información en: <https://www.punkspider.org/>.

⁸Web de la empresa: <http://www.hyperiongray.com/>.

⁹XSS: [https://www.owasp.org/index.php/Cross-site_Scripting_\(XSS\)](https://www.owasp.org/index.php/Cross-site_Scripting_(XSS)).

¹⁰Path traversal: https://www.owasp.org/index.php/Path_Traversal.

- *Mail header injection*. Se trata de un ataque similar al de inyección SQL en el que, aprovechando un formulario de envío de correo, se inyecta código para enviar un correo a alguien que no es el que estaba programado en la página web.
- *Operating system command injection*¹¹. Se trata de un ataque que ejecuta comandos propios del sistema operativo a través de una aplicación vulnerable.
- *XPath injection*¹². Similar a la SQLi. Mediante el envío de información intencionalmente mal formada, un atacante puede averiguar cómo se estructuran los datos XML, o acceder a datos a los que no se suelen tener acceso.

Se pueden marcar las vulnerabilidades que se quieren buscar. Además, cuenta con la opción de buscar todas las vulnerabilidades marcadas o hacer que pare cuando alguna de las mismas sea encontrada.

Se ha realizado una búsqueda usando una página web creada específicamente para este tipo de pruebas, puesto que es vulnerable en diversos tipos de fallos, y es usada para afinar las búsquedas: <http://crackme.cenzic.com/kelev/view/home.php>.

La herramienta parece que no funciona, se ha realizado la búsqueda con la web anterior y únicamente muestra el mensaje de “buscando”. Poniendo el portal de Google, no encuentra fallos, por lo que no se pueden sacar conclusiones con respecto a su funcionamiento.

¹¹OSCi: https://www.owasp.org/index.php/OS_Command_Injection.

¹²XPi: https://www.owasp.org/index.php/XPATH_Injection.

3.2. Metasploit

Metasploit¹³ se trata de una aplicación multiplataforma específica para realizar pruebas de *pentesting*.

La principal diferencia es que se trata de una aplicación que se debe instalar de manera local y que se puede cargar con todo tipo de módulos para ejecutar diferentes tipos de *exploits*.

Tiene dos tipos de licencia:

1. Versión Pro, funcional gratuitamente durante 14 días.
2. Versión de la comunidad gratuita, limitada en búsquedas y realizables con un *exploit* a la vez.

Es una herramienta muy potente y usada por todo tipo de expertos en seguridad. Desde luego, es una aplicación mucho más potente que la que se pretende realizar en este proyecto.

3.3. OWASP Zed Attack Proxy Project

OWASP Zed Attack Proxy¹⁴ es una aplicación multiplataforma, creada por OWASP, para la realización de pruebas de penetración sobre búsquedas de vulnerabilidades en aplicaciones web.

Está diseñada para su uso por gente con diferentes rangos de conocimiento de seguridad y es ideal para desarrolladores y para aquellos que deseen empezar a realizar pruebas de penetración.

Provee escáneres automatizados, al igual que otro tipo de herramientas, que permiten encontrar vulnerabilidades de seguridad.

¹³Más sobre Metasploit en: <http://www.metasploit.com/>.

¹⁴Más acerca de OWASP ZAP: https://www.owasp.org/index.php/OWASP_Zed_Attack_Proxy_Project.

4. Captura de requisitos

En este capítulo se presentará la captura de requisitos. La captura de requisitos es el primer paso para poder realizar una buena aplicación y se podría considerar como uno de los pasos más importante en el proceso de realización de cualquier aplicación.

Un buen análisis de este punto supondrá la mejor elección de las herramientas necesarias que se adapten a los requerimientos actuales y futuros.

4.1. Requisitos funcionales

En este apartado, se expondrán los requisitos funcionales de la aplicación. Estos requisitos definen funciones del sistema o sus componentes. Por lo tanto, estos requisitos consistirán en:

- La página web debe permitir a cualquier usuario consultar su contenido.
- Sólo los usuarios registrados y conectados en la página web podrán buscar vulnerabilidades.
- La API estará preparada para contener la existencia de dos o más perfiles de usuarios registrados. Dependiendo del perfil, un usuario podrá realizar unas acciones u otras.
- A falta de concretar el modelo de negocio, todos los usuarios podrán buscar vulnerabilidades del árbol web de la URL introducida. Como contrapartida, para no saturar del todo a la API y servidor, cada usuario sólo podrá realizar una única búsqueda de manera simultánea.

Blog y wiki:

- Tanto el blog como la wiki deben permitir a cualquier usuario consultar su contenido.
- Sólo los usuarios registrados y conectados podrán escribir comentarios o añadir/modificar contenido en la wiki.

Además, para tener el proyecto lo más controlado posible, y en previsión de lo que los usuarios puedan ser capaces de hacer con él, se contará con un *log* que guarde todos los movimientos de los mismos.

Por último, y como punto más importante, el proyecto no causará ningún desperfecto en las páginas en las que los usuarios pidan la búsqueda de vulnerabilidades. Por lo tanto, **el proyecto será capaz de buscar las vulnerabilidades descritas pero no aprovecharse de ellas**.

4.2. Requisitos no funcionales

Los requisitos no funcionales definen requisitos que se deben cumplir siendo independientes de las funcionalidades que se van a implementar para el desarrollo.

- La web debe ser accesible, de forma que cualquier usuario pueda utilizar sus funcionalidades sin un periodo de adaptación.
- El sistema debe ser estable.
- El proyecto se debe ajustar totalmente a la legalidad española.

4.3. Jerarquía de actores

La aplicación no tiene más que dos actores, como se ve en la Figura 7.



Figura 7: Jerarquía de actores.

- **Usuario anónimo.** Representa a todo usuario que llega a la página web y no se identifica.
- **Usuario identificado.** Representa a aquellos usuarios que han decidido acceder con sus credenciales a la página web.

4.4. Casos de uso

Los casos de uso se emplean como una representación gráfica de las funcionalidades del sistema. Un caso de uso representa una acción del sistema, por consiguiente, el modelo de casos de uso mostrará cómo debería interactuar el sistema con el usuario para conseguir su objetivo. También permite definir y representar los requerimientos necesarios por la aplicación.

Se trata, por ende, de una representación de la secuencia de acciones entre un sistema y sus actores, respondiendo a un evento que inicia uno de los actores principales.

4.4.1. *Front-end*

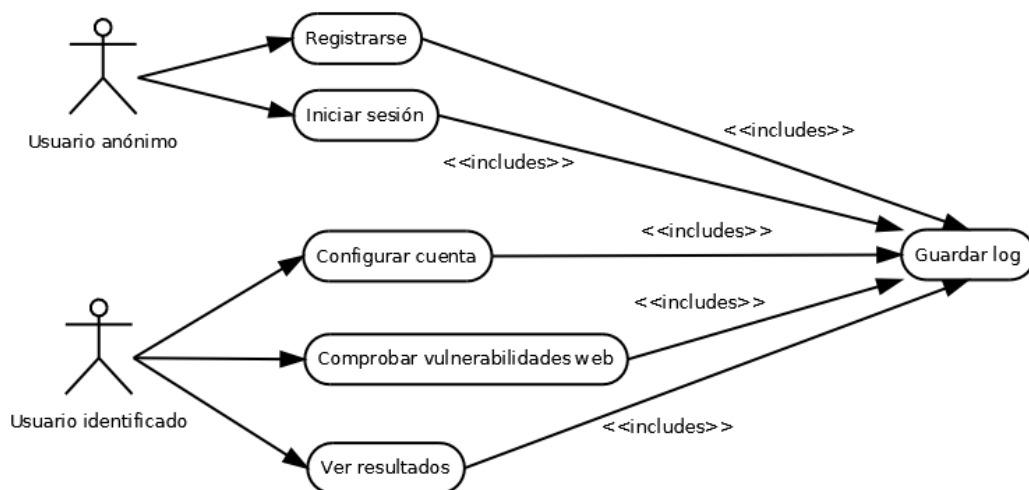


Figura 8: Casos de uso del *front-end*.

Para el usuario anónimo, existen los siguiente casos de uso, que se pueden ver en la Figura 8:

- **Registrarse.** Permite a un usuario anónimo registrarse en la página web con sus credenciales.
- **Iniciar sesión.** Permite a un usuario identificarse con sus credenciales.

Mientras tanto, un usuario identificado cuenta con otros casos de uso, que son:

- **Configurar cuenta.** Permite a un usuario cambiar datos sobre sus credenciales registradas en la página.
- **Comprobar vulnerabilidades web.** Permite a un usuario buscar vulnerabilidades mediante la inserción de una URL.
- **Ver resultados.** Permite a un usuario acceder a los resultados de otras búsquedas realizadas por él mismo anteriormente.

A su vez, todos los casos de uso incluyen el caso de uso **Guardar log**, que guardará un registro en la página de todo lo que haga cualquier usuario que acceda a la misma, por temas de seguridad.

4.4.2. *Back-end*

La Figura 9 muestra el único caso de uso que tiene el núcleo, además de los subcasos de los que hace uso.

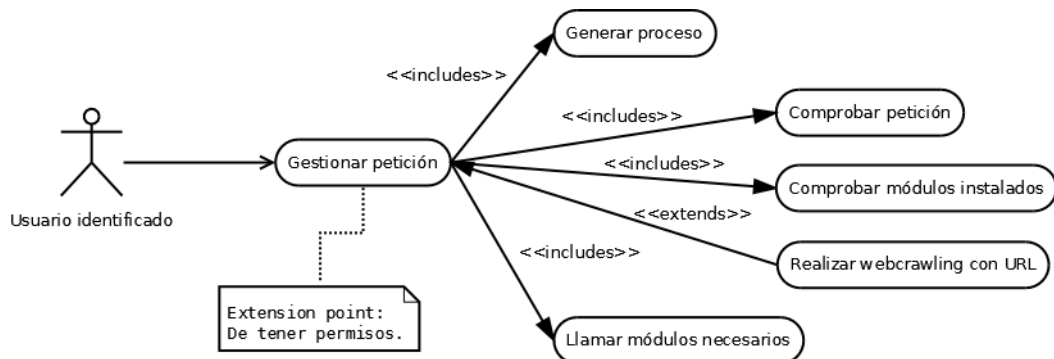


Figura 9: Casos de uso del *back-end*.

Lo que realizará el núcleo es gestionar la petición recibida por el usuario identificado, que éste realizará mediante la API. Para ello, contará con los siguientes subcasos para poder completar dicha gestión:

- **Generar proceso.** Generará un número de proceso único que identificará la petición recibida.
- **Comprobar URL.** Comprobará si la petición recibida es correcta.

- **Comprobar módulos instalados.** Comprobará si el/los módulo/s a usar está/n operativo/s o no.
- **Realizar *webcrawling* con URL.** Buscará todas las páginas de las que se compone el sitio web suministrado. Únicamente se realizará si el usuario tiene los permisos necesarios para hacerlo.
- **Llamar módulos necesarios.** Se encargará de llamar a los módulos que buscan las propias vulnerabilidades.

El diagrama de casos de uso del módulo de inyección SQL, Figura 10, incluye un único caso de uso que, a su vez, incluye dos subcasos.

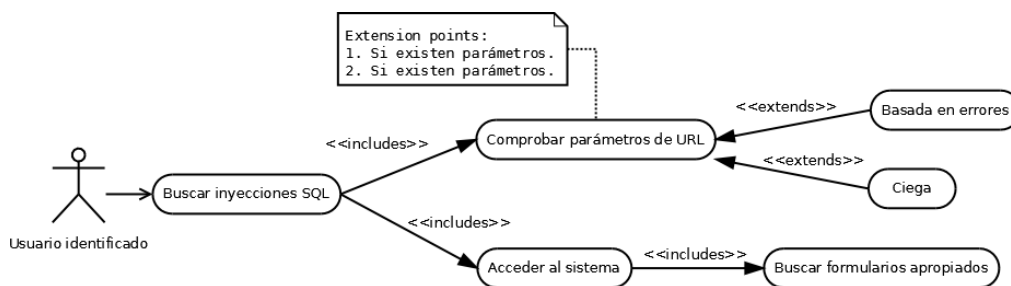


Figura 10: Casos de uso del módulo de inyecciones SQL.

- **Comprobar parámetros de URL.** Comprobará si existen parámetros en la URL suministrada por el usuario. De haberlas, se buscarán si existen posibles inyecciones SQL mediante los dos subcasos, uno por cada tipo de inyección, de uso de este mismo subcaso.
- **Acceder al sistema.** Tratará de acceder al sistema, siempre y cuando exista un formulario para ello.

Por último, queda el módulo CSRF, al que se le ha llamado “configuración de seguridad incorrecta”, del que se puede encontrar su diagrama en la Figura 11.

Estaba previsto que los dos tipos de búsquedas se realizasen bajo un único módulo, pero se ha decidido su separación en dos módulos, puesto que son dos tipos de búsquedas diferentes, aunque parte de la lógica sea similar (búsqueda de formularios). Esta separación hará que los diagramas, y los propios módulos, sean más simples.



Figura 11: Casos de uso del módulo CSRF.

Se puede observar que en los casos de uso de las Figuras 10 y 11 se incluyen el subcaso **Buscar formulario apropiados**. Será el encargado de buscar aquellos formularios que puedan contener fallos para la búsqueda de sus respectivas vulnerabilidades.

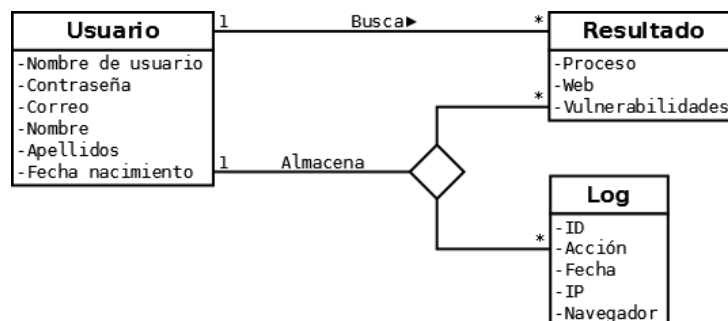
4.5. Modelo de dominio

Un modelo del dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema. Muchos de los objetos del dominio pueden obtenerse de una especificación de requisitos o mediante la entrevista con los expertos del dominio.

El objetivo del modelo de dominio es comprender y describir las clases más importantes dentro del contexto del sistema.

4.5.1. *Front-end*

En la Figura 12 se puede ver el diagrama de modelo de dominio de la página web.

Figura 12: Modelo de dominio del *front-end*.

Se ha pretendido realizar el modelo más sencillo posible, de ahí que hayan salido, únicamente, tres tablas:

- **Usuario:** objeto que guardará los datos de los usuarios que se den de alta desde el *front-end*.
- **Resultado:** objeto que guardará los resultados obtenidos cuando un usuario quiere buscar vulnerabilidades de una página web que haya introducido.
- **Log:** objeto que guardará los datos necesarios para tener un registro preciso de lo que hacen todos los usuarios en su uso del *front-end*.

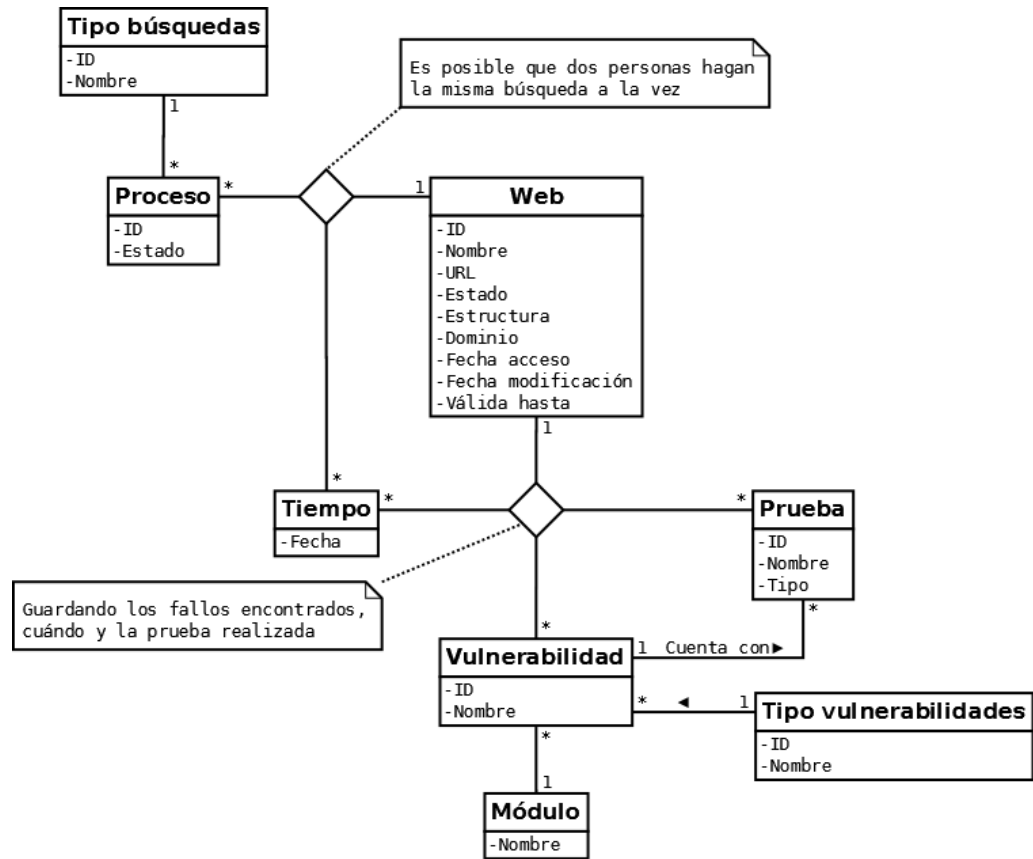
Además se puede ver que:

- Un usuario podrá realizar las búsquedas que quiera, pero cada búsqueda sólo pertenece a un único usuario.
- Un usuario podrá realizar las búsquedas que quiera y el registro de cada búsqueda quedará guardada en el *log*. También es posible que el usuario no haga una búsqueda, pero la acción que haya realizado también se guarda en dicho registro.

4.5.2. *Back-end*

Aunque en la planificación se interpretó que sería necesario realizar un modelo de dominio para el núcleo y un modelo de dominio para cada uno de los módulos, al final, solamente ha hecho falta realizar un único diagrama (Figura 13) que vale tanto para el núcleo como para cualquier módulo que se quiera insertar. Por lo tanto, se ahorra el tiempo de realización de este diagrama en cada uno de los módulos.

- **Proceso:** entidad que guardará el proceso que identifica una búsqueda.
- **Web:** entidad que guardará los datos de la página web que haya introducido el usuario.
- **Tipo búsquedas:** entidad que guardará el tipo de búsquedas que se pueden llegar a realizar.
- **Tiempo:** entidad que guardará el tiempo.

Figura 13: Modelo de dominio del *back-end*.

- **Prueba:** entidad que guardará la prueba realizada para buscar una vulnerabilidad.
- **Vulnerabilidad:** entidad que guardará la vulnerabilidad que se quiere buscar.
- **Tipo vulnerabilidad:** entidad que guardará el tipo de vulnerabilidades que se pueden buscar.
- **Módulo:** módulo que realiza la búsqueda de un tipo de vulnerabilidad.

5. Análisis y diseño

La siguiente fase a describir en un proyecto de desarrollo software es la fase de análisis y diseño. En esta fase se van a desarrollar las pautas de funcionamiento del sistema, tomando como pilar fundamental su arquitectura, definida en el planteamiento inicial, con el fin de definir un sistema lo suficientemente preciso como para permitir su construcción física.

Se analizan y diseñan todos los aspectos imprescindibles para el desarrollo de la aplicación, especificando el diseño de la base de datos y las estructuras del *back-end* y *front-end*.

Para una mejor comprensión de las funcionalidades, se adjuntan en el Anexo II los diagramas de secuencia más importante relacionados con el diseño del proyecto.

5.1. Diagrama de base de datos

El proyecto contará con dos bases de datos, en la API y en el servidor de búsqueda de vulnerabilidades.

Se ha decidido la realización de dos bases de datos con el objetivo de liberar de tareas innecesarias a cada una de las partes implicadas en el proyecto.

La primera de estas bases de datos se alojará en la API y contendrá los datos necesarios para que el *front-end* pueda operar casi autónomamente del servidor de búsqueda. Por lo tanto, la segunda base de datos recaerá en el servidor de búsqueda, que contendrá aquellos datos que sean necesarios para la realización de las búsquedas propias de las vulnerabilidades y que no sean importantes o necesarios para los usuarios finales.

Por último, se debe mencionar que sendos diagramas se obtendrán a partir de los modelos de dominio que se han realizado en la etapa anterior.

5.1.1. API

El diagrama de base de datos de la API se encuentra en la Figura 14, en la que se puede ver que contiene tres tablas que se expondrán a continuación.

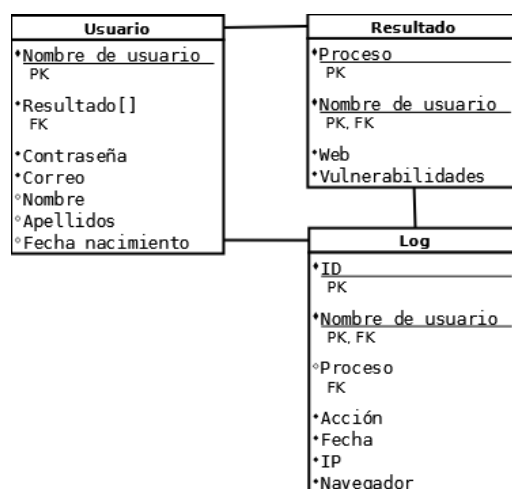


Figura 14: Diagrama de base de datos que tendrá la API.

- **Usuario:** tabla que almacenará los datos necesarios para la identificación de un usuario único. Además, también se almacenará en ella los índices de los resultados de las búsquedas realizadas por el usuario.
- **Resultado:** tabla que almacenará los datos necesarios para registrar vulnerabilidades encontradas en cada web de un proceso.
- **Log:** tabla que almacenará los datos necesarios para detallar las acciones realizadas por cada usuario en el *front-end*.

Aunque la fase de elección de herramientas se hará más adelante, se sabe que se desarrollará una base de datos de tipo NoSQL en la API. Entre las virtudes de este tipo de bases de datos, se encuentra la posibilidad de almacenar una lista de elementos dentro de otro elemento. Es por ello que, en el diagrama de la Figura 14, la tabla **Usuario** almacene una lista de identificadores de la tabla **Resultado**. Aunque a la larga el almacenamiento doble puede propiciar una falta de espacio, se pretende tener más rapidez a la hora de acceder a los datos.

- En la tabla **Usuario** se almacenan los resultados obtenidos de sus búsquedas con el objetivo de acceder a ellas de manera sencilla sin tener que recorrer la tabla **Resultado** al completo para la filtrando los resultados por el usuario en cuestión.
- La tabla **Resultado** se usará cuando se deseen saber datos de los resultados de manera general o para la realización de estadísticas futuras.

5.1.2. Servidor de búsqueda

El diagrama de base de datos del servidor de búsqueda se encuentra en la Figura 15, en la que se puede ver que contiene cinco tablas que se expondrán a continuación.

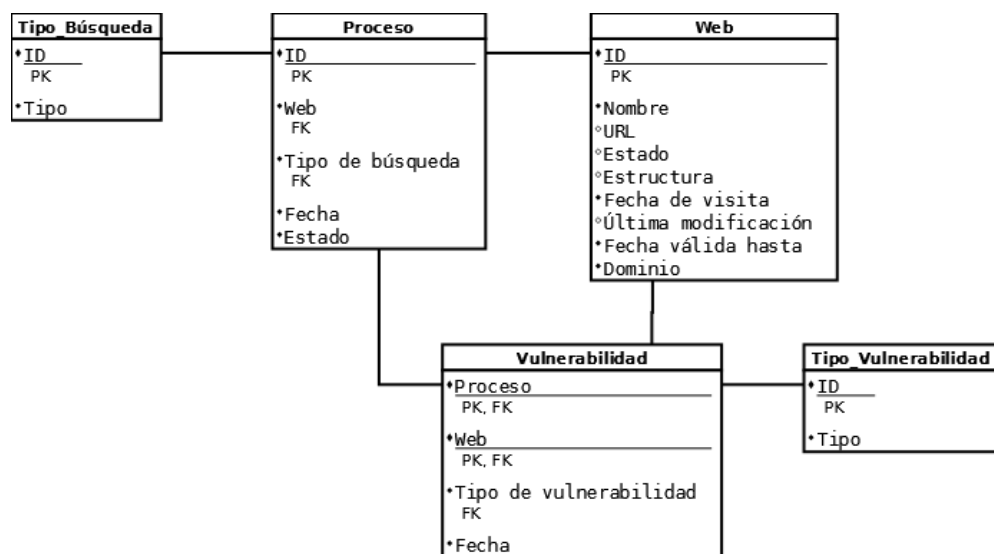


Figura 15: Diagrama de base de datos del servidor de búsqueda.

- **Proceso:** tabla que contendrá los datos que identifiquen a cada una de las búsquedas realizadas, además de saber el estado de las mismas.
- **Web:** tabla que contendrá los datos relacionados con la información de cada web de la que se vayan a buscar vulnerabilidades.
- **Vulnerabilidad:** tabla que contendrá los datos que identifiquen a cada vulnerabilidad encontrada. Es importante recalcar que una web puede tener más de una vulnerabilidad y ser descubierta en un mismo proceso.
- **Tipo_Búsqueda:** tabla que guardará los diferentes tipos de búsquedas que se pueden realizar.
- **Tipo_Vulnerabilidad:** tabla que contendrá los diferentes tipos de vulnerabilidades que el servidor puede encontrar.

5.2. Diagrama de clases del servidor de búsqueda

El siguiente paso es la realización del diagrama de clases. En este momento, no se realizarán diagramas de clases de la API y del *front-end* puesto que no se sabe todavía las herramientas que se van a usar y, dependiendo de la herramienta, es posible que se tenga que realizar de una manera u otra; son herramientas tan actuales que tienen características propias y son diferentes entre ellas. Durante su desarrollo se expondrán todos estos puntos que no se están comentado ahora.

Por lo tanto, sí que se puede realizar el diagrama de clases del servidor de búsqueda, puesto que no importa el lenguaje a usar para realizarlo. Este diagrama se puede observar en la Figura 16.

Se ha pretendido realizar un diagrama lo más simple posible. Las clases que no contienen métodos ni atributos dependerán de herramientas, librerías o técnicas que actualmente se desconocen y que se explicarán en la sección de desarrollo.

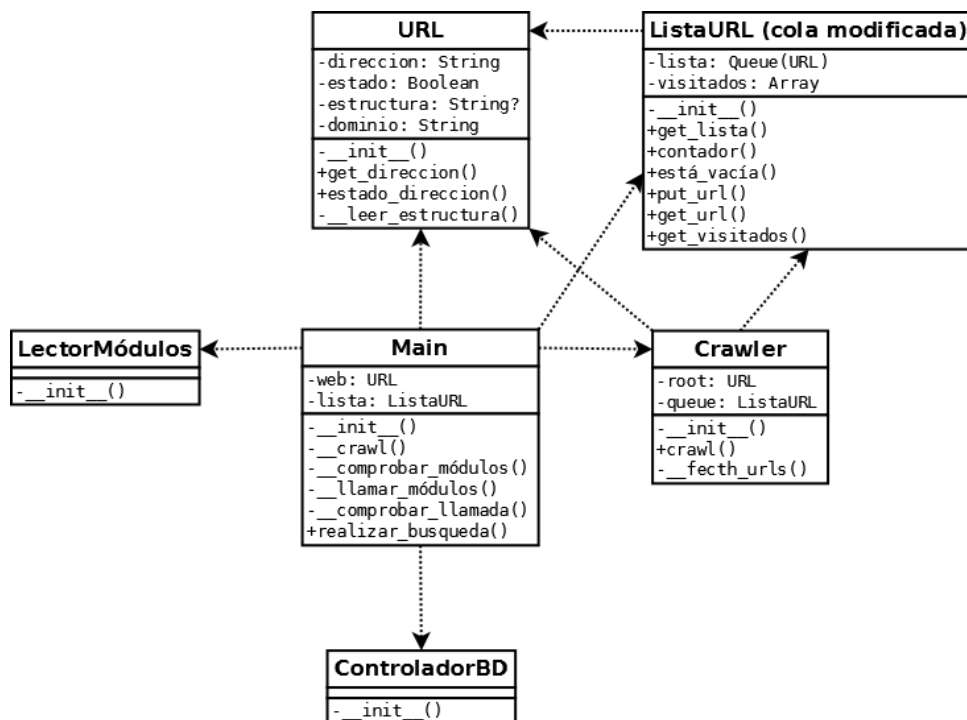


Figura 16: Diagrama de clases del servidor de búsqueda.

A continuación, se explicará cada una de las clases:

- **Main:** se trata de la clase principal de la aplicación, encargada de tomar la entrada y leerla, comprobar los módulos instalados y llamarlos para realizar la búsqueda solicitada por un usuario.
- **URL:** clase que guarda el contenido y otros datos relaciones con la URL suministrada o aquellas que se irán leyendo durante la ejecución.
- **ListaURL:** clase que contendrá una cola de URL lista para ser leída por los diferentes módulos de búsquedas. Se pretende realizar una “cola modificada” no porque se vaya a modificar el código fuente de la misma, sino porque se pretende mejorar su funcionamiento. Esta mejora se explica en los puntos siguientes de manera simplificada. Durante la explicación del desarrollo del servidor, se expondrá más detalladamente lo que se ha de realizar.
 - La lista de visitados contendrá todas las URL que han pasado por la lista.
 - La cola, como cualquier otra cola, se irá vaciando conforme se lean sus elementos. Por lo tanto, se irá vaciando conforme el primer módulo la vaya leyendo.
 - Una vez se vacíe la cola, lo que se quiere es volver a rellenarla con todos los datos que tenía en principio para que esté lista para usar por el siguiente módulo.
 - El motivo para realizar el punto anterior es, básicamente, para tener una única cola durante la ejecución de una búsqueda y no tantas colas como módulos.
- **ControladorBD:** clase que se conectará con la base de datos y que contendrá los métodos relacionados para realizar llamadas a la misma.
- **Crawler:** clase que leerá el árbol de direcciones de la página web introducida por el usuario y que creará la ListaURL que usarán el resto de módulos.
- **LectorMódulos:** clase que será capaz de leer los módulos de vulnerabilidades que están instalados y disponibles para su uso cuando se realicen peticiones.

5.3. Esquema REST de la API

Aunque no podamos todavía realizar un diagrama de clases de la API, sí que es posible realizar el esquema REST de la API.

Pero primero, es necesario explicar qué es REST. REST, que viene de *REpresentational State Transfer*, es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. Permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP, por lo que es simple y convencional.

Se puede considerar REST como un *framework* para construir aplicaciones web respetando HTTP, por lo que es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

Se ha pensado en el desarrollo de una API *Rest* porque permite la existencia de múltiples opciones de programas cliente, lo que es una ventaja importante. Tampoco es necesario la existencia de un programa, cualquier llamada a una función de la API con los parámetros adecuados será respondida con los datos requeridos en formato JSON.

El esquema a realizar, contiene las llamadas que se pueden realizar a la API, junto con los parámetros necesarios y el método HTTP a utilizar. Además, se muestran los datos que devuelve la API.

Como nunca se debe guardar estado en el servidor, toda la información que se requiere para mostrar la información que se solicita debe estar en la consulta por parte del cliente, es por ello por lo que se debe especificar el usuario y la contraseña al servidor en cada llamada.

El esquema REST del proyecto es el que se puede ver en la Figura 17.

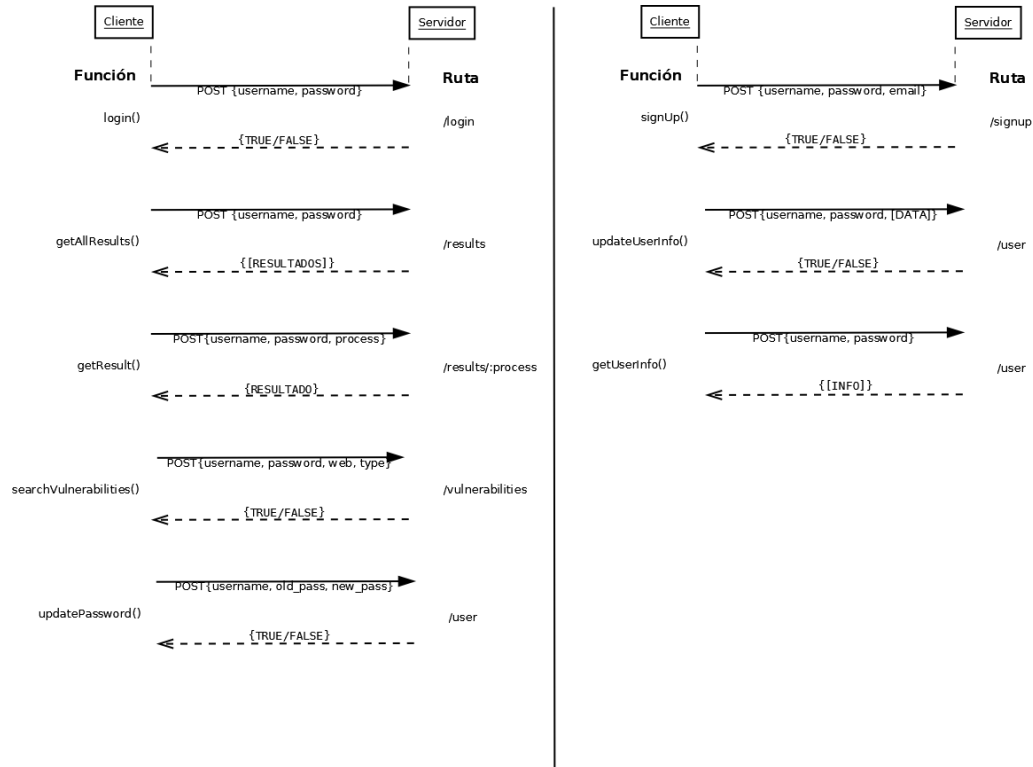


Figura 17: Esquema REST de la API.

6. Elección de lenguajes y tecnologías

Es importante la elección de lenguajes de programación y diferentes tecnologías a usar durante el desarrollo. Una mala elección hará que el desarrollo no sea sencillo y que existan problemas que no puedan solventar por no estar capacitadas para ello.

Se trata de un buen punto para esta elección, puesto que ya se conocen todos los detalles necesarios para empezar con el desarrollo del proyecto.

Para estudiar cada elección, se harán uso de tablas de “pros y contras” para estudiar las diferentes alternativas. Con estas tablas, y contando con la experiencia que se tiene hasta ahora, se esperan sacar las conclusiones que permitan seleccionar la mejor elección entre todas las alternativas que se encuentren.

6.1. Python

La elección de Python como lenguaje para la realización del núcleo de la aplicación encargada de la búsqueda de las vulnerabilidades no ha estado en duda en ningún momento, puesto que se trata de un lenguaje en el que es muy amigable trabajar con contenido web, además de ser un lenguaje muy rápido.

Sin embargo, el problema radica en que, actualmente, existen dos versiones de Python que se pueden usar para el desarrollo de aplicaciones, la 2.x y la 3.x.

Se recomienda el uso de Python 3.x (sección “*Which version should I use?*” [4]), siempre y cuando, no haga falta el uso de alguna dependencia que sólo esté disponible para versiones 2.x.

Por lo tanto, la elección radica entre si se prefiere usar un lenguaje más nuevo y mejor preparado frente a otro que usa más gente y del que existe más ayuda y soporte.

Finalmente, se implementará el núcleo en Python 3.x por ser más actual. Se espera que el aprendizaje del mismo haga que las dudas sobre su uso sean las mínimas posibles.

A continuación, se hablará sobre las diferencias entre las maneras de programar en Python con las que se tiene al programar en Java y sobre la elección del IDE que se empleará para realizar la programación en este lenguaje.

6.1.1. Diferencias entre Python y Java

Se tratará de explicar algunas diferencias entre estos dos lenguajes tras la lectura de una infografía (realizada por Perception System, leída en [5]) que resume de manera muy amigable, pero un tanto confusa, estos puntos.

Una de las diferencias más grandes entre Python y Java es la forma en la que cada lenguaje maneja las variables. Java tiene un tipado estático, lo que obliga a indicar el tipo cuando se declara una variable por primera vez y no permitirá cambiarlo. Por el contrario, Python usa el tipado dinámico, no teniendo que indicar el tipo al declarar las variables y permitiendo el cambio de tipo durante la ejecución del programa.

Otra diferencia es la manera de separar el código por bloques. Java, como muchos otros lenguajes, usa las llaves para definir el principio y el final de cada función o definición de clase; Python, en cambio, usa la indentación para separar el código en bloques.

La gran ventaja de Java es que puede ser usado para crear aplicaciones independientes de la plataforma, puesto que un sistema que tenga instalada una máquina virtual de Java podrá ejecutar casi cualquier aplicación Java. Sin embargo, es necesario compilar la aplicación en Python para cada diferente sistema si se desea realizar una aplicación multiplataforma.

Aunque, generalmente, las aplicaciones en Java se ejecutan de manera más rápida que las de Python, el tiempo de desarrollo de un mismo programa en los dos lenguajes es mucho más corto¹⁵ en Python que en Java.

6.1.2. Elección del IDE

Elegir un buen IDE es clave para la correcta realización del proyecto. Este IDE debe contener varios aspectos mínimos, que son:

- Que esté disponible para plataformas GNU/Linux.
- Que esté pensado para trabajar en Python.
- Que se pueda cambiar el tema de colores que resalten el código de ser necesario.
- Compatible con *git* de serie o, en su defecto, mediante un plugin de fácil instalación.

¹⁵Más información en: <https://www.python.org/doc/essays/comparisons/>.

- Que resalte el código de páginas web. Una única herramienta para poder programar el servidor, la API y la página web sería interesante.
- Que la instalación de librerías de Python sea sencillo o que una librería instalada manualmente se pueda usar sin tener que configurar nada.

Viendo los puntos y buscando en Internet, se llega a dos opciones: Eclipse o PyCharm. Es por ello que es necesario comparar estas dos alternativas y, la primera manera, será crear una tabla de pros y contras¹⁶ de cada una de ellas.

Pros	Contras
Usado durante toda la carrera	Necesidad de instalar el <i>plugin</i> Py-Dev
Soporte para multitud lenguajes de programación	Importar librerías con PyDev no es intuitivo
Integración con <i>git</i>	Poca documentación en Python
Código libre y gratuito	Los ajustes del IDE no son intuitivos

Tabla 7: Pros y contras de Eclipse.

Pros	Contras
Buena documentación	Interfaz desfasada
Integración con <i>git</i>	Rendimiento
Importar librerías con un simple clic	
Versión gratuita para la comunidad	
Soporte para lenguajes de programación que se llevan bien con Python	
Pensado para su uso con Python	
Completado de código	

Tabla 8: Pros y contras de PyCharm.

Tras estudiar los pros y contras que se pueden ver en las Tablas 7 y 8, la elección parece sencilla.

¹⁶Eligiendo el mejor IDE para Python. <http://pedrokroger.net/choosing-best-python-ide/>

Aunque Eclipse sea el IDE que más se haya usado durante el transcurso de la carrera, la instalación del *plugin* PyDev parece un simple apaño del IDE para que funcione junto con Python, no pudiendo aprovechar toda su potencia como lo hace con Java.

Por lo tanto, y sabiendo que el conocimiento de Python por parte del desarrollador es el justo en estos momentos, es preferible la elección de un IDE que tenga una buena documentación sobre Python, que haga más sencillo programar con este lenguaje y que sea pensado para el uso del mismo. Es, por ello, que el IDE que se usará sea PyCharm.

6.2. Base de datos del servidor de búsqueda

Conociendo el tamaño de la base de datos, no parece necesario recurrir a soluciones de bases de datos de empresas, como Oracle. Por lo tanto, quedan dos soluciones por comparar: MySQL (o su *fork*, MariaDB) o PostgreSQL.

Las dos opciones son bastante similares¹⁷:

- Son de código abierto.
- MySQL Workbench y PostgreSQL pgAdmin III son herramientas muy potentes.
- Tienen compañías detrás en caso de necesitar soporte profesional.
- Son plataformas ya maduras que, además, son usadas por grandes compañías: Google Cloud SQL, Pinterest, Facebook y Twitter usan MySQL; Amazon Redshift e Instagram, PostgreSQL.

Como se desconoce qué hará Oracle con MySQL, y viendo que parte de la comunidad parece haberlo dejado de lado al crear el *fork* MariaDB, se usará PostgreSQL por tener un futuro garantizado y por haber sido recomendado por compañeros que han trabajado con las dos opciones.

6.3. Blog y wiki

Se considera esencial que el blog y la wiki sean una única herramienta para facilitar el trabajo en ella.

¹⁷Pros y contras de PostgreSQL y MySQL. <https://www.quora.com/What-are-pros-and-cons-of-PostgreSQL-and-MySQL>

Se había pensando en realizar esta parte de manera manual y personalizada como el resto del proyecto; sin embargo, el propio reto de la realización de una herramienta de búsqueda de vulnerabilidades de por sí ya es lo suficientemente complicado como para añadir la realización de toda una wiki.

Las opciones son demasiadas, por lo que se ha escogido DokuWiki¹⁸, por las siguientes razones:

- Está realizada totalmente en PHP, lenguaje que se conoce muy bien.
- Es simple.
- No hace falta un gestor de base de datos, guarda los datos en ficheros.
- Tiene *plugins* disponibles para personalizar totalmente la herramienta.
- Ofrece la creación de un blog.

6.4. *Frameworks* para el *front-end*

En los últimos años, se ha visto cómo han ido apareciendo *frameworks* para ayudar en el desarrollo de aplicaciones web y ahorrar mucho tiempo y esfuerzo a los desarrolladores.

El principal problema está en que han aparecido muchos en poco tiempo y siempre están en constante evolución. Un día puede ser uno el mejor pero, al siguiente, quedarse anticuado y todo el mundo pasarse a la competencia.

Otro problema es el soporte que tienen las diferentes opciones. Al final, la gente acaba usando el que más soporte tenga y más mercado abarque frente a la competencia.

Se tiene pensado el uso de dos *frameworks* para el desarrollo del *front-end* de la aplicación: uno para el motor y otro para la interfaz.

6.4.1. Motor

Existen dos exponentes en lo que a uso de *frameworks* de JavaScript para el desarrollo MVC de *front-end*: AngularJS [6], de Google, y ReactJS [7], de Facebook.

¹⁸<https://www.dokuwiki.org/dokuwiki>

Antes de comenzar sobre la elección de uno de los dos *frameworks*, no está de más realizar tablas de pros y contras sobre cada uno de ellos. Estos datos se pueden observar en las Tablas 9 y 10.

Pros	Contras
Buena documentación	Problemas de rendimiento
Comunidad muy activa	Lógica entre HTML y JavaScript
Enlace de datos bidireccional	Versión 2 deja obsoleta a la 1
Patrón MVC	

Tabla 9: Pros y contras de AngularJS.

Pros	Contras
Aprendizaje sencillo	Problemas con librerías externas
Componentes aislados	Cambio de mentalidad de trabajo
<i>Re-renderizado</i>	Rendimiento con el <i>re-renderizado</i>
Patrón Flux [8]	Patrón Flux
	Curva de aprendizaje

Tabla 10: Pros y contras de ReactJS.

Viendo los resultados se pueden sacar varias conclusiones:

- AngularJS se adapta más a la manera de trabajar actual que se tiene, puesto que usa el patrón MVC, no teniendo que adaptarse al nuevo patrón propuesto por Facebook.
- ReactJS parece ser muy fácil de usar al principio, pero conforme se va introduciendo y buscando mayores funcionalidades a realizar, su uso se hace tedioso y no cuenta con la buena comunidad que hay detrás de AngularJS. Además, aunque se dice que el patrón Flux es muy sencillo, se desconoce su funcionamiento. Es, por lo tanto, la razón que hace que sea tanto una ventaja como un inconveniente.
- Los dos *frameworks* tienen problemas de rendimiento. Como la aplicación no es muy grande, no supone gran problema.
- El que AngularJS versión 2 deje obsoleta a la primera versión puede suponer un problema a medio/largo plazo.
- La documentación y comunidad de AngularJS es sobresaliente.

La comparativa está ajustada. Al final, todo depende del tiempo del que se dispone para la realización del proyecto. Aunque muchos desarrolladores invitan al uso de ReactJS, al final, la buena documentación y comunidad de AngularJS hacen inclinar la balanza. Además, ya se han realizado trabajos con AngularJS y se tiene más experiencia usando el patrón MVC (el patrón Flux se desconoce).

6.4.2. Interfaz

El único conocimiento que se tiene, actualmente, para el desarrollo de la interfaz de una página web es sobre Ionic [9]. Sin embargo, se hará uso de Bootstrap [10], de Twitter, puesto que Ionic sólo ofrece interfaces adaptadas a móviles, cosa que no se pretende en este proyecto por el momento.

Además, Bootstrap ofrece un soporte sobresaliente e, incluso, ejemplos para todo tipo de componentes. No está de más mencionar que existen usuarios que adaptan este *framework* para cambiarle el aspecto y dejarlo más personalizado. Es posible que se escoja alguna de estas modificaciones para realizar la interfaz.

6.5. API

Para el desarrollo de la API y, junto con las tecnologías del *front-end*, se hará uso de *MEAN stack*¹⁹ (acrónimo de MongoDB, Express, AngularJS y Node.js), puesto que acelera y simplifica el desarrollo de este tipo de aplicaciones web. Además, ya se tiene experiencia previo con el uso de estas tecnologías.

De las herramientas de las que se compone MEAN, conviene hablar de MongoDB²⁰, la base de datos NoSQL de la que se hará uso.

MongoDB, como cualquier otra base de datos NoSQL, en vez de guardar los datos en tablas como se hace en las base de datos relacionales, se guardan estructuras de datos en documentos tipo JSON con un esquema dinámico, que MongoDB llama formato BSON.

Una de las diferencias más importantes con respecto a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección –concepto similar a una tabla de una base de datos relacional–, pueden tener esquemas diferentes. Sin embargo, se hará uso de

¹⁹MEAN stack: [http://mean.io/#!/.](http://mean.io/#!/)

²⁰MongoDB: <https://www.mongodb.org/>.

una librería de Node.js, llamada Mongoose²¹, para la realización de esquemas para validar la validación de los datos.

Otra de las herramientas de este paquete es Express, que no es más que un *framework* para desarrollar aplicaciones web sobre Node.js.

Por último, AngularJS, en este caso, sólo se utilizará en la parte del *front-end*, puesto que una API no requiere de una interfaz gráfica para funcionar.

6.6. Resumen

Para realizar un pequeño resumen, se ha modificado la imagen de la arquitectura que se va a realizar (capítulo Planteamiento inicial, sección de Arquitectura, ver Figura 2) para añadir las tecnologías que se van a utilizar para el desarrollo. El resultado es el que se puede observar en la Figura 18.

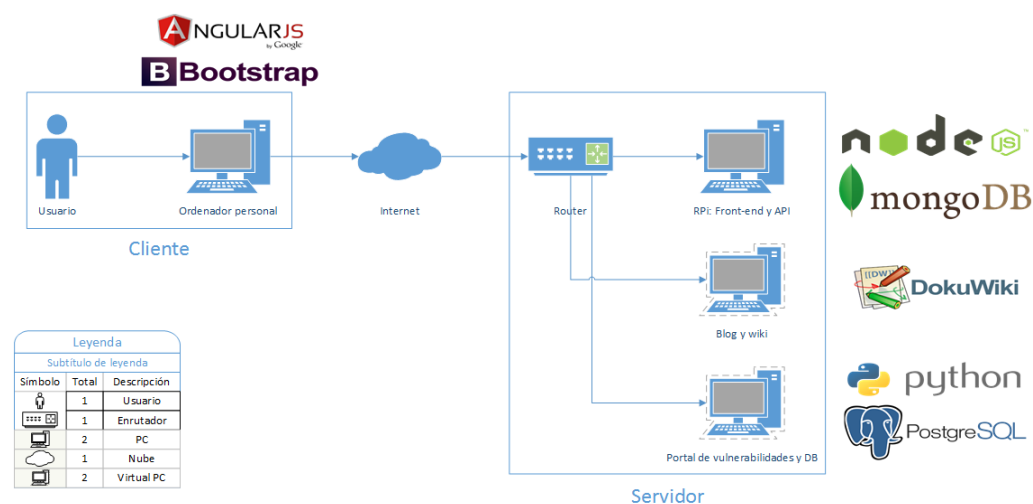


Figura 18: Tecnologías que se van a utilizar.

- Para el *front-end* y que será lo que cargue el cliente, se utilizarán los *frameworks* de JavaScript AngularJS y Bootstrap. Como bien se ha comentado en el capítulo de Análisis y diseño, sección esquema REST de la API, no es necesaria la realización de ningún *front-end*, puesto que la API responde a cualquier llamada a una función realizada correctamente. Sin embargo, conviene realizar una mínima interfaz para que no sea el usuario el encargado de realizar esas llamadas manualmente.

²¹Mongoose: <http://mongoosejs.com/>.

- Para la API, se hará uso de *MEAN stack*, cuyos paquetes principales son Node.js, con Express, como motor de la propia API y MongoDB como motor de la base de datos.
- Para el blog y wiki se hará uso de DokuWiki.
- Por último, para el motor de búsqueda se hará uso del lenguaje Python y PostgreSQL como motor para la base de datos general.

Además, se debe añadir que se hará uso del IDE PyCharm para el desarrollo en Python y de las tecnologías JavaScript de la API y del *front-end*.

7. Desarrollo

En este capítulo se cubrirán las diversas fases acometidas a lo largo del desarrollo del sistema.

7.1. *Back-end*

Se llamará *back-end* a la parte lógica del servidor que contiene el buscador de vulnerabilidades y su base de datos.

La Figura 19 muestra el diagrama de flujo sobre lo que se realiza en el *back-end* cuando éste recibe una nueva petición.

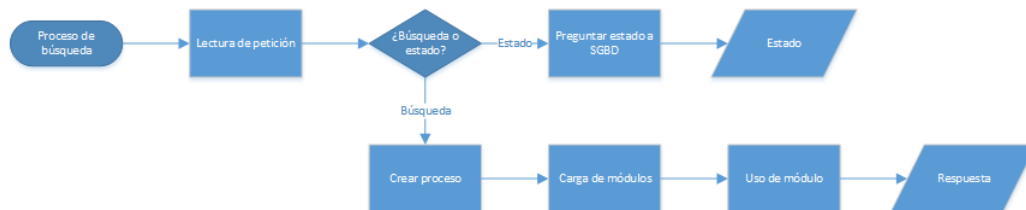


Figura 19: Diagrama de flujo del proceso.

- Se lee la petición. Existen dos diferentes acciones a realizar si la petición es correcta:
 1. La primera puede ser preguntar por el estado de un proceso que esté activo.
 2. La segunda puede ser una petición de búsqueda de vulnerabilidades con los módulos a usar.
 - a) Se crea el nuevo proceso junto con la página web especificada.
 - b) Se cargan los módulos solicitados y se comprueban que éstos estén activos.
 - c) Se les llama uno a uno para buscar diferentes tipos de vulnerabilidades.
 - d) Se guarda en la base de datos todos los fallos encontrados, dónde y de qué tipo.
 3. Se envía/n la/s respuesta/s a la API.
- De ser incorrecta la petición, no se realiza nada, se desestimará la petición.

7.1.1. Base de datos general

Se ha utilizado el diagrama de base de datos generado en la fase de análisis y diseño (sección Diagrama de bases de datos, ver Figura 15) para realizar la base de datos general. Además, como se ha comentado en la fase de elección de herramientas, se ha escogido el Sistema Gestor de Base de Datos (SGBD) PostgreSQL. Junto con el SGBD, se hará uso de la herramienta gráfica de mantenimiento pgAdmin para realizar todas las tareas necesarias sobre la base de datos, con el fin de no tener que realizar todas las acciones mediante comandos.

Aún sabiendo la existencia de herramientas que crean los *scripts* necesarios para crear la base de datos y sus tablas, se ha decidido realizar esta operación manualmente, con la finalidad de coger soltura con pgAdmin.

Lo primero que se debe hacer es crear un nuevo servidor que será el que aloje la base de datos general del proyecto. La Figura 20 muestra una captura del programa pidiendo los datos necesarios para crear el servidor. Estos datos serán los que luego se usen para realizar la conexión desde la aplicación en Python.

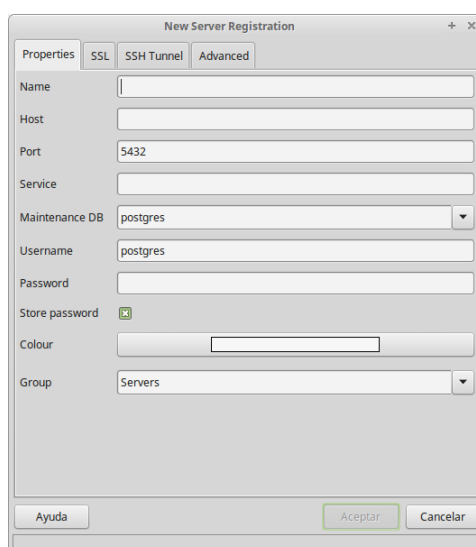


Figura 20: Datos para crear un nuevo servidor de base de datos.

Con el nuevo servidor, ya es posible la creación de la base de datos, usuarios, procedimientos, etc. Durante el desarrollo, se hará uso del usuario que crea PostgreSQL por defecto, pero se recomienda crear un nuevo usuario que tenga los permisos mínimos necesarios para el uso de la base de datos cuando se entre en producción.

7.1.1.1. Secuencias

Para realizar identificadores que se autoincrementen, es necesaria la creación de una secuencia por cada identificador que requiera de esta función. No existe la opción *autoincrement* como la que puede existir en otros SGBD como MySQL.

Los datos que se deben introducir, aparte del nombre de la secuencia y el dueño de la misma, son los que se muestran en la Figura 21.

Figura 21: Datos para crear una nueva secuencia.

Una vez creada la secuencia, se puede añadir a la tabla que se requiera que su columna identificador sea autoincrementente. Para ello, cuando se crea una tabla o se modifica, a la hora de añadir o modificar una nueva columna, se debe añadir un valor por defecto como se puede ver en la Figura 22, en el que se está indicando que el siguiente valor será el que diga la secuencia.

Figura 22: Añadiendo secuencia a una columna.

Como se ha comentado anteriormente, cada columna que se quiera autoincrementar debe tener asociada una secuencia diferente, puesto que es la

propia secuencia la que guarda el número que después se asociará a un nuevo registro.

7.1.1.2. Procedimientos almacenados

Con la finalidad de evitar un gran número de llamadas a la base de datos desde el servidor de búsquedas, se ha decidido realizar el mínimo número de procedimientos almacenados que realizarán todas las tareas necesarias en la base de datos que la herramienta necesitará para funcionar.

PostgreSQL hace uso del lenguaje PL/PgSQL, que es similar al lenguaje PL/SQL de Oracle, para el desarrollo de procedimientos almacenados.

Mediante la herramienta pgAdmin, los parámetros se indican mediante interfaz gráfica, debiendo programar, únicamente el cuerpo del procedimiento, es decir, las declaraciones y el cuerpo del procedimiento. El resto del código lo genera pgAdmin de manera automática con los valores que se le indiquen.

Cinco han sido los procedimientos almacenados que han desarrollado para que realicen todas las operaciones relacionadas con la base de datos que requiera la parte realizada en Python que se explicará en posteriores secciones.

1. *get_current_process_status(text)*. Devuelve el estado de la búsqueda activa, de haberla, del usuario que se le indique como parámetro.
2. *get_process_status(integer, text)*. Devuelve el estado de una búsqueda de la que se conoce su identificador y el usuario. Si el usuario no ha sido el creador de la búsqueda, se devuelve un error. Este procedimiento se encuentra inactivo porque sirve en situaciones en las que un usuario puede realizar más de una búsqueda.
3. *new_process(text, text, integer, integer)*. Crea un nuevo proceso de búsqueda, cuyos parámetros son el usuario, la web por la que se comienza la búsqueda, el tipo de búsqueda a realizar y un identificador de estado de la búsqueda, por si se desea que empiece desde otro punto. Se creará un nuevo proceso de búsqueda siempre y cuando el usuario no tenga otro proceso en marcha, en el que se guardará la página web en la base de datos de no existir previamente. Se devolverá el número de proceso o *null* de haber error.
4. *update_process(integer, integer)*. Actualiza el estado de la búsqueda para un proceso concreto.

5. *vulnerability_found(integer, integer, text)*. Guarda la vulnerabilidad encontrada de la página web indicada sobre un proceso de búsqueda activo.

Para devolver elementos de la base de datos, es necesaria la creación de cursores en los procedimientos almacenados, que serán los encargados de almacenar dichos elementos y que serán accesibles llamando al cursor desde la aplicación realizada en Python. Para evitar confusiones, los cursores se llaman igual que los procedimientos en los que se encuentran.

A continuación se muestra el código del procedimiento *get_process_status(integer, text)* escrito usando pgAdmin.

```
1 DECLARE
2     s refcursor := 'get_process_status';
3 BEGIN
4     OPEN s FOR SELECT status FROM process_list
5             WHERE process = p AND puser = u;
6     RETURN s;
7 END;
```

1. Se indica que existen variables a declarar.
2. Se declara un nuevo cursor como referencia, indicando su nombre.
3. Empieza el cuerpo del procedimiento.
4. Se abre el cursor en el que se introducirá la respuesta de la sentencia *SELECT*.
5. *p* y *u* son los nombres de los parámetros requeridos de entrada.
6. Se devuelve el cursor.
7. Se indica que se ha terminado el procedimiento.

7.1.2. Servidor de búsquedas

En esta sección se irán explicando todas las partes que componen el servidor de búsquedas, desde su núcleo y características propias, hasta los diferentes módulos que buscan las vulnerabilidades.

7.1.2.1. Conexión con el SGBD

Se utilizará el adaptador llamado `psycopg2`²² para realizar las conexiones desde el servidor en Python hacia la base de datos en PostgreSQL. Se deberá instalar desde los repositorios del Sistema Operativo.

Se tenía la idea de la realización de una clase Singleton que realizase todas las acciones necesarias contra la base de datos. Sin embargo, se rechaza esta idea de realizar una clase Singleton porque su realización en Python es complicada, muchas de las maneras para llegar a realizar este tipo de clases no gustan a la comunidad de Python, y mucha parte de esta comunidad no ve necesario el uso de clases Singleton en Python. Por ello, será necesario cerrar la conexión con la base de datos tras cada petición que se realice a la misma para evitar cualquier tipo de error.

Esta clase estará compuesta por todos los métodos necesarios para realizar las llamadas a cada uno de los procedimientos almacenados creados en la base de datos, mandando los parámetros necesarios y formateando la respuesta recibida para que sea legible por el resto de la aplicación.

Además, aprovechando que cada vez que una instancia es creada por Python, se llama automáticamente a su método llamado `__init__`, se establecerá la conexión con la base de datos en esta llamada, puesto que así sabemos que la conexión ya estará establecida cuando queramos realizar una petición y, de fallar, dará error y la aplicación no será capaz de continuar.

Por último, es importante entender cómo se realizan llamadas a los procedimientos almacenados de PostgreSQL usando esta librería desde Python. Para ello, se mostrará el método que realiza la llamada a la base de datos para crear un nuevo proceso de búsqueda.

```
1 def new_process(self, url, user, s_type, status):
2     cur = self.conn.cursor()
3     cur.callproc("new_process", [url, user, s_type, status])
4     aux = self.conn.cursor('new_process')
5     process = aux.fetchone()
6     cur.close()
7     self.conn.commit()
8     return process[0]
```

²²Más información en: <http://initd.org/psycopg/>.

PostgreSQL funciona mediante cursores, que serán los encargados de realizar las llamadas una vez esté la conexión con la base de datos realizada y de recoger los valores resultantes de los procedimientos almacenados de requerirlo.

1. Declaración del método con su paso de parámetros. El parámetro *self* se debe poner siempre que el método pertenezca a una clase con herencia de *object*.
2. Se establece un nuevo cursor a través de la conexión.
3. Se realiza la llamada al procedimiento almacenado usando el método *callproc* que maneja el cursor, pasándole como parámetros el nombre del procedimiento y, en una lista, los parámetros del mismo.
4. Si el procedimiento devuelve alguna respuesta, se llamará a dicha respuesta usando el cursor.
5. Como pueden ser varias líneas (se trata de una base de datos), se deberían recorrer e ir guardándolas una a una. En este caso, sólo se devuelve el identificador del proceso comenzado de haber ido correctamente.
6. Se cierra el cursor.
7. Se realiza el *commit*.
8. Se devuelve, si existe, la respuesta.

7.1.2.2. Lectura de peticiones

La lectura de las peticiones recibidas desde la API se gestionará usando JSON Schema²³. Esta librería se encarga de leer las peticiones en JSON recogidas y validarlas. De ser alguna petición correcta, se seguirá adelante con la ejecución; en caso contrario, lanzará una excepción y se enviará un error.

Estas peticiones se validan usando un esquema que esta petición debe seguir para ser correcta.

Se mostrará un ejemplo, usando el esquema de validación para la petición de búsqueda de vulnerabilidades usada en el servidor.

²³Más información en: <http://json-schema.org/>.

```
1 search_schema = {  
2     "title": "JSON from API",  
3     "type": "object",  
4     "properties": {  
5         "user": {"type": "string"},  
6         "url": {"type": "string"},  
7         "search_options": {  
8             "type": "array",  
9             "items": {  
10                "number": {"type": "number"},  
11                "module": {"type": "string"}  
12            },  
13            "required": ["number", "module"]  
14        },  
15    },  
16    "required": ["user", "url", "search_options"]  
17 }
```

Se puede observar que tenemos un título, que es opcional, el tipo del que debería el JSON de la petición, en este caso, un objeto, y las propiedades de dicho objeto.

Dentro de las propiedades, se ha de enviar el usuario y la URL como cadenas de caracteres; las opciones de búsqueda será una lista cuyos elementos será el número y nombre del módulo. Los apartados *required* muestran los elementos que son obligatorios.

El código siguiente muestra el JSON que se enviará desde la API para realizar una búsqueda.

```
1 var msg = {
2     "user": b.USERNAME,
3     "url": b.URL,
4     "search_options": [
5         {
6             "number": 1,
7             "module": "crawler"
8         },
9         {
10            "number": 2,
11            "module": "sqlinjection"
12        },
13        {
14            "number": 3,
15            "module": "csrf"
16        }
17    ]
18 }
```

7.1.2.2.1. Instalación y uso

La instalación de esta librería la realiza el propio IDE que se ha decidido instalar, PyCharm, cuando se escribe la línea en la que se importa la librería para poder usarla.

Dos son las diferentes peticiones que se pueden recibir en el servidor: la petición de búsqueda de vulnerabilidades y la petición de saber existe una búsqueda activa para ese usuario. El funcionamiento de este método se explicará a continuación.

La petición recibida se leerá la primera vez intentando validarla con el esquema de búsqueda de vulnerabilidades. De ser correcta, se recogerán los datos de la petición y se seguirá la ejecución buscando vulnerabilidades. En cambio, de no ser válida, se aprovechará el lanzamiento de la excepción para intentar validarla con el segundo esquema.

Como la primera vez, de ser esa validación correcta, la ejecución derivará en la llamada a la base de datos para conocer si existe una búsqueda activa para ese usuario. Por lo tanto, de ser errónea de nuevo, en la segunda excepción lanzada, se devolverá el control al núcleo diciendo que la petición recibida ha sido errónea y que no se seguirá con la ejecución.

```
1 def __check_call(self, call):
2     try:
3         validate(call, search_schema)
4     except:
5         try:
6             validate(call, current_status_schema)
7         except:
8             return 0
9         else:
10            self.user = call["user"]
11            return 2
12    else:
13        self.user = call["user"]
14        self.url = call["url"]
15        self.actions = call["search_options"]
16    return 1
```

7.1.2.3. Gestor de módulos

Como bien se ha ido explicando durante toda esta memoria, el servidor estará compuesto por un núcleo y por una serie de módulos que serán los que hagan las funciones de búsqueda de vulnerabilidades o funciones de añadido que no se pensaron en fases anteriores y que pueden ser útiles en un futuro.

Para que estos módulos puedan ser leídos por el núcleo, será necesario pensar en una estructura de ficheros estándar para todos los módulos. Cada módulo estará contenido en un directorio propio que irá dentro de otro directorio llamado “módulos” para tenerlos todos en un mismo lugar.

El módulo estará compuesto, mínimo, por un fichero en Python llamado “module.py” que contendrá un método llamado “main”, que será el método al que llame el núcleo.

Dentro del núcleo, será necesario comprobar que los módulos requeridos en una nueva búsqueda están instalados o no.

Para ello, se ha realizado un método que recorre el directorio de los módulos. Antes de recorrer este directorio, habrá sido necesario leer la petición realizada por el usuario para saber qué módulos hacen falta.

Antes de continuar con la explicación, el código del método realizado se muestra a continuación.

```
1 def __check_modules(self):
2     # Getting the root folder of the project.
3     path = os.path.dirname(os.getcwd())
4     # Adding the modules folder to the path.
5     path = os.path.join(path, "modules")
6     # Looping looking for the installed modules.
7     for name in os.listdir(path):
8         folder = os.path.join(path, name)
9         if os.path.exists(os.path.join(folder, "module.py")):
10             self.modules[name] = True
11         else: # Module required is not installed.
12             self.modules[name] = False
```

Se puede observar como, una vez dentro del directorio de los módulos, se recorre la lista de módulos hasta que coincida con el primero que nos hace falta. Hará falta una lista en la que se guarde si el módulo necesario está operativo o no. Por lo tanto, aunque un método no esté disponible, la ejecución se realizará en aquellos en los que sí lo estén, por lo que es importante que un método no dependa del resto y sea autosuficientes.

7.1.2.4. Cola personalizada

La estructura de datos que mejor se adapta para su uso en este proyecto es la cola. Las razones para su uso son las siguientes:

- Ir colocando todas las páginas a buscar una de detrás de la otra.
- Una vez vista una página ya no se tendrá que volver a ver.

Sin embargo, se ha de realizar una clase con una cola, pero se han de introducir otras estructuras y ciertos métodos para que se adapten perfectamente al proyecto.

El primer problema radica en las páginas ya visitadas. De tener un único módulo que buscara vulnerabilidades no se tendría problemas, pero no es el caso. Como bien se ha expuesto antes, una página visitada ya no se vuelve a visitar, lo que haría que otro módulo no pudiera leerla.

Lo que se ha realizado es añadir una lista en la que se irán guardando todas las páginas web que ha llegado a tener la cola para, una vez terminada,

volver a llenarla con los mismos elementos y así esté lista para su siguiente uso.

Copiar una cola no es una solución, puesto que Python no realiza una copia de una variable, sino que le asigna el mismo puntero.

El segundo problema se trata de la no repetición de páginas que serán visitadas. Para la búsqueda en colas, se tiene que recorrer la cola entera para comprobar que un elemento a introducir no esté ya en la misma. Aprovechando la lista creada para el anterior problema y sabiendo que Python ordena los elementos, se realizará la búsqueda en la lista en vez de en la propia cola.

El código siguiente muestra el método que saca el primer elemento de la cola. Se debe comprobar la excepción, puesto que, cuando se trata de retirar un elemento y la cola está vacía, salta la excepción de “cola vacía”.

```
1 def get_url(self):
2     try:
3         aux = self.url_list.get(block=False)
4         return aux
5     except queue.Empty:
6         # Filling the queue again for next use.
7         for u in self.aux:
8             self.url_list.put(URL(u))
9     return None
```

De haber elemento, se retornará sin problema; de no haberlo, a la hora de tratar la excepción, se aprovechará el momento para rellenar de nuevo la cola, aprovechando la lista con todas las direcciones. En este segundo caso, se devolverá *none* para decir que no hay más elementos.

7.1.2.5. Lectura de páginas web

Python es capaz de realizar llamadas a páginas web y conseguir su contenido, pero para buscar etiquetas dentro de ese contenido, se debe recorrer al completo. Por este motivo, se hará uso de la librería llamada BeautifulSoup²⁴, puesto que permite la fácil lectura de las etiquetas HTML y poder recorrerlas como queramos, incluso pudiendo leer el resto de etiquetas de las se compone.

²⁴Más información en: <http://www.crummy.com/software/BeautifulSoup/>.

Esta librería se instala usando el gestor de paquetes del sistema operativo e importándola como “bs4” en el código en Python.

7.1.2.6. Módulo de *webcrawling*

Durante el análisis y diseño, esta parte se pensaba hacer mediante una clase del propio núcleo del servidor. Sin embargo, puesto que es posible que no se use siempre dependiendo del usuario que esté usando la aplicación, es más cómodo convertir esta clase en un módulo aparte que será leído solamente cuando haga falta.

Una vez se tiene la manera de realizar llamadas a las páginas web, este módulo lo que realiza es una lectura de todas las etiquetas `<a>` en busca de hipervínculos en su atributo `href`.

Antes de realizar la llamada, el módulo guardará la página web en la que se va a buscar el árbol web con el objetivo de guardar, solamente, aquellas páginas pertenecientes a la misma aplicación web.

Por lo tanto, se leerá la página web introducida por el usuario y, usando la librería Beautiful Soup, se irán recorriendo todas las etiquetas `<a>`, almacenando aquellas que sean del mismo dominio y que no se hayan encontrado antes.

Una vez leída la página, se procederá a la siguiente, la primera encontrada y se volverá a realizar el bucle.

Al final, lo que se conseguirá será la cola de páginas web en las que se podrá buscar vulnerabilidades.

7.1.2.7. Módulo de inyección SQL

Se ha realizado un diagrama de flujo, el mostrado por la Figura 23, para explicar de manera visual los pasos que se realizan por parte del módulo para buscar entre los diferentes tipos de inyecciones SQL que se van a tratar de realizar.

Se dependen de dos factores para poder realizar estos ataques:

- Para la realización de las inyecciones SQL basadas en error y las ciegas se depende de que la URL de la página web contenga parámetros.
- Para tratar de autenticarse en el sistema del usuario, se necesita que la página web en la que se tratará de realizar el ataque contenga un formulario de conexión al sistema.

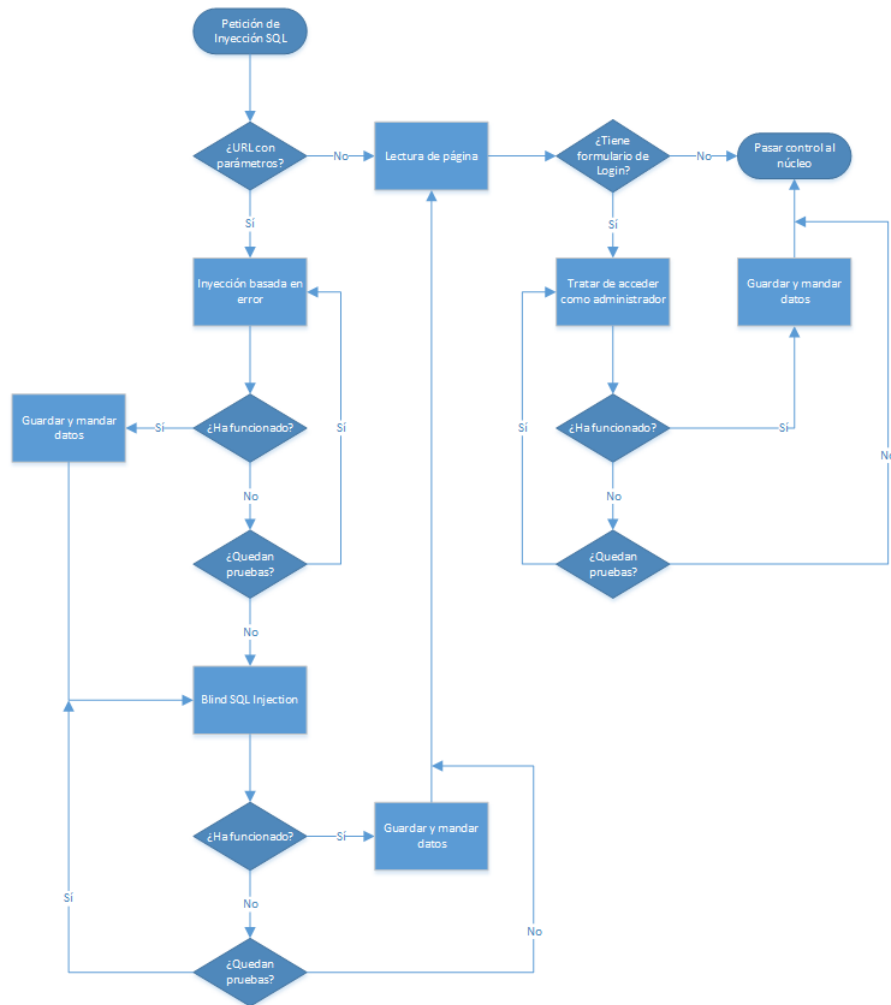


Figura 23: Diagrama de flujo del proceso de Inyección SQL.

Por lo tanto, sabiendo las dependencias anteriores, se puede pasar a comentar el funcionamiento del módulo.

1. Si la URL contiene parámetros:

- a) Se tratará de realizar ataques basados en errores. Lo que se pretende es que, introduciendo el parámetro adecuado, el SGBD del sistema muestre un error determinado. De mostrarse, la página web sería vulnerable a este tipo de ataques.
- b) El siguiente punto será realizar ataques ciegos. El funcionamiento es similar a los ataques basados en errores. Se introducirán parámetros en los que se especificarán tareas a realizar, normalmente

calcular decimales de π , por el SGBD durante el tiempo que se le diga. De quedarse la página web cargando durante el tiempo especificado por la tarea, se puede decir que la página sería vulnerable a este tipo de ataques.

2. De no tener parámetros o tras haber realizado los ataques anteriores, se buscaría si la página contiene un formulario de conexión. Saber si existe este formulario es sencillo, se buscan dentro de todos los formularios, aquél que contenga un campo texto y un campo de contraseña solamente (sin importar los campos ocultos). De existir, se tratará de autenticar en el sistema. Si tras realizar la conexión, la nueva página web no contiene el formulario de conexión, se puede decir que la página es vulnerable.
3. Se irán realizando pruebas en cada tipo de inyección hasta que no haya más o se encuentre una que logre realizar el ataque. No es necesario recorrer todas las pruebas de haber funcionado ya una de ellas.
4. Los datos se irán guardando en la base de datos general y enviando a la API conforme se vayan encontrando.

Por evitar causar daños, se ha decidido no activar la parte de búsqueda de inyecciones ciegas porque requiere tener a la máquina que alberga a la web del usuario trabajando durante un periodo determinado por nuestra herramienta para tratar de saber si es vulnerable o no. No se quiere hacer ningún daño y es posible que la web del usuario esté en plataformas en las que se cobra según potencia consumida (p. ej. Amazon AWS).

7.1.2.8. Módulo de CSRF

En páginas web realizadas con lenguajes instaurados (p. ej. PHP y ASP), muchos de los problemas derivados por ataques CSRF se pueden solucionar añadiendo un *token* diferente como campo oculto a cualquier formulario que la página web contenga. Estos elementos se generan en el servidor en el momento de servir formularios al cliente.

Cada *token* no es más que una cadena aleatoria generada mediante funciones *hash* y que se guardan en el servidor con el objetivo de comprobar que la información enviada por un usuario sea efectivamente enviada por ese usuario. Estos elementos deberían tener un tiempo de límite válidos, es decir, que caduquen si no se realiza comprobación.

Al servir el *token* junto con el formulario a un usuario, al enviar la información rellena, se adjunta esta cadena que se comprueba con la guardada en el servidor. Si coinciden, se puede decir que el usuario es el que ha enviado dicha información.

Una vez explicado el funcionamiento del uso de *tokens*, se puede pasar a comentar el funcionamiento del módulo. Se trata de un módulo bastante sencillo. Los pasos que se realizan son los siguientes:

1. Se recorre la lista con las URL a buscar.
2. Por cada URL, se leerá el código en busca de formularios.
3. De existir algún formulario, se recorren sus campos en busca de algún campo oculto (un *input* de tipo *hidden*).
4. De no haber alguno que contenga algún *token*, se parará la búsqueda en esa URL y se considerará el error, avisando de la existencia del mismo.
5. Volvemos al paso 2.

Es posible que, aún no teniendo este *token*, la aplicación web no sea vulnerable. Existen otros métodos para evitar este fallo, pero la introducción de este *token* es la más sencilla.

Al ser un proyecto enfocado a desarrolladores noveles en este campo, se ha pretendido adecuar la herramienta a ellos. En un futuro, de continuar su desarrollo, se pueden añadir otras búsquedas sobre esta vulnerabilidad.

7.1.2.9. Servidor de escucha

Una de las partes que no se previeron durante las fases de análisis y diseño fue la realización de una clase que sirviera para establecer un servidor de escucha entre la API y la propia aplicación que genera las búsquedas de vulnerabilidades.

Se decide la realización de un servidor que reciba *sockets* cuyo contenido sean los objetos JSON que la aplicación es capaz de entender.

Python está preparado para realizar este tipo de tareas, por lo que no es necesario la instalación de ninguna librería.

Lo que se realiza es una escucha infinita de *sockets*. Para ello, se realiza un servicio que será la parte a la que se llame cuando un nuevo *socket* es abierto por la API.

Además, para poder tener más de un *socket* abierto a la vez, es necesario que se creen hilos o hijos que sean los que vayan leyendo cada uno de ellos.

El código que realiza las funciones básicas de servidor, se encuentra tras esta línea.

```
1 class Service(socketserver.BaseRequestHandler):
2     def handle(self):
3         # Data not shown.
4
5         # Getting the data sent via socket.
6         data = self.request.recv(1024).decode('UTF-8')
7         # The data received is like 11#{'foo':"bar"}.
8         # Getting the part after the '#' character.
9         # It'll be in the tail.
10        head, sep, tail = data.partition('#')
11        # That data is a JSON and have to be loaded.
12        data = json.loads(tail)
13
14        core = Core() # New Core instance.
15        response = core.start(data) # Starting the Core.
16
17        # More data not shown.
18
19 class ThreadedTCPServer(socketserver.ForkingMixIn,
20                          socketserver.TCPServer):
21     pass
22
23 t = ThreadedTCPServer(('localhost', 9999), Service)
24 t.serve_forever()
```

Aunque pueda parecer extraño, la clase *ThreadedTCPServer* (sección 21.21.1. *Server Creation Notes*, [11]) que es la que realiza el servidor, no contiene ningún método, es la forma oficial de realizar esta tarea.

7.1.2.10. Llamadas entre *back-end* y API

Cada módulo será el encargado de realizar las llamadas a la API con el fallo encontrado, encapsulando sus datos en un objeto JSON.

La llamada se realizará como si de un navegador se tratase, por lo que se enviará el objeto JSON mediante una llamada HTTP POST como se puede ver en el extracto de código siguiente.

```
1 data = {
2     "PROCESS": self.process,
3     "WEB": w,
4     "VULNERABILITY": v,
5     "USER": self.user
6 }
7 requests.post(api, json=data)
```

Es importante seguir el mismo patrón de objetos JSON para todas las llamadas realizadas desde el servidor para que sea validadas correctamente desde la API.

7.2. API

La API será la parte encargada de gestionar las peticiones de los usuarios, guardar sus datos, los datos de todas sus búsquedas y será la parte encargada de realizar las llamadas al *back-end*.

Como bien se ha comentado en el capítulo anterior sobre la elección de tecnologías, la API tendrá un motor montado sobre Node.js, lo que quiere decir que se tendrá que programar en lenguaje JavaScript.

7.2.1. Estructura de directorios

No existe una estructura concreta para la realización de una aplicación haciendo uso del motor de Node.js, pero sí que se recomienda seguir una estructura MVC (modelo vista controlador) con estos directorios (sección *Example*, [12]):

- ***controllers***/: se añadirán los controladores (la lógica) de la aplicación.
- ***helpers***/: contendrá aquél código y funcionalidad que se comparten con diferentes partes del proyecto.
- ***models***/: representará los datos y manejará su almacenamiento.

- **public/**: contendrá aquellos ficheros estáticos.
- **views/**: proveerá los diseños.
- **tests/**: contendrá los tests.
- **app.js**: fichero de inicialización de la aplicación.
- **package.json**: fichero en el que se escribirán los paquetes de los que dependerá la aplicación.

Como se trata de una API, es decir, no contiene ninguna vista, los directorios que harán falta serán la de controladores y modelos, además de los ficheros *app.js* y *package.json*. El directorio de pruebas no se realizará, puesto que la realización de las pruebas requiere de una librería externa y se desconoce su funcionamiento. En el apartado de las pruebas se explicará lo que se ha realizado para *testear* esta parte del código.

7.2.2. Base de datos no relacional

Se hará uso de la librería Mongoose para realizar las tareas pertenecientes a la base de datos no relacional MongoDB, además de servirnos también para la realización de los esquemas. Un “esquema” se podría considerar semejante a lo que se llama “tabla” en lenguaje de base de datos relacional. Por lo tanto, cada uno de estos “esquemas” irá guardado en el directorio de modelos.

Se mostrará un ejemplo antes de continuar la explicación.

```
1 exports = module.exports = function(app,mongoose) {
2     var Schema = mongoose.Schema;
3     var userSchema = new Schema({
4         USERNAME: {type:String, required:true},
5         PASSWORD: {type:String, required:true},
6         EMAIL: {type:String, required:true},
7         NAME: {type:String},
8         LASTNAME: {type:String},
9         TYPE: {type:Number, max:3, default:1},
10        RESULTS: [{type:Schema.Types.ObjectId,
11            ref:'Result', required:true}]
12    }); mongoose.model('User',userSchema); };
```

Como se puede observar, cada esquema está compuesto por campos, como si una base de datos relacional se tratara, en el que se debe especificar, mínimo, el tipo del campo. Otros datos que se pueden indicar son si el campo es requerido, valores por defecto, mínimos o máximos de ser campos numéricos, etc.

7.2.3. Enrutamiento de las llamadas

Cada petición recibida debe redirigirse a su correspondiente controlador para que se pueda encargar de ella, por lo que se debe realizar un enrutamiento de las llamadas. Para ello, se debe seguir el esquema REST, visto en el capítulo de análisis y diseño, para saber las llamadas que tendremos procedentes desde el *front-end*. Como se trata de una API pequeña, se aprovechará el fichero principal (*app.js*) para indicar el enrutamiento de las llamadas. De haber sido un proyecto más complejo, se tendría que realizar un nuevo directorio para indicar estas rutas.

A continuación, se podrá observar el código realizado para tratar algunas de las llamadas.

```
1 // Router options
2 var router = express.Router();
3 app.use(router);
4
5 router.route('/signin').post(userController.signIn);
6 router.route('/signup').post(userController.signUp);
7 router.route('/getprofile').post(userController.getUserInfo);
8
9 router.route('/results').post(resultController.getAllResults);
10 router.route('/search').post(resultController.search);
11
12 router.route('/saveresult').post(coreController.saveResult);
13
14 // More routes not shown.
```

Para poder cargar el enrutamiento, se debe llamar al método *Router* de la librería Express. Express es parte del *MEAN stack*, y no es más que una librería pensada para ayudar a los desarrolladores a lidiar con las entradas y salidas del sistema, en este caso, una API.

Después, se le debe indicar al motor de Node.js, mediante la línea número 2 (*app.use(router);*), que se quiere hacer uso de ese método al que se ha llamado en la línea anterior.

Por último, el resto de líneas, se realizan los enrutamientos. Se han separado por controladores, siendo el primero grupo las llamadas relacionadas con los usuarios, el segundo aquellas sobre los resultados y, el tercero, sobre aquellas peticiones recibidas por el *back-end*. Lo que se debe hacer es indicar que la llamada recibida a una función concreta por parte del *front-end*, en la que se ha enviado los datos mediante POST, se le será redirigido hacia una función concreta dentro de su respectivo controlador.

7.2.4. Poblar datos y agrupación por proceso

Como ya se ha comentado, MongoDB permite que se puedan guardar listas de elementos dentro de un campo de una base de datos. Por lo tanto, a la hora de llamar a dicho campo, MongoDB contestará mediante el envío de los índices.

Sin embargo, MongoDB también ofrece la manera de poder obtener los campos relacionados con cada uno de los índices. Se entenderá mejor mediante un ejemplo.

Cada usuario tiene almacenado los índices de cada una de las vulnerabilidades que el sistema ha encontrado en cada una de sus búsquedas. A la hora de acceder a dichos resultados, MongoDB permite rellenar con el resto de campos que componen los resultados (proceso, web, vulnerabilidad, etc) a cada uno de ellos, sin tener que recorrer la “tabla” de resultados en busca de cada uno de los campos.

En el código siguiente se buscan un usuario en concreto y se pueblan sus resultados de vulnerabilidades.

```
1 User.findOne({USERNAME: b.USERNAME}).populate('RESULTS')
2 .exec(function(error, user) { /* Data not shown.*/ }
```

El segundo problema trata sobre la agrupación por proceso. Como bien se ha explicado, cada proceso de búsqueda puede tener muchos resultados (un resultado por cada vulnerabilidad encontrada). MongoDB no permite agrupar por proceso, por lo que, la búsqueda realizada en el código anterior, nos saldrían todos los resultados encontrados para ese usuario.

Lo que se quiere, al ser todos los datos objetos JSON, es agrupar por cada proceso, los resultados del mismo. Para ello, se encontró una librería para Node.js llamada *Underscore*²⁵.

```
1 var r = _.chain(user.RESULTS).sortBy("DATE", "WEB")
2 .groupBy("PROCESS", "WEB").value();
```

Se indica el objeto a usar (los resultados), se ordenan por fecha y por web, se agrupan mediante proceso y por web (una web puede tener más de una vulnerabilidad) y, por último, se piden los valores, es decir, el resultado. Este resultado será otro objeto JSON con la agrupación realizada.

7.2.5. Llamadas entre API y *back-end*

Uno de los aspectos que se debían realizar por completo eran las llamadas desde la API hacia el *back-end*. Los únicos puntos conocidos dentro de este apartado eran que se debían enviar mensajes codificados en JSON y que estos mensajes debían enviarse mediante *sockets* usando TCP.

Afortunadamente, Node.js está preparado para el envío de serie de mensajes vía *socket* teniendo que especificar, únicamente, el puerto e IP del servidor y si se realiza mediante TCP o UDP. Sin embargo, existe una problemática para el envío de objetos JSON.

La comunidad ya conocía este problema y ciertos desarrolladores ya se pusieron manos a la obra para ofrecer una librería que pudiera realizar esta acción.

Una rápida búsqueda en Internet hace llegar hasta el perfil de GitHub de Sebastian Seilund²⁶, un desarrollador danés que ha realizado un paquete²⁷ para Node.js, apoyándose en la propia librería de *sockets* de Node.js, que permite enviar objetos JSON mediante estos canales.

El extracto de código siguiente muestra una de las llamadas que se realizan desde la API hacia el servidor, abriendo una conexión mediante un *socket* TCP contra el mismo.

²⁵Acerca de *Underscore*: <http://underscorejs.org/>.

²⁶Perfil de GitHub: <https://github.com/sebastianseilund>.

²⁷Más en: <https://github.com/sebastianseilund/node-json-socket>.

```
1 // Opening a socket to communicate with the Python server.
2 var net = require('net');
3 // Decorate a standard net.Socket with JsonSocket.
4 var socket = new JsonSocket(new net.Socket());
5 socket.connect(9999, '127.0.0.1');
6 // Don't send until we're connected.
7 socket.on('connect', function() {
8     socket.sendMessage(msg);
9     socket.on('data', function(data) {
10         response.status(200).send(data);
11     });
12 });
```

7.2.6. Gestión de librerías

Node.js hace uso del gestor *npm* para la instalación de las librerías externas que se necesiten para realizar el proyecto. Como bien se ha comentado en la introducción de la sección, esta gestión de librerías se llevará a cabo en el fichero *package.json*, que no es más que un fichero en formato JSON en el que se indican las diferentes librerías que requiere la aplicación y la versión mínima de cada una de ellas.

7.3. Interfaz gráfica del *front-end*

A la hora de diseñar la interfaz gráfica del *front-end* se han tenido en cuenta los diez heurísticos de Jakob Nielsen [13], con el reto de evitar problemas de usabilidad.

- **Visibilidad del estado del sistema.** El sistema siempre debería mantener informados a los usuarios de lo que está ocurriendo.
- **Relación entre el sistema y el mundo real.** El sistema debería hablar el lenguaje de los usuarios mediante palabras, frases y conceptos que sean familiares al usuario.
- **Control y libertad del usuario.** Es posible que los usuarios realicen una función del sistema por error y necesiten una “salida de emergencia”, sin tener que pasar por una serie de pasos. Se deben soportar las funciones de deshacer y rehacer.

- **Consistencia y estándares.** Los usuarios no deberían preguntarse si acciones, situaciones o palabras diferentes significan en realidad la misma cosa; siga las convenciones establecidas.
- **Prevención de errores.** Mejor que un buen diseño de mensajes de error es realizar un diseño que prevenga la aparición de problemas.
- **Reconocimiento antes que recuerdo.** Se deben hacer visibles los objetos, acciones y opciones. El usuario no tendría que recordar la información que se le da en una parte del proceso para seguir adelante. Las instrucciones para el uso del sistema deben estar a la vista o ser fácilmente recuperables cuando sea necesario.
- **Flexibilidad y eficiencia de uso.** La presencia de aceleradores, que no son vistos por los usuarios novatos, puede ofrecer una interacción más rápida a los usuarios expertos. Se debe permitir que los usuarios adapten el sistema para usos frecuentes.
- **Estética y diseño minimalista.** Los diálogos no deben contener información irrelevante.
- **Ayudar a los usuarios a reconocer, diagnosticar y recuperarse de errores.** Los mensajes de error se deben escribir en un lenguaje claro y simple, indicando de forma precisa el problema y sugerir una solución constructiva al mismo.
- **Ayuda y documentación.** Incluso en los casos en el que el sistema pueda ser usado sin documentación, podría ser necesario ofrecer ayuda y documentación. Dicha información debería ser fácil de buscar, estar enfocada en las tareas del usuario, con una lista concreta de pasos a desarrollar y no ser demasiado extensa.

Teniendo en mente estos diez heurísticos, se han implementado diversos prototipos con el fin de evolucionar la interfaz de manera adecuada.

7.3.1. Prototipo a papel

En el primer paso se ha realizado un prototipo a papel, puesto que es la manera más sencilla y rápida de realizar un diseño y de realizar cambios si se encuentran fallos.

En la Figura 24 se puede ver un formulario de conexión a la web una vez intentado buscar una vulnerabilidad sin haber estado conectado antes. La Figura 26 sería la pantalla equivalente realizada a ordenador.

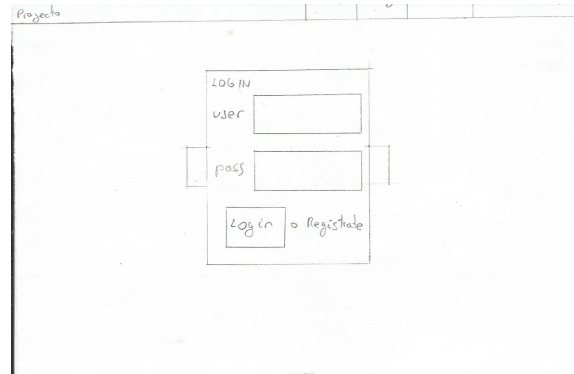


Figura 24: Interfaz de la pantalla principal.

A su vez, la Figura 25 muestra una pantalla en la que es posible tanto conectarse como registrarse.

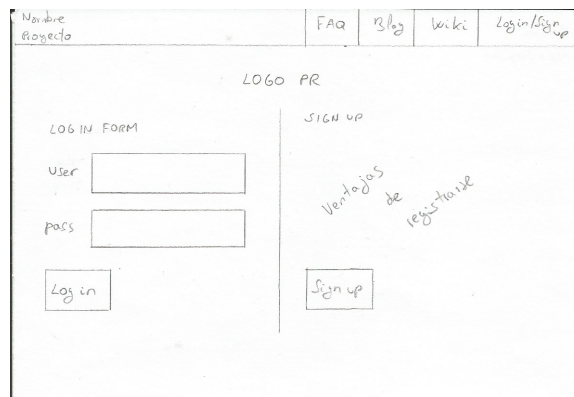


Figura 25: Interfaz del *login*/registro.

El equivalente a ordenador sería la Figura 27 en la que se ha cambiado la parte del registro, poniendo ahí también su formulario.

7.3.2. Prototipo a ordenador

La segunda iteración se trata de pasar este prototipo realizado en papel a un prototipo a ordenador.

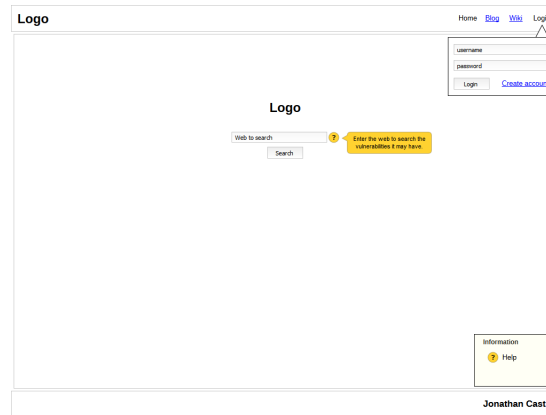
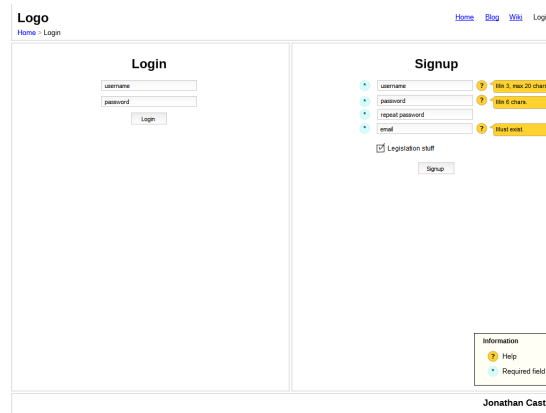


Figura 26: Interfaz de la pantalla principal.

La tarea siguiente será la de presentar este prototipo realizar a ordenador a usuarios potenciales de la aplicación que se va a realizar. Con esta tarea, se pretende buscar fallos de diseño y estructurales, además de que los usuarios pueden dar opiniones y pueden ofrecer mejoras e ideas que pueden no haber sido previstas.

Figura 27: Interfaz del *login*/registro.

Una vez realizado encuestas a estos potenciales usuarios, se tienen que estudiar los cambios sugeridos por éstos y si las ideas dadas merecen la pena ser agregadas. Con ello, se pasaría a realizar un tercer prototipo, segundo a ordenador.

Los cambios propuestos sobre la interfaz de aquellas cosas que no han gustado, o son mejorables, han sido las siguientes:

- No mostrar todas las búsquedas realizadas en el menú izquierdo del usuario, sino la de poner una tabla paginada al entrar en la sección principal.
- Quitar el número de proceso en las búsquedas realizadas, por no ser un dato que los usuarios deban conocer.
- En el botón de buscar, en el formulario de búsqueda de vulnerabilidades, poner la palabra “*hack*” por estar de moda.
- Eliminar las “migas de pan”.
- A la hora de actualizar la contraseña, poner primero la caja para insertar la contraseña antigua.

Sin embargo, no todos los cambios sugeridos entraban dentro del marco de trabajo o estaban dentro de los objetivos del mismo. Todos los encuestados recibieron información acerca del proyecto, los objetivos principales y se les explicó el marco de trabajo a seguir. Por lo tanto, aunque toda sugerencia es bienvenida, se tendrán que rechazar los siguientes puntos:

- Que no sea obligatorio el registro.
- La posibilidad de compartir resultados o búsquedas con otros usuarios registrados. Se les explicó que este proyecto no se trata de una red social.

También comentaron sugerencias e ideas constructivas que podrían ser interesantes añadir durante el desarrollo o tras finalizar los objetivos:

- Dar la oportunidad de ver pruebas realizadas previamente por parte del administrador para que un usuario no registrado vea un ejemplo y que haga que éste se registre si le ha gustado cómo funciona la página web.
- Poder relanzar búsquedas ya realizadas.
- Que el sistema envíe correos electrónicos.

- Tal y como se muestra la hora de comienzo de la búsqueda, poner lo que ha tardado también dicha búsqueda.
- Dar la opción de poder poner un nombre a la búsqueda.

A continuación, Figura 28, muestra la pantalla del prototipo con las vulnerabilidades encontradas en una página web.

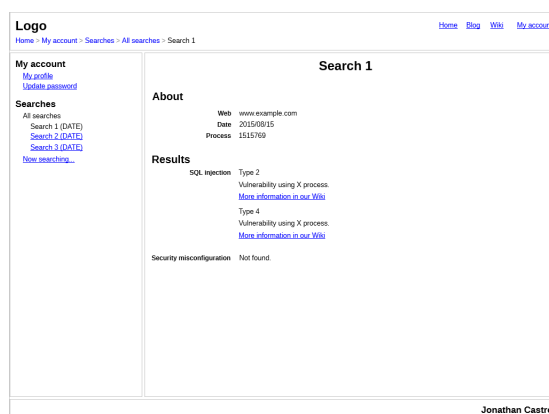


Figura 28: Interfaz de una búsqueda.

7.3.3. Primera evolución de la Interfaz gráfica

Una vez empezada la programación de la interfaz usando Bootstrap, el ahorro de tiempo es considerable comparándolo con el uso de una herramienta de prototipado, por lo que, a partir de ahora, se ajustará la interfaz directamente desde el propio código por ser más sencillo.

La primera tarea ha sido traspasar lo ya realizado en el prototipo antes de considerar los puntos obtenidos con las encuestas realizadas con los clientes potenciales.

Lo primero que se puede observar es el nombre en clave del proyecto. Proviene de un *Pokémon* llamado Vulpix. Se escogió por empezar por 'vul', al igual que la palabra vulnerabilidad.

Además, en la Figura 29 se puede contemplar que se ha perdido el formulario de conexión flotante. La razón principal de su pérdida es por querer reducir elementos en pantalla y tener menos trabajo redundante para realizar una misma acción.

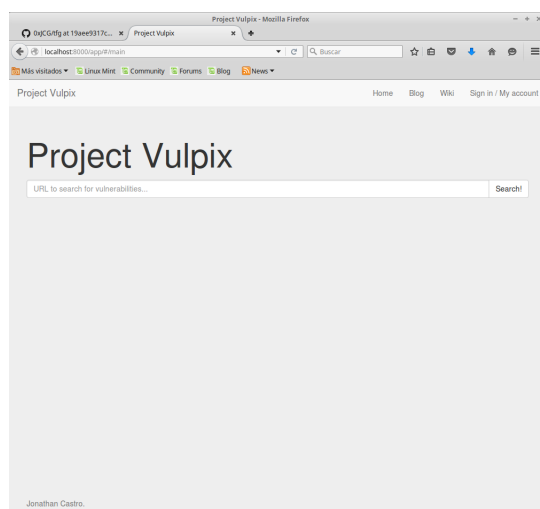


Figura 29: Primera evolución de la interfaz de la página principal.

A su vez, también se ha eliminado la leyenda de botones, por ser tratarse de un elemento innecesario viendo el *target* de usuarios al que está pensada esta página web.

Al final, se ha buscado mantener lo más simple posible la interfaz y eliminar todo aquello que se veía innecesario o que cargaba la interfaz.

La Figura 30, que se encuentra a continuación, muestra la evolución en la pantalla de conexión y registro.

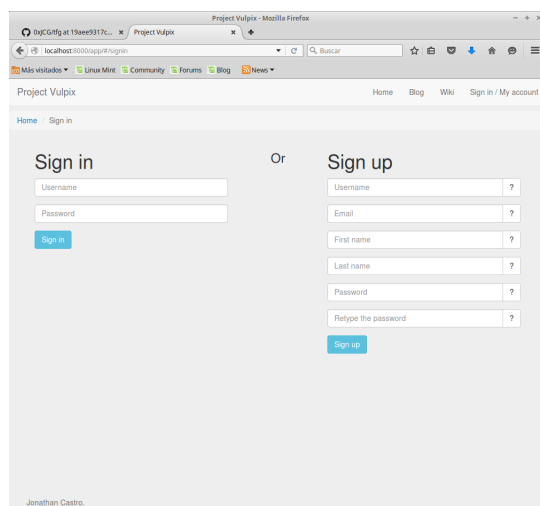


Figura 30: Primera evolución de la interfaz del *login*/registro.

A su vez, se volvió a pedir a los potenciales clientes que dieran su opinión con respecto a esta nueva interfaz, avisándoles que las opiniones previas del prototipo se tendrían en cuenta en la segunda evolución, junto con los cambios que sugieran en esta primera iteración. Algunos puntos fueron interesantes:

- Centrar el título del proyecto en la pantalla principal con respecto a la caja de búsqueda de vulnerabilidades.
- Centrar todo el buscador con respecto a la página.
- Añadir un logotipo personal. Por ahora, hasta encontrar un nombre, se quedará el nombre que se ve en las Figuras.
- Cambiar el tipo de letra en las cajas en la que escriben los usuarios.
- Cambiar el *favicon*. Sale el de Bootstrap. Se cambiará cuando tenga logotipo final.
- Ha gustado la velocidad de respuesta de la web.

7.3.4. Segunda evolución de la Interfaz gráfica

El principal problema de la primera evolución era que el tema elegido era el que Bootstrap pone por defecto. Por lo tanto, se ha buscado un tema más acorde para la página web y que fuese lo más amigable posible para cualquier tipo de usuarios, incluidos aquellos que puedan tener problemas de visión.

Además, se han tomado en cuenta algunos puntos que los potenciales clientes comentaron en sus encuestas, tanto en las iniciales con el prototipo, como en la primera evolución.

A continuación, se irán exponiendo todos los cambios realizados.

La Figura 31 muestra la segunda evolución de la pantalla principal. Se puede observar que el menú superior se ha oscurecido y que el título se ha centrado con respecto a la caja de búsqueda. Se ha tratado de centrar verticalmente el formulario de búsqueda pero no quedaba del todo bien.

En la Figura 32 se puede observar que se ha eliminado las “migas de pan”. En posteriores encuestas se verá si ese cambio ha sido correcto.

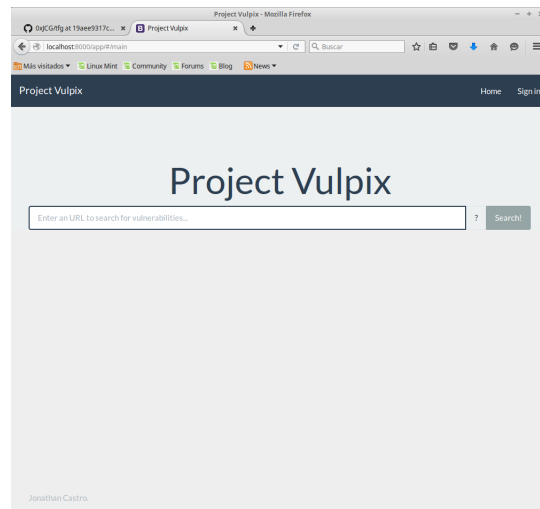


Figura 31: Segunda evolución de la interfaz de la página principal.

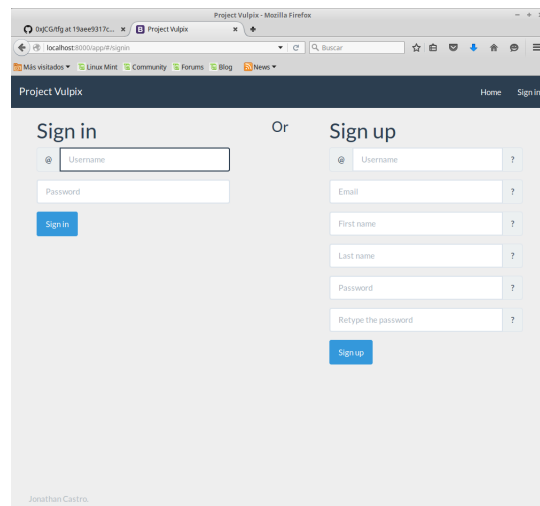


Figura 32: Segunda evolución de la interfaz del *login*/registro.

Otro cambio importante ha sido la eliminación completa del menú izquierdo dejando una barra de navegación en la parte superior que muestra las pestañas accesibles y marca la pestaña en la que está el usuario.

El resto de cambios realizados han sido:

- No se muestra el número de proceso de una búsqueda.
- A la hora de actualizar la contraseña, la caja de la contraseña actual se ha puesto la primera.

7.3.5. Interfaz gráfica definitiva

Los cambios generados desde la segunda evolución son escasos, pero son importantes:

- La eliminación de las “migas de pan” ha gustado en las encuestas.
- Se ha buscado un nombre para el proyecto propio, añadiendo un logotipo y cambiado el *favicon*.
- Se ha añadido el aviso de las *cookies*.
- Se han realizado los términos de uso y las políticas de privacidad y de *cookies*.
- Se ha añadido un vínculo hacia la página de conexión desde el aviso de la pantalla principal al intentar realizar una búsqueda sin estar conectado.
- Se ha añadido una imagen que gira en el aviso de una búsqueda en marcha para que el usuario vea que está en marcha.
- El usuario podrá eliminar su cuenta.
- Se muestra el mensaje tras el formulario de registro en el que, de registrarse, el usuario acepta los términos y condiciones.

Se mostrarán dos de las pantallas que más han cambiado. La primera es la página principal (Figura 33) y la segunda muestra el mensaje tras el formulario de registro (Figura 34) explicado en el último punto anterior.

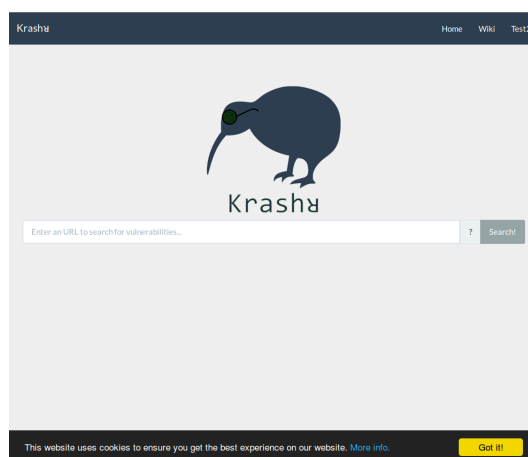
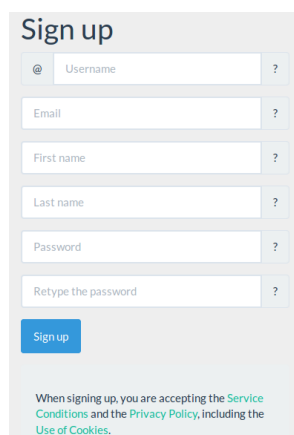


Figura 33: Página principal final.



The image shows a 'Sign up' form with the following fields: Username (with an '@' icon), Email, First name, Last name, Password, and Retype the password. Each field has a question mark icon to its right. Below the fields is a blue 'Sign up' button. At the bottom, there is a disclaimer: 'When signing up, you are accepting the [Service Conditions](#) and the [Privacy Policy](#), including the [Use of Cookies](#).'

Figura 34: Formulario de registro final.

7.4. *Front-end*

AngularJS ofrece un proyecto²⁸ para no tener que comenzar desde cero, por lo que se empezará el *front-end* desde ese proyecto.

7.4.1. Estructura de directorios

AngularJS, como se comentó en el capítulo de elección de tecnologías, este *framework* hace uso del patrón MVC (modelo vista controlador). Sin embargo, existe por la red discusiones acerca de la estructura correcta para guardar cada uno de los ficheros.

Existen dos aproximaciones, ordenamiento por tipo o por característica.

El ordenamiento por tipo consiste en guardar cada componente del mismo tipo en un mismo directorio, es decir, por poner un pequeño ejemplo, todos los controladores en un único directorio compuesto por todos los controladores. Por lo tanto, se trataría de la aproximación realizada en la API.

En cambio, la segunda aproximación sería ordenar todo por característica. Se pondrá otro ejemplo para tratar de explicar esta manera de realizar la estructuración. Se supondrá que se tiene un blog en el que se ve contenido, pero que existe un panel de administración. Esta aproximación cuenta que toda la parte de administración debería ir junta (tanto sus controladores, como modelos, etc) y la parte de contenido en otro directorio aparte.

²⁸Angular-seed: <https://github.com/angular/angular-seed>.

Ninguna de las de opciones es mejor que la otra. La aproximación por tipo es buena para pequeños proyectos por ser más clara a simple vista; sin embargo, una aproximación por característica sería adecuada para proyectos grandes con muchos ficheros, puesto que se tendrían ordenados por características y para realizar un cambio sólo se tendrían que realizar cambios en el directorio de la característica a modificar, sin tener que entrar en ningún otro directorio.

Por lo tanto, y puesto que la API tiene una estructura similar, se realizará la aproximación por tipo. Lo que sí que se hará será realizar una adaptación del mismo, para separar los ficheros por lenguajes, para tratar de mejorar la visibilidad de los mismos. La estructura final se puede ver en la Figura 35.

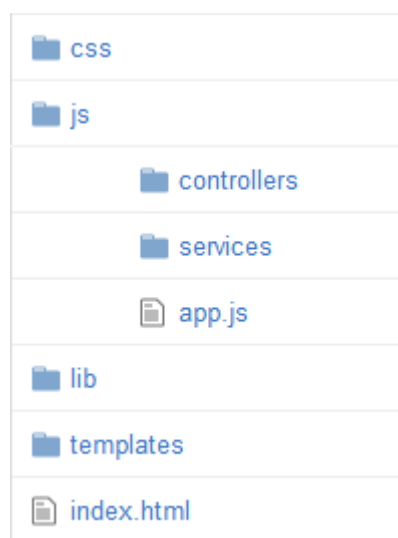


Figura 35: Estructura final de los ficheros del *front-end*.

- **css**: contiene todos los ficheros de estilos.
- **js**: contiene todos los ficheros JavaScript del *front-end* (controladores, servicios y el fichero de inicialización de Angular).
- **lib**: contiene todas las librerías externas utilizadas.
- **templates**: contiene todos los ficheros HTML que conforman todas las páginas de la web.
- **index.html**: fichero índice en el que se llama a todas las librerías y *frameworks* usados, además de ser el fichero en el que se irán *embebiendo* las páginas del directorio *templates* cuando haga falta.

7.4.2. Gestión de librerías

Al igual que la API, el *front-end* también hace uso de un fichero *package.json* para realizar la gestión de librerías.

Además, en esta ocasión se debe indicar qué pasos se realizan para iniciar la aplicación, puesto que no es una aplicación basada en Node.js, pero que sí hace uso de varios de sus componentes.

Por lo tanto, se tendrá que indicar la librería que levanta el propio *front-end*, el puerto, indicar si se deben realizar pruebas antes de inicializar, utilizar otro tipo de gestor diferente de *npm* para descargar librerías (es posible que *npm* no contenga todas las librerías necesarias y se tenga que hacer uso de Bower).

7.5. Instalación de módulos

Para la instalación de nuevos módulos, se requiere la modificación de dos ficheros, añadir nuevos datos a la base de datos general y colocar el directorio del módulo en el lugar donde le corresponde (directorio *modules* dentro del servidor de búsqueda).

El primer fichero a modificar es el fichero *core.py*, dentro del directorio *core* del servidor de búsqueda, concretamente, su método *start*. El extracto de código siguiente muestra dónde se debe incluir la llamada al nuevo módulo.

```
1 if action['number'] == 2:
2     from App.modules.sqlinjection.module import main
3     main(url_list, process, self.user)
4 elif action['number'] == 3:
5     from App.modules.csrf.module import main
6     main(url_list, process, self.user)
7 else:
8     continue
```

Se debería crear un nuevo elemento *elif* con el número que se le haya asignado al nuevo módulo (debe ser diferente de 1, 2 ó 3). Tras ello, se realiza la importación y se llamaría al método *main* del mismo.

En Python no existe la estructura *switch case*, su alternativa es el uso de diccionarios. Como se trata de una manera un tanto extraña (y nunca

vista), se decidió hacer uso de *if-else*. Con pocos módulos no hay problema; de aumentar el número de módulos, se debería cambiar a la realización mediante diccionarios.

El segundo fichero a modificar se encuentra en la API, y es el encargado de realizar la petición al servidor de búsqueda. Por lo tanto, el fichero es el llamado *resultController.js*, dentro del directorio *controllers* de la API. La función es la llamada *search*, cuya parte a modificar se encuentra en el extracto siguiente:

```
1 var msg = {
2     "user": b.USERNAME,
3     "url": b.URL,
4     "search_options": [
5         {
6             "number": 1,
7             "module": "crawler"
8         },
9         {
10            "number": 2,
11            "module": "sqlinjection"
12        },
13        {
14            "number": 3,
15            "module": "csrf"
16        }
17    ]
18 }
```

Se trata de la petición a enviar al servidor. Bastaría con añadir un nuevo elemento al objeto JSON, indicando el número asignado al módulo y su nombre, siendo este nombre el mismo que el del directorio que contiene el módulo.

Para evitar la existencia de módulos muy extensos y sobrecargados, se recomienda que cada módulo trabaje con un máximo de 10 vulnerabilidades.

Por lo tanto, en la tabla *vulnerability_types* de la base de datos general, los identificadores de las vulnerabilidades están gestionadas en tramos de 10: los número 0 al 9 están reservados para el módulo de inyección SQL, del 10 al 19 para el módulo CSRF, y así sucesivamente.

7.6. Blog y wiki

El blog y la wiki funcionan bajo una versión modificada de DokuWiki para que funcionen junto con la API desarrollada.

Se debe instalar la herramienta y configurarla. Tras ello, se debe cambiar el fichero PHP que realiza la gestión de usuarios por el modificado.

Las modificaciones realizadas son dos:

- Llevar el registro de un nuevo usuario hacia el portal de búsqueda de Krashr, evitando al de la propia wiki.
- Identificar a los usuarios a través de la API, pero manteniendo la interfaz de identificación proporcionada por DokuWiki. El usuario acabará conectado a DokuWiki si la API de Krashr aprueba su conexión.

8. Legislación

La aplicación debe respetar la legislación actual. Su desconocimiento no exime de su cumplimiento.

Dos son las leyes que afectan al proyecto: la LOPD y la LSSI.

La LOPD (Ley Orgánica de Protección de Datos) afecta a Krashr porque almacena datos correspondientes al nivel básico de los usuarios: sus nombres y apellidos. Además, la LOPD contempla el derecho de cancelación, por lo que ha añadido la opción de que un usuario elimine su cuenta. No se ha realizado el registro del fichero puesto que se trata de un proyecto académico que, en principio, no será puesto en funcionamiento.

A su vez, este proyecto también se ve afectado por la LSSI (Ley de Servicios de la Sociedad de la Información) por el uso de las llamadas *cookies*. Una *cookie* es un pequeño fichero de información enviada por un sitio web y almacenada en el navegador del usuario, de manera que el sitio web puede consultar la actividad previa del usuario. Las *cookies* se utilizan también para realizar seguimientos de usuarios a lo largo de un sitio web.

Tres son los “documentos” que se han realizado para el cumplimiento de la legislación vigente: condiciones del servicio, políticas de privacidad y políticas de *cookies*.

- **Condiciones del servicio:** explican cada uno de los derechos y obligaciones del proyecto y los usuarios. Para realizar este texto, se han leído las condiciones del servicio realizadas por empresas y servicios muy usados como Google o Twitter.
- **Políticas de privacidad:** describen cómo el proyecto conserva y usa la información personal de los usuarios de la página web. Al igual que el punto anterior, se han leído las políticas de privacidad realizadas por empresas como Google o Twitter.
- **Políticas de *cookies*:** describen cómo se hacen uso de las *cookies* en el proyecto. Para la realización de este texto, se ha hecho uso de una plantilla creada por wikiHow²⁹.

²⁹Plantilla: <http://www.wikihow.com/Sample/WikiHow-Cookie-Policy>.

Por último, se ha incluido un aviso sobre el uso de las *cookies* (Figura 36) cuando un usuario entra por primera vez a la página. Para ello, se ha utilizado una librería externa, llamada Cookie Consent³⁰, que únicamente se le debe especificar los textos a mostrar y el enlace a la página con las políticas de *cookies*. Con ello, la librería crea un fragmento de código que se debe añadir al código del *front-end* para que aparezca.

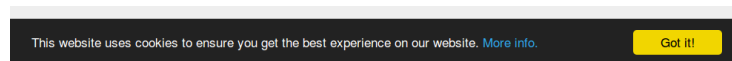


Figura 36: Aviso sobre el uso de las *cookies*.

³⁰Más acerca de Cookie Consent en: <http://silktide.com/cookieconsent>.

9. Pruebas

En esta sección se explicarán las pruebas realizadas sobre los distintas partes que componen el proyecto, su contexto y, dependiendo del tipo de prueba, sus resultados.

9.1. *Front-end* y API

Las pruebas del *front-end* y API son pruebas de caja negra, es decir, se comprobarán las respuestas que produce el *front-end* dependiendo de las entradas que se introduzcan, sin tener en cuenta su funcionamiento interno.

Se ha decidido realizar únicamente este tipo de pruebas por la dificultad añadida que supone la realización de pruebas de caja blanca por el uso de las tecnologías, y el gran número de librerías totales, que se ha escogido para la realización de esta parte del proyecto.

Nota: las pruebas de la página principal son las últimas que se han realizado.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	Introducir una dirección errónea	Aviso alertando del error	“Realiza” la búsqueda.
1.b	Introducir una dirección errónea	Aviso alertando del error	Todo correcto. Cambiados los <i>button</i> por <i>submit</i> en todos los formularios.
2.a	Buscar vulnerabilidades sin estar conectado	Cargar la pantalla de conexión	Se ha decidido mostrar un aviso.
3.a	Buscar vul. estando conectado	Cargar la pantalla de búsqueda	Todo OK.
4.a	Buscar vul. con URL incorrecta	Mostrar aviso	No muestra nada.
4.b	Buscar vul. con URL incorrecta	Mostrar aviso	Todo OK. Aviso mal configurado.

Tabla 11: Pruebas de la página principal.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	Introducir unos datos erróneos a la hora de conectar	Aviso alertando del error	Todo OK.
2.a	Introducir unos datos válidos a la hora de conectar	Conectar y pasar a la página de la cuenta personal	No hace nada.
2.a	Introducir unos datos válidos a la hora de conectar	Conectar y pasar a la página de la cuenta personal	Todo OK. Nombre de la función equivocado.
3.a	No introducir todos los datos al registrarse	Avisar del error	Todo OK.
4.a	No introducir datos correctos al registrarse	Avisar del error	Intenta el registro.
4.b	No introducir datos correctos al registrarse	Avisar del error	Todo OK. Se había olvidado una comprobación.
5.a	Las contraseñas no coinciden	Avisar del error	Todo OK.
6.a	Introducir todos los datos al registrarse	Registrar, conectar y pasar a la página de la cuenta personal	No hace nada.
6.b	Introducir todos los datos al registrarse	Registrar, conectar y pasar a la página de la cuenta personal	Todo OK. Había un fallo a la hora de llamar a la API, se llamaba a una función inexistente con los parámetro, además, en diferente orden.

Tabla 12: Pruebas de la página de conexión/registro.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	No introducir todos los datos necesarios	Avisar del error	Todo OK.
2.a	Introducir una contraseña incorrecta	Avisar del error	Todo OK
3.a	Introducir un correo erróneo	Avisar del error	No hace nada.
3.b	Introducir un correo erróneo	Avisar del error	Todo OK. El nombre del aviso estaba mal escrito.
4.a	Introducir datos correctos	Actualizar y mostrar los datos	Todo OK.

Tabla 13: Pruebas de la página de la cuenta personal.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	No introducir todos los datos necesarios	Avisar del error	Todo OK.
2.a	Contraseña antigua no coincide	Avisar del error	Todo OK.
3.a	Contraseñas nuevas no coinciden	Avisar del error	Cambia la contraseña.
3.b	Contraseñas nuevas no coinciden	Avisar del error	No se había realizado la comprobación.
4.a	Introducir datos correctos	Actualizar la contraseña y cargar la página de la cuenta personal	Da error.
4.b	Introducir datos correctos	Actualizar la contraseña y cargar la página de la cuenta personal	Todo OK. Había un <i>IF</i> mal cerrado en la función.

Tabla 14: Pruebas de la página de cambio de contraseña.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	Cargar página sin tener una búsqueda activa	Aviso sobre la no existencia de una búsqueda	Todo OK.
2.a	Cargar página con una búsqueda activa	Mostrar estado de la búsqueda	Muestra aviso de que no hay búsqueda.
2.b	Cargar página con una búsqueda activa	Mostrar estado de la búsqueda	Se queda el aviso de cargando estado.
2.c	Cargar página con una búsqueda activa	Mostrar estado de la búsqueda	Funciona. No funcionaba el <i>socket</i> , no envía JSON. Usando nueva librería llamada JSON-Socket.

Tabla 15: Pruebas de la página de búsqueda.

Nº	Acción	Resultado esperado	Resultado obtenido
1.a	Cargar página sin tener ninguna búsqueda realizada	Mostrar aviso	Todo OK.
2.a	Cargar la página con una búsqueda de un único resultado	Mostrar resultado	Se queda el aviso de cargando datos.
2.b	Cargar la página con una búsqueda de un único resultado	Mostrar resultado	Todo OK. Mala realización de la carga de datos desde MongoDB.
3.a	Cargar la página con varios resultados	Mostrar resultados	Los muestra, pero no sale la información de las vulnerabilidades.
3.b	Cargar la página con varios resultados	Mostrar resultados	Todo OK. Se debían poblar los datos usando otra función de Mongoose.
4.a	Agrupar los resultados por proceso	Mostrar resultados agrupados	Se para el servidor, ha dado error en la función. MongoDB no agrupa los datos junto con la población de datos.
4.b	Agrupar los resultados por proceso	Mostrar resultados agrupados	No se agrupa con la librería escogida.
4.c	Agrupar los resultados por proceso	Mostrar resultados agrupados	Todo OK. Usada la librería Underscore.

Tabla 16: Pruebas de la página de resultados.

9.2. *Back-end*

En cambio, en esta sección sí que se han realizado pruebas de caja blanca usando la librería *unittest* de Python, que es similar a las pruebas realizadas en Java con JUnit durante el transcurso de la carrera. Esta librería ha sido usada para realizar las pruebas del servidor de búsqueda.

En cambio, para realizar las pruebas de la base de datos general, se han realizado las llamadas a los procedimientos almacenados usando la consola de la herramienta pgAdmin para ver si funcionaban, o no, de manera visual.

No hay secciones de resultados, puesto que se han realizado las pruebas tantas veces como ha hecho falta hasta que cada uno de los procedimientos y métodos funcionasen correctamente.

9.2.1. Base de datos general

En las siguientes pruebas se muestran las pruebas realizadas a los procedimientos almacenados de la base de datos (capítulo Desarrollo, sección *back-end*, subsección Base de datos general) con sus nombres en castellano, junto con el contexto en las que se han realizado.

Nº	Procedimiento	Contexto
1.a	Estado actual del proceso	Usuario con proceso en marcha
1.b	Estado actual del proceso	Usuario sin proceso en marcha
2.a	Estado de un proceso	Datos incorrectos
2.b	Estado de un proceso	Usuario inexistente
2.c	Estado de un proceso	Proceso inexistente
2.d	Estado de un proceso	Usuario y proceso correctos
3.a	Nuevo proceso	Usuario con proceso en marcha
3.b	Nuevo proceso	Usuario sin proceso en marcha y URL inexistente
3.c	Nuevo proceso	Usuario sin proceso en marcha y URL existente
4.a	Actualizar proceso	Búsqueda terminada
4.b	Actualizar proceso	Actualizar estado
5.a	Vulnerabilidad encontrada	Web inexistente
5.b	Vulnerabilidad encontrada	Web existente

Tabla 17: Pruebas de la base de datos general.

9.2.2. Núcleo del servidor de búsqueda

Nº	Método	Contexto
1.a	Lectura de petición	Petición correcta de búsqueda
1.b	Lectura de petición	Petición correcta de estado de un proceso
1.c	Lectura de petición	Petición incorrecta
2.a	Comprobación de URL	URL correcta
2.b	Comprobación de URL	URL incorrecta
3.a	Comprobación de módulos	Sólo el <i>crawler</i>
3.b	Comprobación de módulos	Sólo el de inyección SQL
3.c	Comprobación de módulos	Sólo el de CSRF
3.d	Comprobación de módulos	Todos

Tabla 18: Pruebas del núcleo del servidor.

9.2.3. Módulo de *webcrawling*

Nº	Método	Contexto
1.a	<i>Fetching</i>	Web sin vínculos
1.b	<i>Fetching</i>	Web con vínculos
1.c	<i>Fetching</i>	Web con vínculos repetidos
2.a	<i>Crawling</i> de URL	Web sin vínculos
2.b	<i>Crawling</i>	Web con vínculos
2.c	<i>Crawling</i>	Web con vínculos repetidos

Tabla 19: Pruebas del módulo de *webcrawling*.

9.2.4. Módulo de inyección SQL

Nota: existen algunas pruebas en la que aparecen guiones (“-”) que significan que son independiente del contexto.

Nº	Método	Contexto
1.a	Búsqueda de inyecciones	Web sin fallos
1.b	Búsqueda de inyecciones	Web con fallo de autenticación
1.c	Búsqueda de inyecciones	Web con inyección ciega
1.d	Búsqueda de inyecciones	Web con inyección basada en error
1.e	Búsqueda de inyecciones	Web con todas las inyecciones
2.a	Guardar resultado	-
3.a	Enviar resultado	-

Tabla 20: Pruebas del módulo de inyección SQL.

9.2.5. Módulo de CSRF

Nota: existen algunas pruebas en la que aparecen guiones (“-”) que significan que son independiente del contexto.

Nº	Método	Contexto
1.a	Búsqueda de CSRF	Web sin fallo
1.b	Búsqueda de CSRF	Web con fallo
2.a	Guardar resultado	-
3.a	Enviar resultado	-

Tabla 21: Pruebas del módulo de CSRF.

9.2.6. Adaptador de la base de datos del servidor de búsqueda

Nota: existen algunas pruebas en la que aparecen guiones (“-”) que significan que son independiente del contexto.

Nº	Procedimiento	Contexto
1.a	Estado actual del proceso	-
2.a	Estado de un proceso	-
3.a	Nuevo proceso	-
4.a	Actualizar proceso	-
5.a	Vulnerabilidad encontrada	-

Tabla 22: Pruebas del adaptador de la base de datos.

10. Conclusiones

Este capítulo trata de echar la vista atrás con una mirada crítica para puntualizar aquellos aspectos que no se han cumplido durante el desarrollo de este TFG con vistas a mejorar la planificación inicial en futuros proyectos.

Además, se valorarán las posibles ampliaciones a realizar en un futuro y se finalizará con la exposición de las conclusiones personales por parte del desarrollador.

10.1. Cambios realizados durante el desarrollo

Aunque la planificación estaba bien desgranada, sí que se han tenido que realizar cambios. Es complicado acertar con las partes a realizar desde un primer momento, puesto que es muy fácil realizar cambios durante el transcurso de cualquier proyecto.

Los cambios realizados han sido los siguientes:

- El más evidente, fue la equivocación de llamar al módulo de CSRF como “configuración de seguridad incorrecta”, por lo que se tuvo que cambiar el nombre a bastantes ficheros (y cambiar esta memoria) durante la implementación de código.
- El módulo que, en principio, iba a estar compuesto por la búsqueda de las vulnerabilidades de inyección SQL y CSRF, se decidió separarlo en dos módulos diferentes.
- Aunque en la planificación se escribió que el *webcrawling* se realizaría mediante un módulo, durante la fase de captura de requisitos se decidió que era mejor que formara parte del núcleo del servidor (ver Figura 9). Decidir que fuera parte del núcleo fue un error, puesto que no todos los usuarios iban a hacer uso del mismo. Hasta la implementación no se cayó en cuenta de este error cometido, por lo que, a partir de ese momento, el *webcrawling* pasó a ser de nuevo un módulo del servidor de búsqueda.
- La clase que recibiría las peticiones de la API en el servidor se pasó por alto durante toda la planificación, teniendo que buscar información sobre ello durante la implementación para poder realizarla.

- Las función de cerrar sesión se añadió durante la implementación del *front-end* cuando el desarrollador se dio cuenta de que no podía cambiar de un usuario a otro a la hora de realizar diferentes pruebas.
- La función de eliminar fue añadida en la etapa final del desarrollo, durante la lectura de la LOPD que contempla el derecho de cancelación. Como no se factura nada, no se está obligado a guardar los datos de los usuarios los 5 años que obliga Hacienda.

Por lo tanto, se ha decidido rehacer el diagrama de clases del servidor de búsquedas, Figura 37, para mostrar cómo que ha quedado tras terminar la implementación y, de esta forma, poner el nombre de todos los métodos que faltaban en el anterior diagrama (ver Figura 16).

Para evitar confusiones, se han añadido colores para diferenciar las dependencias de cada uno de los módulos.

Nota: las llamadas desde el núcleo a los módulos no se han insertado para no recargar el diagrama.

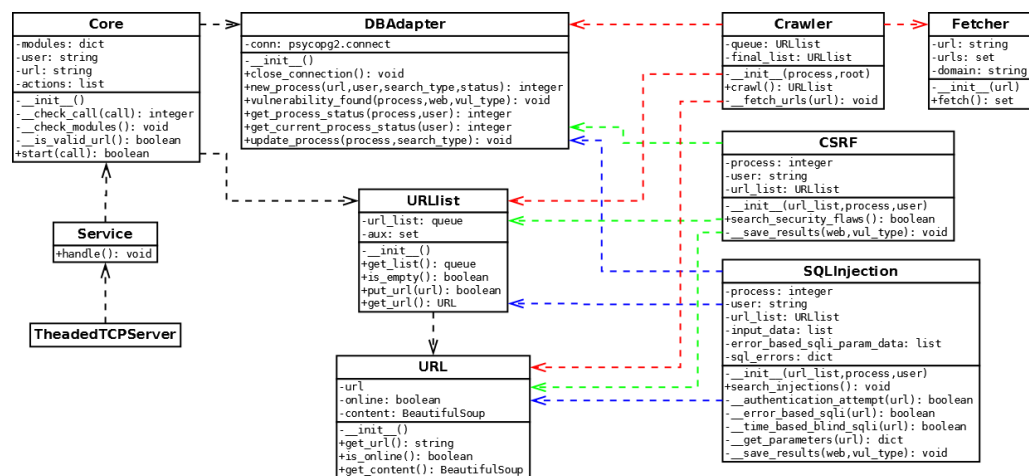


Figura 37: Diagrama de clases final del servidor de búsquedas.

10.2. Planificación final

Se ha de comentar que se han alcanzado todos los objetivos iniciales planteados en la planificación inicial del proyecto (sección Objetivos). Sin embargo, se han invertido menos horas de lo previsto inicialmente. En total, la desviación es de unas 30 horas de menos sobre las 572 horas iniciales, lo que supone un decremento de un 5 %.

La finalización del proyecto se ha retrasado poco más de un mes, puesto que ha sido imposible mantener el ritmo semanal de 40 horas.

En la Figura 38 se recogen las horas reales dedicadas a cada proceso haciendo una comparativa con las planificadas previamente.

Nombre de tarea	Duración	Realidad
Portal de búsqueda de vulnerabilidades	572 horas	542 horas
1 Organización y aprendizaje	116 horas	100 horas
1.1 Planificación de las tareas	8 horas	10 horas
1.2 DOP	30 horas	36 horas
1.3 Instalación de sistemas de desarrollo	18 horas	23 horas
1.4 Aprendizaje de nuevo lenguaje (Python)	40 horas	20 horas
1.5 Aprendizaje sobre realización de una API Rest	8 horas	6 horas
1.6 Aprendizaje sobre MEAN Stack	10 horas	3 horas
1.7 Creación y puesta a punto del repositorio GIT	2 horas	2 horas
2 Realización del back-end de la aplicación (BBDD y Módulos)	229 horas	208 horas
2.1 Realización del núcleo de la aplicación y comprobación de URL	28 horas	17 horas
2.2 Realización del módulo de lectura del árbol de URL	20 horas	7 horas
2.3 Realización del módulo de inyección SQL y configuración de seguridad	20 horas	15 horas
2.4 Diseño de base de datos general	6 horas	3 horas
2.5 Implantar base de datos y comprobar su funcionamiento	5 horas	5 horas
2.6 Implementación del código	150 horas	161 horas
3 Realización del front-end (Web) y API de la aplicación	100 horas	112 horas
3.1 Diseño de casos de uso	8 horas	6 horas
3.2 Diseño de modelo de dominio	4 horas	2 horas
3.3 Diseño de base de datos no relacional	3 horas	2 horas
3.4 Diseño de esquema REST	5 horas	4 horas
3.5 Diseño de la interfaz gráfica	30 horas	41 horas
3.6 Implementación del código	50 horas	57 horas
4 Implantación (API, Web y Módulos)	6 horas	20 horas
5 Wiki y Blog	13 horas	19 horas
6 Leyes	42 horas	6 horas
7 Documentación	66 horas	77 horas

Figura 38: Tabla comparativa de la duración inicial y real del proyecto.

Hagamos un pequeño balance (cada punto se refiere a cada punto principal de la Figura 38):

- 1. Organización y aprendizaje:** el tiempo ahorrado en este apartado se ha visto en el aprendizaje sobre la realización de una API. Los apartados 1.5 y 1.6 son casi redundantes, por lo que parte de esas horas ahorradas han ido a la escritura del DOP. Aprender Python tampoco supuso los problemas que se pensaban.
- 2. Realización del *back-end* de la aplicación (BBDD y Módulos):** el ahorro de tiempo ha venido en que no ha hecho falta la realización

de varios diagramas, como pueden ser los diagramas de clase de los módulos (puesto que no requerían más que una o dos clases cada uno), o los modelos de dominio porque iban junto al del núcleo. La implementación sí que se ha visto incrementada en tiempo, por los problemas ocasionados con la realización de partes no previstas.

3. **Realización del *front-end* (Web) y API de la aplicación:** los análisis y diseño de la API y *front-end* no fueron tan complicados, sin embargo, se hicieron más de un prototipo, lo que conllevó dedicar más tiempo del esperado. Además, también se ha tardado más tiempo en la implementación, y podría haber sido peor de no haber realizado los prototipos con el *framework* Bootstrap.
4. **Implantación (API, Web y Módulos):** la implantación ha llevado más tiempo del necesario por problemas ocasionados con la Raspberry Pi y de la no lectura de *sockets* por parte de la máquina donde se tiene el servidor de búsqueda.
5. **Wiki y Blog:** la mayoría de tiempo dedicado a este apartado ha sido la modificación de la herramienta de creación de wiki.
6. **Leyes:** se ha dedicado el tiempo mínimo para tratar de ajustarse a la legislación actual. No se ha profundizado.
7. **Documentación:** la documentación ha llevado más tiempo del esperado.

10.2.1. Reevaluación económica

Aprovechando los datos de la Tabla 6 que indica los gastos totales y sabiendo, por lo tanto, los gastos por hora, se puede crear una nueva tabla con los gastos finales totales (Tabla 23).

El cálculo original se realizó en meses por coincidir el número de total de horas casi con el número de meses a tardar en el desarrollo del TFG. Como se ha tardado menos horas de las esperadas, aún habiendo dedicado un mes más de trabajo, el gasto final será inferior al calculado inicialmente.

Concepto	Gasto inicial con 572h	Gasto final con 542h
Mano de obra	7698,45 €	7294,69 €
Hardware	263,59 €	249,77 €
Gastos indirectos	395,19 €	374,46 €
Total	8357,23 €	7918,92 €

Tabla 23: Gasto final total.

El ahorro entre los gastos previstos y los reales es de 438,31 €.

10.3. Resultado final

El resultado final se podrá ver, una vez adaptado para su uso en cualquier ordenador, en el GitHub del desarrollador del proyecto³¹.

Además, existe un vídeo explicando el funcionamiento de la herramienta y se puede encontrar en la siguiente dirección: <https://www.dropbox.com/sh/3zoo6vzihxt6w3w/AAA1aRIB1goTWkWwVR5W09nUa?dl=0>.

10.4. Líneas futuras

Se pueden realizar muchas mejoras a la aplicación. Aunque el trabajo realizado ha sido más que satisfactorio, siempre hay campos de mejora. Además, existen mejoras que se pensaron en añadir durante la escritura del DOP, pero que fueron desechadas por tener que dedicar más tiempo del que se disponía en prepararlas.

Por lo tanto, a continuación se expondrán las mejoras que se tienen pensadas en realizar si se continua con el desarrollo de esta aplicación:

- Soporte multilenguaje. Como bien se puede observar, la página web se encuentra única y exclusivamente en inglés. Sería una mejora muy buena para llegar a más usuarios potenciales, aunque se tendría que modificar todo el *front-end* con llamadas a ficheros en formato JSON que contengan las diferentes traducciones, además de cambiar los controladores para seleccionar el JSON adecuado.
- Actualizar el estado de la pantalla de búsqueda activa automáticamente y, de terminar la búsqueda, redirigir al usuario automáticamente a la

³¹Perfil de GitHub: <https://github.com/0xJCG>.

pantalla de resultados. No supone una mejora sustancial, pero es una mejora enfocada en el usuario. Se debería mantener un canal abierto en todo momento notificando de cualquier cambio, lo que supondría un esfuerzo demasiado grande.

- Añadir nuevos módulos que busquen otras vulnerabilidades (p. ej. XSS) o actualizar los existentes. No es complicado, dependiendo de la vulnerabilidad, y sería una mejora muy buena.
- Mejorar las llamadas a páginas web, es decir, no realizar una búsqueda en una web en la que ya se ha realizado una búsqueda anteriormente y ésta no se haya actualizado desde entonces. La base de datos está preparada para alojar los datos necesarios, pero haría falta modificar o añadir algún procedimiento almacenado. El tiempo de ejecución podría bajar y la experiencia de usuario mejorar considerablemente.
- Mejorar el módulo de *webcrawling* haciendo que no guarde páginas similares. Supondría realizar varias comprobaciones lo que puede llevar a realizar más pruebas para comprobar su funcionamiento; la mejora sería sustancial y ahorraría el tener que realizar búsquedas innecesarias en bastantes páginas. Para entender de qué se trata esta mejora, se explicará mediante ejemplos:
 - Una página con una lista de productos. Hacer que el *crawler* no guarde más de un producto, puesto que sólo cambia el producto en sí.
 - Si ya se tiene la raíz guardada, que no guarde también la dirección que contenga el recurso *index* del mismo por tratarse de la misma página.
- Permitir hacer una búsqueda personalizada con los módulos a usar. No sería muy complicado y daría más libertad a los usuarios.
- Hacer el salto de Angular 1.x a 2.x. Un cambio demasiado complicado que apenas supone una mejora para el usuario, por lo que no merece la pena.
- Mejorar la interfaz con ideas propuestas por los potenciales usuarios (botón para que vuelva a buscar de nuevo, ejemplos de búsquedas, etc).

Las mejoras que se sugieren realizar antes que las otras serían: la mejora de las llamadas a páginas web, la mejora del módulo de *webcrawling* y la

realización de algún módulo nuevo que realice búsquedas de otras vulnerabilidades diferentes a las que ya se buscan.

10.5. Reflexión personal

Se me hace difícil echar la vista atrás y pensar en todo lo que he realizado para llegar hasta donde estoy ahora mismo. Ha sido un camino duro, lleno de baches, largo y pesado, pero ver el resultado hace que se me olviden todas las penas; ha merecido la pena.

Decidí realizar este trabajo porque el ámbito de la seguridad informática es algo que me llama la atención desde hace tiempo. Sé que este proyecto no es más que una semilla, el mundo de la seguridad informática es muy amplio y no he hecho más que arañar la punta del iceberg. Existe mucha información sobre este mundo, pero cuesta encontrar algo de fácil digestión. La idea era tratar de empezar algo con lo que un desarrollador novel pudiera entender de manera relativamente sencilla las vulnerabilidades que se explotan en el proyecto y ayudarle a empezar a “securizar” sus páginas web ante estos fallos graves que, aunque son relativamente sencillos de reparar, pueden hacer mucho daño.

Pero yo también he sido novel y he tenido que adentrarme en Internet en busca de ayuda para poder superar las trabas y continuar con el desarrollo del proyecto. Páginas como Stack Overflow³² me han sido de gran ayuda cuando me atascaba, sobre todo, con Python.

Empezar con un lenguaje de programación nuevo –nuevo para mi– no es sencillo, sobre todo cuando la forma con la que se realizan las cosas cambian con respecto a lo que ya estás acostumbrado. De todas formas, me ha gustado esta experiencia con Python 3.x y me he dado cuenta del gran lenguaje que es y la potencia que tiene, aunque en ocasiones me he arrepentido de no haber escogido la versión 2.x por todo el soporte que esta versión cuenta.

Tampoco había usado nunca PostgreSQL y, aunque a primera vista pgAdmin me ha resultado extraño, enseguida he cogido soltura con la herramienta. Además, he tenido que desempolvar mis conocimientos en PL/SQL.

Por otra parte, resulta muy fácil realizar una API usando Node.js y Express, aunque la mayor parte de los problemas los he tenido con MongoDB y la librería Mongoose. Menos mal que siempre estará la comunidad para echarte una mano; si tienes una duda, seguro que hay alguien que la ha tenido antes y que ha escrito la solución.

³²Página principal de Stack Overflow: <http://stackoverflow.com/>.

De AngularJS prefiero no decir nada, su funcionamiento es como si fuese magia. Funciona y punto, mejor no tocar nada.

Otra sorpresa me la he llevado al empezar con Bootstrap, es tan sencillo *maquetar* páginas web con este *framework* que decidí realizar los prototipos con ella en vez de continuar con la herramienta de prototipado por ser más fácil de usar aunque requiriese escribir código. Vas a su página, buscas el componente que quieres, copias, lo pegas en tu proyecto, lo adaptas al mismo y listo, funciona.

Sin duda alguna, una de las partes más complicadas ha sido la de la realización de llamadas desde la API hacia el servidor de búsquedas por todos los problemas que he tenido en sendas partes. No estaría mal montarme un blog, por llamarlo de alguna manera, comentando los problemas que he tenido durante el desarrollo para ayudar a otros desarrolladores cuando tengan los mismos problemas.

En cuanto a la planificación, el haber dividido tanto el proyecto ha sido un acierto, sobre todo por haber podido saltar de una parte a otra cuando me quedaba estancado. Además, aunque me he quedado corto en la estimación total de horas, me he quedado cerca. Los análisis y diseño no me han llevado tanto tiempo, pero ese tiempo lo he tenido que dedicar a la implementación de la que sí me he quedado corto en estimación. Tampoco había realizado un proyecto de esta envergadura y para ser la primera vez, estoy contento con la planificación realizada.

He aprendido mucho, no creía que el resultado fuese a ser tan satisfactorio. Amigos y compañeros me han felicitado por lo bien que ha quedado el proyecto (cuando les realizaba las encuestas) que, aunque simple y limitado en opciones, hace lo que tiene que hacer y lo hace rápido, algo que, de verdad, no esperaba que fuese así.

Me gustaría agradecer la ayuda de mi Director de proyecto en todos los aspectos en los que me ha hecho falta (sobre todo a la hora de la escritura de esta memoria), de otros profesores cuando me surgían dudas relacionados con sus ámbitos y, por último, a amigos y compañeros por tener que aguantarme durante las entrevistas y encuestas.

Espero que la lectura de esta memoria no te haya sido muy densa y que hayas aprendido tanto como lo he hecho yo escribiéndola. Gracias.

Bibliografía

- [1] Zetter, Kim. “Ashley Madison CEO Resigns in Wake of Hack, News of Affairs”. 28 de agosto de 2015. Disponible en: <http://www.wired.com/2015/08/ashley-madison-ceo-resigns-wake-hack-news-affairs/>.
- [2] Fernández, M. y Romero, P. “Sony admite que un intruso accedió a datos de millones de usuarios de PlayStation”. 27 de abril de 2011. Disponible en: <http://www.elmundo.es/elmundo/2011/04/27/navegante/1303859686.html>.
- [3] Ministerio de Trabajo e Inmigración. “Boletín Oficial del Estado - XVI Convenio colectivo estatal de empresas de consultoría y estudios de mercados y de la opinión pública”. 4 de abril de 2009. Disponible en: <https://www.boe.es/boe/dias/2009/04/04/pdfs/B0E-A-2009-5688.pdf>.
- [4] Cloudream. “*Should I use Python 2 or Python 3 for my development activity?*”. 13 de abril de 2014. Disponible en: <https://wiki.python.org/moin/Python2orPython3>.
- [5] Ríos Ch, Álvaro José. “*Java vs Python*”. 22 de enero de 2015. Disponible en: <http://www.webbizarro.com/noticias/1428/java-vs-python/>.
- [6] Google. “*AngularJS, HTML enhanced for web apps!*”. Disponible en: <https://angularjs.org/>.
- [7] Facebook. “*React, a JavaScript library for building user interfaces*”. Disponible en: <https://facebook.github.io/react/>.
- [8] Wheeler, Ken. “*Getting To Know Flux, the React.js Architecture*”. 28 de octubre de 2014. Disponible en: <https://scotch.io/tutorials/getting-to-know-flux-the-react-js-architecture>.
- [9] Drifty Co. “*Ionic framework*”. Disponible en: <http://ionicframework.com/>.
- [10] Otto, Mark y Thornton, Jacob. “*Bootstrap is the most popular HTML, CSS, and JS framework for developing responsive, mobile first projects on the web*”. Disponible en: <http://getbootstrap.com/>.
- [11] Python Software Foundation. “*21.21. socketserver - A framework for network servers*”. Disponible en: <https://docs.python.org/3.4/library/socketserver.html>.

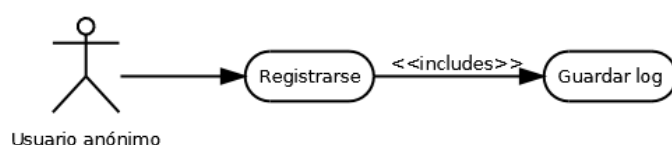
- [12] Fidanov, Stefan. “*Best practices for Express app structure*”. 25 de agosto de 2014. Disponible en: <https://www.terlici.com/2014/08/25/best-practices-express-structure.html>.
- [13] Nielsen, Jakob. “*10 Usability Heuristics for User Interface Design*”. 1 de enero de 1995. Disponible en: <http://www.nngroup.com/articles/ten-usability-heuristics/>.

Anexos

A. Anexo I: Casos de uso extendidos

A.1. *Front-end*

A.1.1. Registrarse



Nombre	Registrarse
Descripción	El usuario debe tener cuenta en el sistema para poder realizar búsquedas.
Actores	Usuario anónimo.
Precondiciones	El correo electrónico y el nombre de usuario no deben estar ya registrados en el sistema.
Requisitos funcionales no	Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. El usuario se encuentra en la interfaz de la Figura 39 en la que aparecerán los campos a introducir de usuario y contraseña. El usuario rellena los campos y pulsará en “<i>Sign up</i>”. 2. Se comprobará con la base de datos si el nombre de usuario y el correo estén ya en uso, además de comprobar que todos los datos estén correctamente rellenos. De estar todo correcto, se le redireccionará a la interfaz de la Figura 40. 3. En caso de que los datos introducidos sean incorrectos o no se hayan insertado todos, aparecerá un mensaje de error en la misma interfaz. La Figura 41 muestra uno de esos errores.
Postcondiciones	Una vez registrado, el usuario será conectado al sistema y redireccionado a su página de perfil.

Tabla 24: Caso de uso extendido: registrarse.

The screenshot shows the 'Project Vulpix' login and registration interface. At the top, there's a dark blue header with 'Project Vulpix' on the left and 'Home' and 'Sign in' on the right. Below the header, the page is split into two columns. The left column is titled 'Sign in' and contains a form with 'Username' and 'Password' fields, and a blue 'Sign in' button. The right column is titled 'Sign up' and contains a form with 'Email', 'First name', 'Last name', 'Password', and 'Retype the password' fields, and a blue 'Sign up' button. A small 'Or' text is placed between the two columns.

Figura 39: Pantalla de registro o conexión.

The screenshot shows the 'My profile' page in the 'Project Vulpix' application. The header is dark blue with 'Project Vulpix' on the left and 'Home' and 'Sign in' on the right. Below the header, there's a navigation bar with 'My profile' (active), 'Update password', 'My searches', and 'Now searching...'. The main content area is titled 'My profile' and contains a form with fields for 'Email' (Jonathan), 'jcastro004@ikasle.ehues', 'Jonathan', and 'Castro'. There's also a field for 'Enter your password to update your profile'. At the bottom, there are two buttons: 'Update profile' (blue) and 'Sign out' (red).

Figura 40: Pantalla de perfil de usuario.

The screenshot shows the 'Project Vulpix' login and registration interface, similar to Figure 39, but with an error message. The 'Sign up' column has an orange error banner at the top that says 'All the required data is not inserted.' Below this, the 'Email' field is highlighted with a red border, and a tooltip says 'Rellene este campo.' (Fill this field). The 'Sign up' button is still visible at the bottom.

Figura 41: Pantalla de registro con un error.

A.1.2. Iniciar sesión



Nombre	Iniciar sesión
Descripción	El usuario debe conectarse al sistema para poder realizar búsquedas.
Actores	Usuario anónimo.
Precondiciones	El usuario debe estar previamente registrado en el sistema.
Requisitos no funcionales	Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. El usuario se encuentra en la interfaz de la Figura 39 en la que aparecerán los campos a introducir de usuario y contraseña. El usuario rellena los campos y pulsará en “<i>Sign in</i>”. 2. Se comprobará con la base de datos si la contraseña es correcta y se le redireccionará a la interfaz de la Figura 40. 3. En caso de que los datos introducidos sean incorrectos o no se hayan insertado todos, aparecerá un mensaje de error en la misma interfaz. La Figura 42 muestra uno de esos errores.
Postcondiciones	El usuario se habrá conectado y será redireccionado a su página de perfil.

Tabla 25: Caso de uso extendido: iniciar sesión.

The screenshot displays the 'Project Vulpix' login and sign-up interface. At the top, a dark blue header contains the project name on the left and 'Home' and 'Sign in' links on the right. The main content area is split into two columns by the word 'Or'. The left column, titled 'Sign in', features an orange error banner at the top stating 'All the required data is not inserted.' Below this, there is an email input field containing 'Jonathan', a password input field, and a tooltip pointing to the password field with the text 'Rellene este campo.' The right column, titled 'Sign up', contains five input fields: 'Username', 'Email', 'First name', 'Last name', and 'Password', each followed by a question mark icon. At the bottom of the sign-up section is a 'Retype the password' field with a question mark icon and a blue 'Sign up' button.

Figura 42: Pantalla de conexión con error.

A.1.3. Configurar cuenta

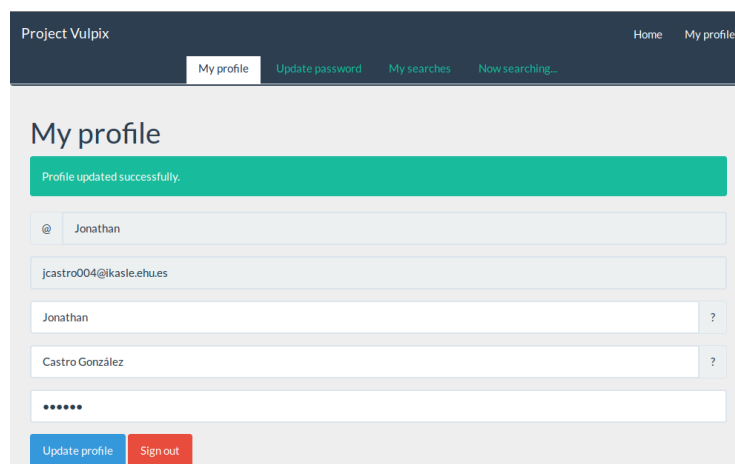
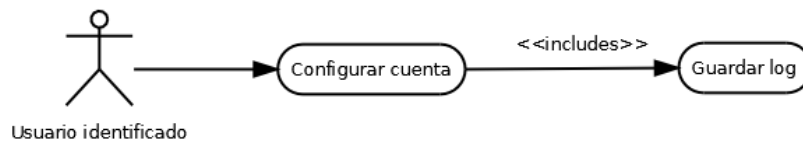


Figura 43: Pantalla de perfil con los datos actualizados.

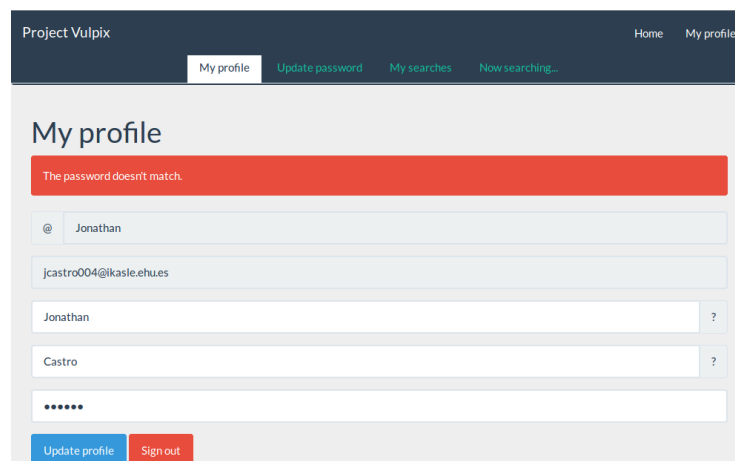


Figura 44: Pantalla de perfil con error.

Nombre	Configurar cuenta
Descripción	El usuario modifica sus datos del perfil.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos funcionales	Ninguno.
Flujo de eventos	<p>Si el usuario desea cambiar sus datos personales del perfil:</p> <ol style="list-style-type: none"> 1. El usuario se encuentra en la interfaz de la Figura 40 en la que aparecerán los campos a introducir. El usuario modificará los datos deseados, introducirá su contraseña y pulsará en “<i>Update profile</i>”. 2. Se comprobará con la base de datos si la contraseña es correcta y se modificarán los datos. Aparecerá un aviso en la misma interfaz, mostrada en la Figura 43. 3. En caso de que los datos introducidos sean incorrectos, aparecerá un mensaje de error en la misma interfaz. La Figura 44 muestra uno de esos errores. <p>En cambio, si lo que desea es cambiar su contraseña:</p> <ol style="list-style-type: none"> 1. El usuario se encuentra en la interfaz de la Figura 45 en la que aparecerán los campos a introducir. El usuario rellena los campos y pulsará en “<i>Update password</i>”. 2. Se comprobará que la contraseña antigua es correcta con la base de datos y si las contraseñas nuevas coinciden. De ser correctos, la contraseña quedará actualizada y se mostrará un aviso (Figura 46). 3. En caso de que los datos introducidos sean incorrectos, aparecerá un mensaje de error en la misma interfaz. La Figura 47 muestra uno de los posibles errores.
Postcondiciones	El usuario habrá modificado los datos deseados de su perfil.

Tabla 26: Caso de uso extendido: configurar cuenta.

Project Vulpix

Home My profile

My profile Update password My searches Now searching...

Update password

Old password for checking

New password ?

Retype the new password ?

Update password

Figura 45: Pantalla de cambio de contraseña.

Project Vulpix

Home My profile

My profile Update password My searches Now searching...

Update password

Password updated successfully.

Old password for checking

..... ?

..... ?

Update password

Figura 46: Pantalla de cambio de contraseña con los datos actualizados.

Project Vulpix

Home My profile

My profile Update password My searches Now searching...

Update password

The new passwords don't match.

.... ?

..... ?

..... ?

Update password

Figura 47: Pantalla de cambio de contraseña con error.

A.1.4. Comprobar vulnerabilidades



Nombre	Comprobar vulnerabilidades
Descripción	El usuario realiza una búsqueda de vulnerabilidades mediante una URL.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos funcionales	no Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. El usuario se encuentra en la interfaz de la Figura 48 en la que aparecerá el campo a introducir la URL. Al rellenar el campo, pulsará en “<i>Search</i>”. 2. Se comprobará con la base de datos si el usuario no está realizando una búsqueda en ese instante y que la URL esté bien escrita. De estar todo correcto, se le redireccionará a la interfaz de la Figura 49. 3. En caso de que la URL sea incorrecta o el usuario no esté conectado, se mostrará un aviso en la misma interfaz, como se muestra en la Figura 50.
Postcondiciones	La búsqueda estará en marcha y el usuario será redireccionado a la página en la que muestra el estado de la búsqueda.

Tabla 27: Caso de uso extendido: comprobar vulnerabilidades.



Figura 48: Pantalla de búsqueda.

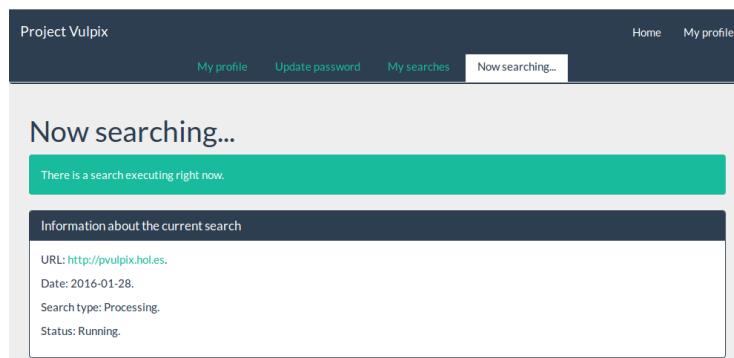


Figura 49: Pantalla de información sobre búsqueda activa.

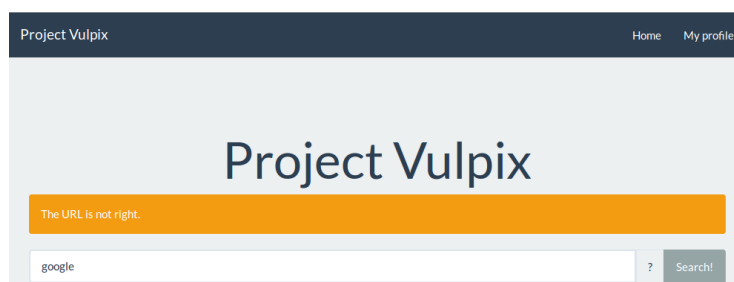
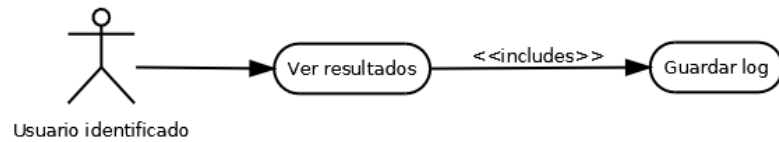


Figura 50: Pantalla de búsqueda con error.

A.1.5. Ver resultados



Nombre	Ver resultados
Descripción	El usuario observa las vulnerabilidades encontradas en sus búsquedas.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos no funcionales	Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. De haberse encontrado vulnerabilidades, éstas aparecerán como se ve en la Figura 51. 2. En caso contrario, se mostrará un aviso en la misma interfaz, como muestra la Figura 52.
Postcondiciones	El usuario verá los resultados obtenidos de las búsquedas realizadas.

Tabla 28: Caso de uso extendido: ver resultados.

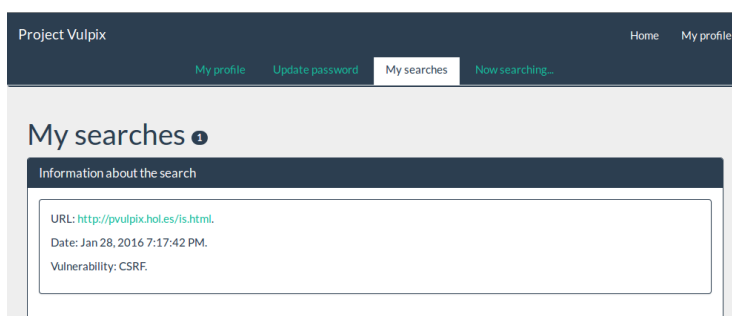


Figura 51: Pantalla de resultados.

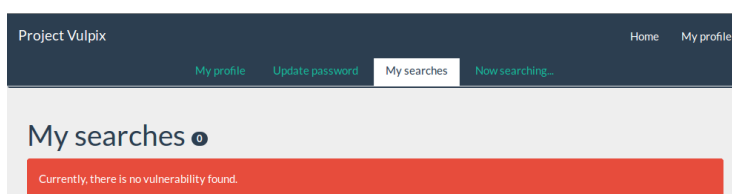
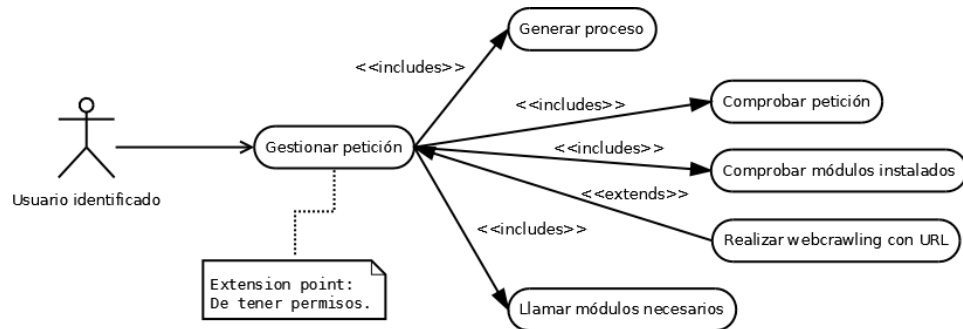


Figura 52: Pantalla de resultados sin búsquedas.

A.2. Back-end

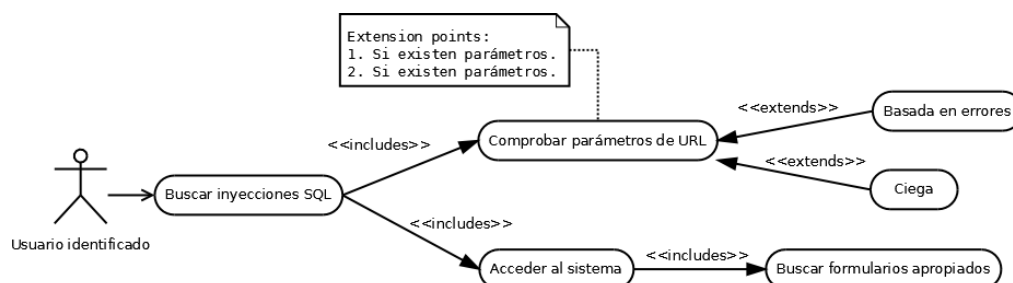
A.2.1. Núcleo



Nombre	Gestionar petición
Descripción	El sistema gestiona la petición del usuario para realizar una nueva búsqueda de vulnerabilidades.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos funcionales	no Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. Se genera un nuevo proceso para tener registrado el estado en el que se encuentra la búsqueda en todo momento. 2. Se comprueba la petición recibida. Si no es correcta, se termina la ejecución. 3. De ser correcta, se comprueban los módulos instalados que requiera la búsqueda. 4. Se realiza el <i>webcrawling</i> mediante la URL que irá suministrada en la petición para buscar el árbol web a partir de la misma. Esta acción se realizará si el usuario tiene los permisos para ello. 5. Se llama a cada uno de los módulos de búsqueda de vulnerabilidades distintos para buscar los distintos fallos en la/s URL/s que se tengan.
Postcondiciones	Los resultados o errores se enviarán al usuario.

Tabla 29: Caso de uso extendido: gestionar petición (núcleo).

A.2.2. Módulo de inyección SQL



Nombre	Buscar inyecciones SQL
Descripción	El módulo busca vulnerabilidades de inyección SQL a través de una, o varias, URL.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos funcionales no	Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. Se comprueba la existencia de parámetros en cada URL. 2. De existir parámetros, se realizarán dos tipos de inyecciones SQL en busca de vulnerabilidades. 3. Si se encuentran errores o las páginas realizan las acciones de la búsqueda, la página web será vulnerable. 4. De no existir parámetros o de haber terminado con estas dos búsquedas, se comprueba la existencia de formularios de conexión en las web de cada URL. 5. De existir estos formularios de conexión, se tratará de autenticar en la página web. 6. De conseguir la autenticación, la página web será vulnerable.
Postcondiciones	Los resultados o errores se enviarán al usuario.

Tabla 30: Caso de uso extendido: buscar inyecciones SQL.

A.2.3. Módulo de CSRF



Nombre	Buscar errores CSRF
Descripción	El módulo busca vulnerabilidades CSRF a través de una, o varias, URL.
Actores	Usuario identificado.
Precondiciones	El usuario debe estar conectado al sistema.
Requisitos no funcionales	Ninguno.
Flujo de eventos	<ol style="list-style-type: none"> 1. Se buscan por formularios dentro del código web de cada URL. 2. De haber formularios, se buscan elementos ocultos en busca de <i>hashes</i>. 3. De no haber dichos elementos o no encontrar <i>hashes</i>, se indicará que dicha web es vulnerable por CSRF. 4. Si existen dichos <i>hashes</i>, la web se considera no vulnerable.
Postcondiciones	Los resultados o errores se enviarán al usuario.

Tabla 31: Caso de uso extendido: buscar errores de CSRF.

B. Anexo II: Diagramas de secuencia

El primer diagrama de secuencia, la Figura 53 que se encuentra a continuación, explica la manera en la que la aplicación recibe una petición del usuario y busca si ese usuario tiene una búsqueda en activo, devolviéndole el número proceso de dicha búsqueda activa.

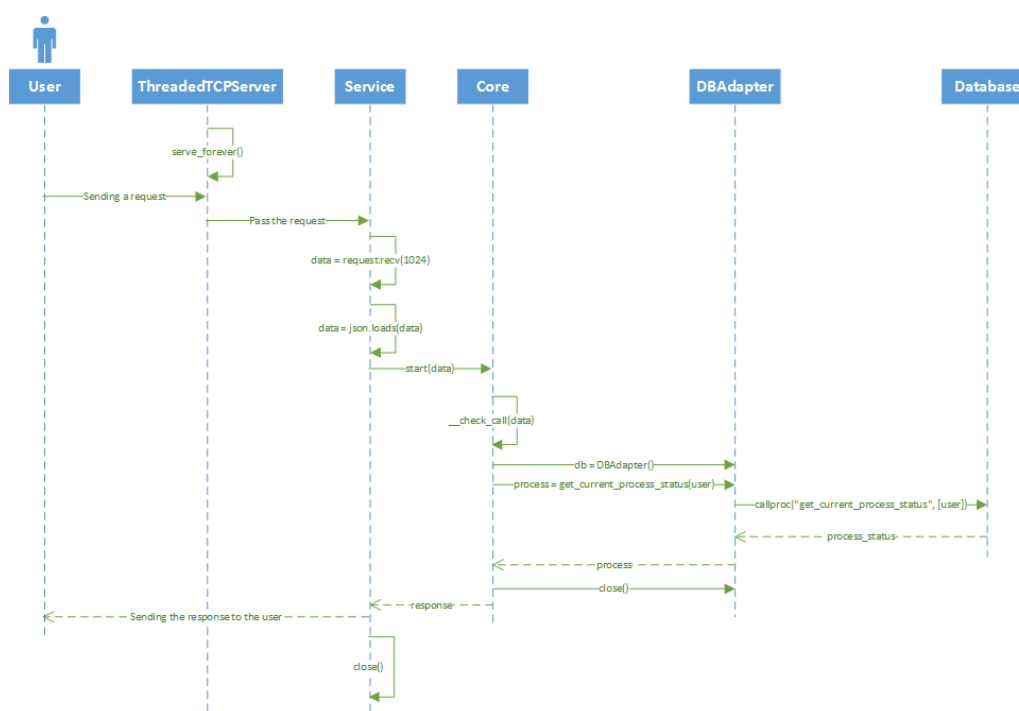
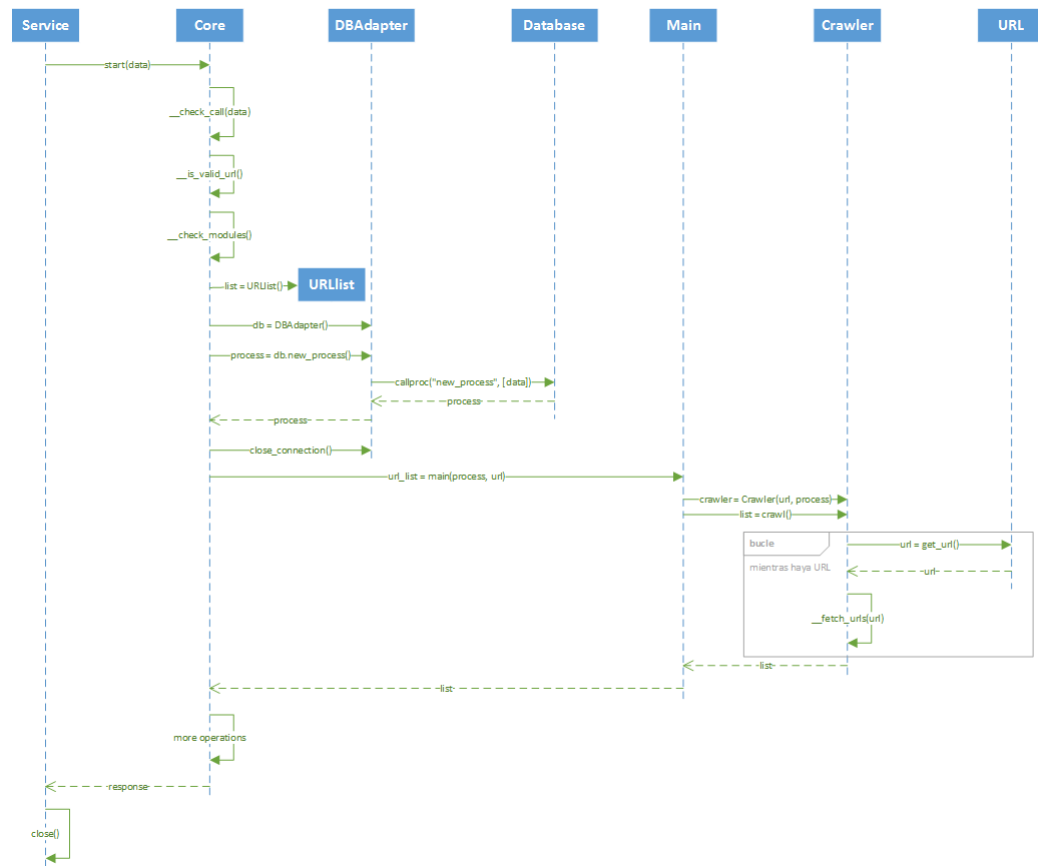


Figura 53: Diagrama de secuencia sobre el estado del proceso.

El siguiente diagrama, Figura 54, explicará qué realiza el sistema para realizar un *webcrawling* a partir de una URL. Se obviará la sección parte de la llamada del usuario hacia el servidor, por ser igual que la del anterior diagrama de secuencia.

Figura 54: Diagrama de secuencia sobre el *webcrawling*.

Por último, se tiene el diagrama de secuencia sobre el módulo de inyección SQL, Figura 55, contando que se realice después de haber llamado al módulo de *webcrawling*, es decir, este diagrama iría en la parte donde pone “*more operations*” del diagrama de la Figura 54.

Se debe mencionar que cada vez que se guarda el resultado de una vulnerabilidad encontrada, se hace una petición hacia el adaptador de la base de datos y se realiza la llamada para comunicar a la API del descubrimiento del fallo. Por lo tanto, esta llamada a la base de datos es una llamada como la que se realiza a la hora de realizar un nuevo proceso, de ahí que no se haya querido repetir la llamada y se haya preferido explicarla.

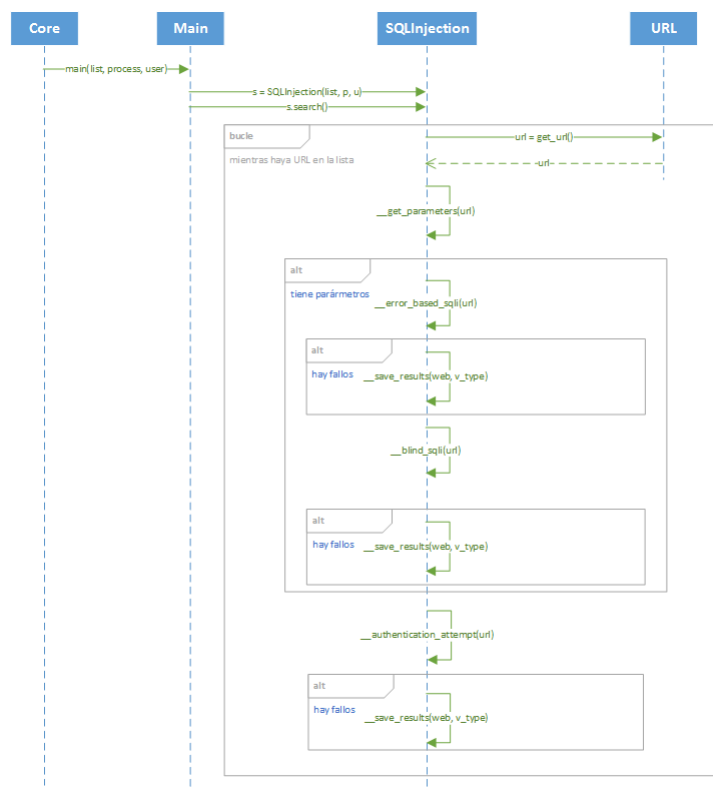


Figura 55: Diagrama de secuencia sobre el módulo de inyección SQL.

C. Anexo III: Manual de instalación

Este manual contendrá los pasos necesarios para realizar la correcta instalación de todas las partes que componen este proyecto.

Como base, se expondrá únicamente la instalación bajo sistemas GNU/Linux basados en Debian. Como adición, se expondrán diversos problemas que puede ocasionar la instalación del *front-end* y la API en una Raspberry Pi 2.

Antes de instalar todos los paquetes y dependencias necesarios, es recomendable actualizar el sistema:

```
sudo apt-get update
sudo apt-get upgrade
```

Ahora ya se puede instalar y configurar todo lo necesario para poder poner en ejecución el proyecto realizado en este TFG.

Antes de comenzar la instalación propia, se explicarán los diferentes directorios de los que se compone el proyecto.

- **back-end.** Ficheros que corresponden con el motor de búsqueda de vulnerabilidades y la base de datos (subdirectorio *db*).
- **front-end.** Ficheros correspondientes al *front-end* de la aplicación, es decir, la página web realizada en AngularJS y Bootstrap.
- **api.** Directorio donde se encuentra todo lo necesario para arrancar la API.
- **wiki.** Carpeta que componen el blog y la wiki.

En las siguientes secciones se irán explicando la instalación de cada una de las partes que componen toda la aplicación, así como los problemas que puede haber.

C.1. Base de datos

La base de datos está realizada en PostgreSQL. La instalación de pgAdmin no es necesaria, pero se recomienda su uso por ser un método más sencillo y cómodo que abrir la terminal de comandos propia de la base de datos.

```
sudo apt-get install postgresql pgadmin
```

Se abrirá pgAdmin y se creará un nuevo servidor (ver Figura 56). Los datos introducidos se deberán introducir en el servidor Python posteriormente.

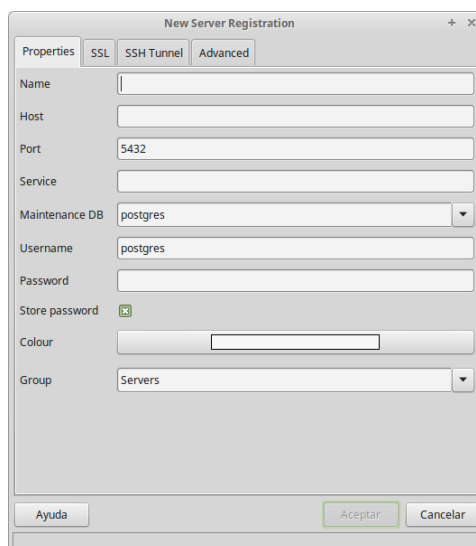


Figura 56: Datos para crear un nuevo servidor de base de datos.

Una vez creado, se deberá crear la base de datos para la aplicación y, una vez creada, restaurar (ver Figura 57) el archivado adjunto con las tablas y procedimientos almacenados.

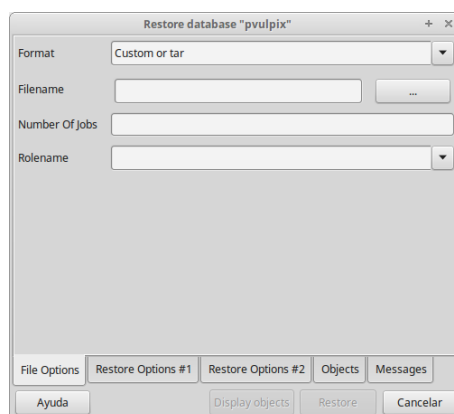


Figura 57: Pantalla para restaurar los datos de la base de datos.

C.2. *Back-end*

Para poder lanzar la aplicación realizada en Python, es necesario la instalación de Python 3.x y PIP. Además, desde los repositorios, se pueden instalar las librerías de BeautifulSoup (para la lectura de páginas web) y Psycopg 2 (adaptador para bases de datos PostgreSQL).

```
sudo apt-get install python3 python3-pip beautifulsoup psycopg2
```

A parte, también es necesaria la instalación de otras librerías usando el programa PIP que descarga librerías desde el gestor propio de paquetes de los servidores de Python.

```
sudo pip install jsonschema
```

Antes de poder iniciar el servidor, será necesario cambiar los datos de conexión a la base de datos que se hayan realizado en el apartado anterior. Para ello, hará falta modificar el fichero *db_adapter.py* (ver Figura 58), perteneciente al directorio *core*, e indicar los datos que se encuentran marcados.

```
def __init__(self):  
    self.conn = psycopg2.connect(database="pvulpix", user="postgres", password="seguridad", host="127.0.0.1", port="5432")
```

Figura 58: Datos a introducir en el adaptador.

Ya es posible arrancar la aplicación. Para ello, se debe abrir una nueva terminal en la que se tiene que ir al directorio donde se encuentra el fichero *server.py* y escribir:

```
cd back-end/search-motor/core/  
python server.py
```

C.3. *Front-end y API*

Para poder implantar la API y el *front-end*, se requiere la instalación de Node.js, npm (gestor de paquetes de Node.js) y MongoDB (la base de datos no relacional de la que hace uso la API).

```
sudo apt-get install nodejs npm mongodb
```

Una vez instalados los anteriores programas, será necesario arrancar la base de datos por un lado, el *front-end* desde otro y, por último, la API también desde otra terminal.

Se arranca el motor de MongoDB usando, para ello, los ficheros de diccionarios mínimos. Se elige esta opción porque MongoDB puede llegar a requerir, nada más arrancar, de 8 GB de almacenamiento y es una buena forma de ahorrar espacio sin ver decrementado su rendimiento (para esta aplicación en concreto).

```
mongod --smallfiles
```

Después, se abrirá otra terminal en el que se desplazará al directorio donde se encuentre el *front-end* y se procederá a arrancarlo. Si las librerías no están previamente instaladas, primero se descargarán e instalarán, para terminar de arrancar la aplicación sin ningún problema.

```
cd front-end  
npm start
```

Una vez arrancado, se mostrará la dirección IP y el puerto en el que se ha arrancado la página web.

Por último, en una nueva terminal se cambia al directorio de la API, se instalarán las librerías y se arrancará la propia API.

```
cd ../api  
npm install  
node App.js
```

Como el *front-end* no requiere del motor de Node.js para funcionar, no hace falta arrancarlo como se arranca la API, pero sí que es necesario la instalación de las diferentes librerías haciendo uso de npm.

C.3.1. Raspberry Pi

En este apartado se explicará cómo instalar la API en el *front-end* en una Raspberry Pi 2. No es obligatorio, pero sí que resulta interesante. Además, es importante la realización de los pasos siguientes para no tener problemas.

C.3.1.1. Raspbian

La instalación en este sistema es igual al mencionado en la sección anterior, puesto que este sistema está basado en Debian.

Tras actualizar el sistema y los paquetes instalados, nos preocuparemos ahora de la instalación de Node.js, que ha de hacerse manualmente, puesto que el paquete que se encuentra en los repositorios de Raspbian no es lo suficiente moderno, por lo que parte de la lógica del *front-end* no es compatible. Por lo tanto, nos descargaremos el paquete en el directorio local del usuario en el que estemos conectados y lo instalaremos:

```
cd ~
wget http://node-arm.herokuapp.com/node_latest_armhf.deb
sudo dpkg -i node_latest_armhf.deb
```

Una vez instalado Node.js, verificamos que se haya instalado correctamente:

```
node -v
```

Una vez realizada esta acción, el resto se instala y ejecuta como si de otro sistema basado en Debian se tratara, visto en la sección anterior.

Nota: es posible que, aún instalando la versión actual de Node.js, ésta no funcione correctamente, por lo que se recomienda, de seguir deseando hacer uso de la Raspberry Pi, cambiar Raspbian por Arch Linux ARM. Otra opción sería descargar el código y tener que compilarlo, pero tampoco está exento de problemas.

C.3.1.2. Arch Linux ARM

La instalación difiere de Raspbian, puesto que esta distribución no está basada en Debian. Los paquetes a instalar son los mismos, pero se instalan usando la herramienta llamada “pacman”.

Actualizamos el sistema e instalamos Node.js y npm, más sus correspondientes dependencias:

```
sudo pacman -Syu
sudo pacman -S nodejs npm
```

Sin embargo, al igual que ocurre en Raspbian, en Arch Linux ARM (ALA) también existen problemas. En este caso, es necesario instalar MongoDB

manualmente, puesto que la versión que está en los repositorios de ALA tiene problemas y no funciona correctamente.

Además, como se instala una versión antigua, es necesario la instalación de una dependencia de la que hace uso MongoDB en unas versiones anteriores de la que instala el propio sistema. Versiones posteriores no son compatibles con la versión de MongoDB a instalar.

Se adjuntan los instaladores (para Raspberry Pi 2) junto a este manual, al ser difíciles de encontrar en Internet.

Contando que los instaladores se han colocado en la carpeta principal del usuario activo, el código a ejecutar es:

```
cd ~
sudo pacman -U boost-libs-1.58.0-2-armv7h.pkg.tar.xz
sudo pacman -U mongodb-3.0.4-1-armv7h.pkg.tar.xz
```

De esa manera, para arrancar la base de datos, API y *front-end*, se deben seguir los mismos pasos utilizados en el apartado general.

Nota: se recomienda, hasta que el paquete del repositorio de MongoDB funcione correctamente, mantener las versiones instaladas manualmente de estos dos paquetes a la hora de actualizar el sistema.

C.4. Blog y wiki

Para la instalación del blog y la wiki, será necesaria la instalación de un servidor web que pueda interpretar PHP (p. ej. Apache o *nginx*).

Como esta herramienta es un complemento al proyecto en general, no se explicará de manera detallada su instalación.

Se deberá modificar el fichero *api-data.php* que se encuentra en la raíz del directorio de la herramienta para añadir la dirección y el puerto en la que se encuentra la API. Es necesaria la API para el funcionamiento del blog/wiki.

Una vez puesto el directorio de la herramienta dentro del directorio del servidor con PHP, se accederá a la dirección de la misma usando el navegador y se seguirán los pasos que muestra la propia herramienta.

Una vez instalada, la herramienta será completamente funcional y se podrá empezar a usar.

D. Anexo IV: Manual de usuario

En este anexo se explicará cómo desenvolverse por la interfaz gráfica de la aplicación usando diferentes capturas para tratar de hacer más visual este manual.

En la Figura 59 se muestra la pantalla principal de la página web en que la se puede ver el logotipo y el buscador.

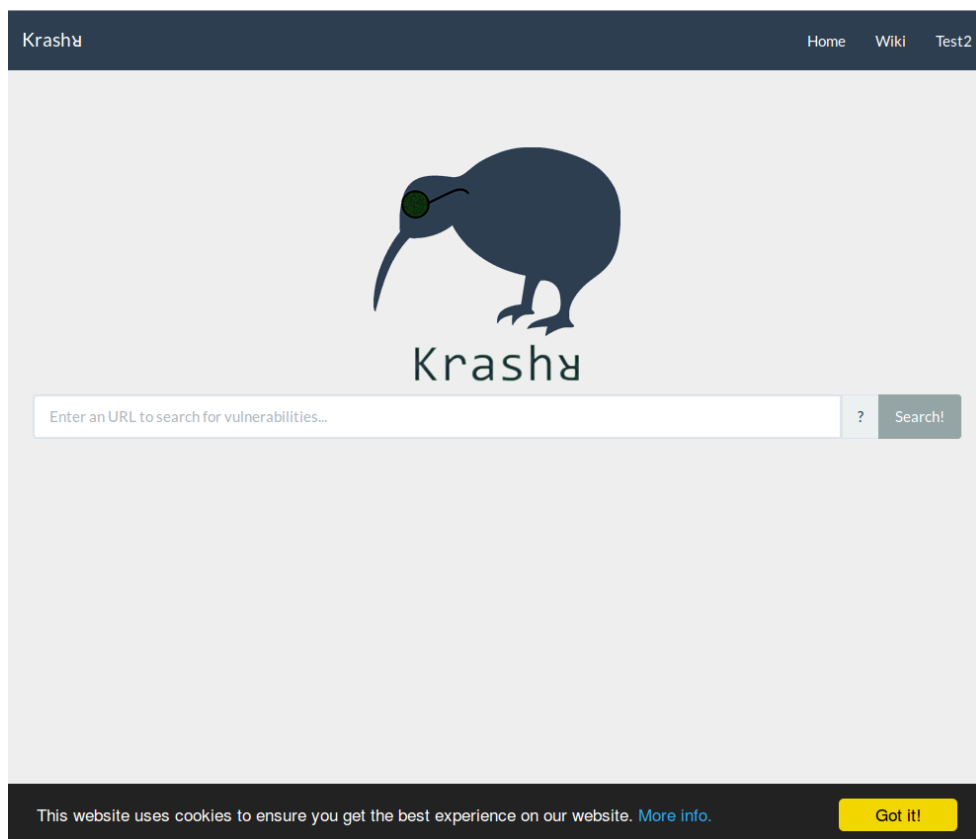


Figura 59: Página principal de Krashr.

D.1. Registro o conexión

La página web requiere que el usuario se registre y conecte a la misma si éste pretende realizar una búsqueda de vulnerabilidades a una página web.

Por lo tanto, nada más acceder a la página web, se tendrá que hacer clic en el vínculo extremo de la derecha de la barra superior de la página (ver

Figura 60).

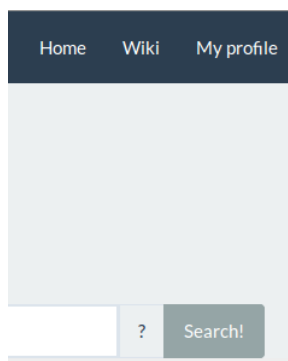


Figura 60: Extremo superior derecho del *front-end*.

La página está preparada y mostrará diferentes avisos (ver Figura 61) de no realizar alguna acción correctamente. Todos los datos son obligatorios y tienen una longitud mínima para poder continuar con el registro o conexión.

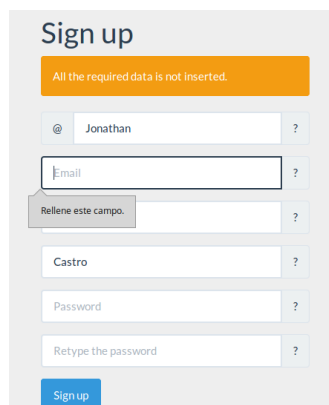


Figura 61: Formulario de registro con aviso.

La página web conecta al usuario nada más registrarse automáticamente. Sólo deberá conectarse cuando se eliminen los datos del navegador o cuando se usa un navegador/ordenador diferente al que se realizó el registro.

D.2. Actualizar perfil

Para acceder al perfil (ver Figura 62), se debe pulsar en el vínculo extremo de la derecha del menú principal o en el primer enlace en el submenú (siempre

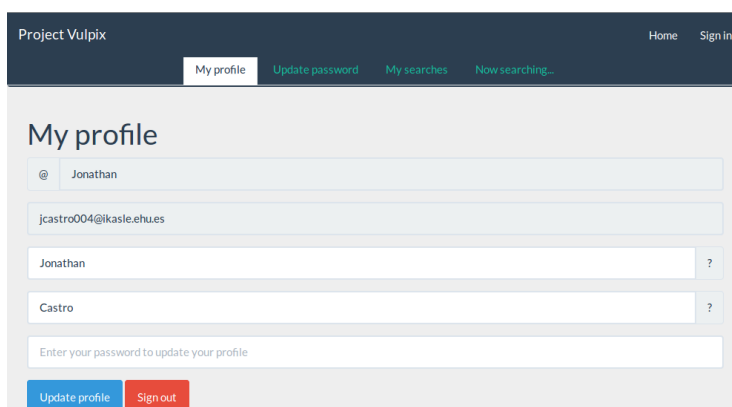


Figura 62: Pantalla de perfil de usuario.

que esté activo). En el perfil podrá modificar el usuario sus datos si así lo desea.

El usuario tendrá que volver a escribir su contraseña actual (ver Figura 63) para poder actualizar sus datos. Este campo es obligatorio para tratar de asegurar que es el propio usuario el que actualiza sus datos.

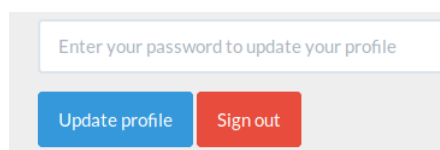
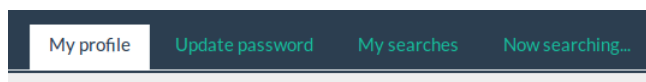


Figura 63: Campo para insertar la contraseña al actualizar el perfil.

Al igual que en los formularios de registro y conexión, se mostrarán avisos si existe alguna complicación con el proceso de actualización.

D.3. Actualizar contraseña

En el submenú (ver Figura 64), la segunda opción lleva a una página en la que el usuario podrá actualizar su contraseña si así lo desea.

Figura 64: Submenú del *front-end*.

Para ello, el usuario tendrá que introducir su antigua contraseña para

tratar de asegurar que es el propio usuario el que está realizando la acción, además de la nueva contraseña por duplicado, para ayudar al usuario a escribir la contraseña correcta sin equivocaciones.

D.4. Realizar una nueva búsqueda

Las búsquedas se realizan en la página principal (ver Figura X) de la aplicación web. En la parte central se puede observar un campo en el que se introducen las direcciones de las diferentes páginas web de las que se pretender buscar las diferentes vulnerabilidades.

Como bien se ha comentado, es necesario que el usuario esté previamente conectado a la aplicación (ver apartado *Registro o conexión*). De no estarlo, se mostrará un aviso indicando del error (ver Figura 65).

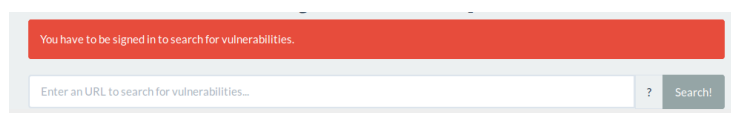


Figura 65: Pantalla de búsqueda con aviso de conexión.

Una vez introducida la dirección de la página web deseada, que debe seguir el patrón indicado en la ayuda del buscador (ver Figura 66), la búsqueda habrá comenzado y el usuario será redireccionado a la pantalla del estado de la búsqueda.

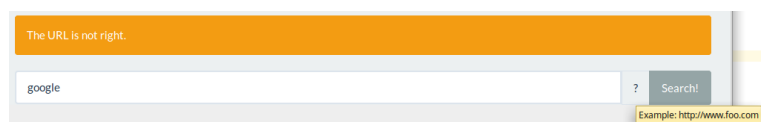


Figura 66: Pantalla de búsqueda con error.

D.5. Ver estado de la búsqueda

En esta sección el usuario podrá ver en qué estado está la última búsqueda que ha realizado (ver Figura 67), de haber alguna activa. En caso contrario, se mostrará un aviso (ver Figura 68).

Cada búsqueda se realiza independientemente de dónde se encuentre el usuario dentro de la página web; no importa que éste se salga de esta pantalla, la búsqueda continuará hasta que finalice o falle.

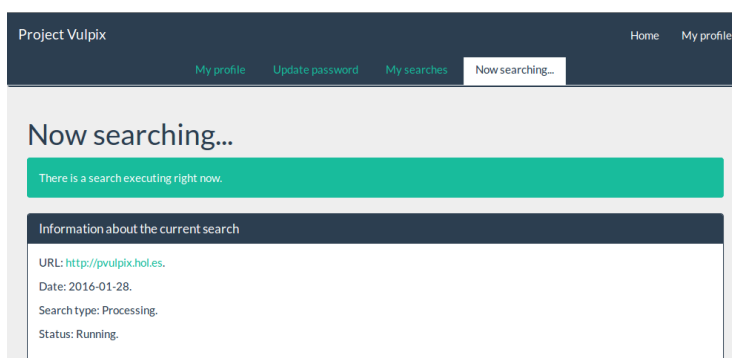


Figura 67: Pantalla de información sobre búsqueda activa.



Figura 68: Pantalla de información sin búsqueda activa.

Nota: esta pantalla no se recarga sola, de querer ver el estado actual, se deberá recargarla manualmente.

D.6. Ver resultados

Los resultados se mostrarán agrupados por búsqueda realizada (ver Figura 69).

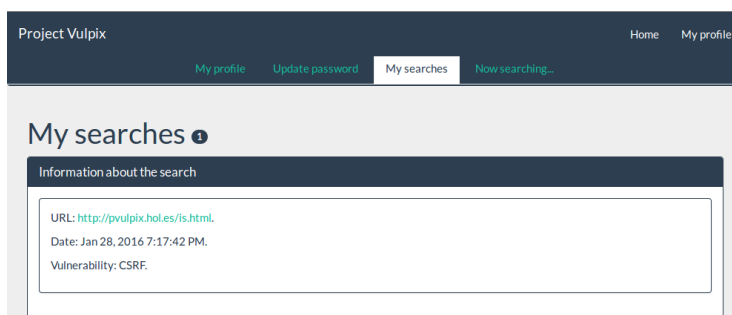


Figura 69: Pantalla de resultados.

Cada resultado mostrará la página en la que se ha encontrado la vulnerabilidad, la vulnerabilidad con un enlace a la wiki y la hora en la que se ha encontrado.