Universitat de Girona
Escola Politècnica Superior

Projecte fi de grau

Estudi: Grau en Enginyeria Informàtica

Títol: La seguridad como punto de partida del desarrollo web

Document: Memòria

Alumne: Sergi Bergillos Pedraza

Tutor: Antonio Bueno Delgado
Departament: Arquitectura i tecnologia de computadors
Àrea: Arquitectura i tecnologia de computadors

Convocatòria (mes/any): Setembre 2021

Universidad de Girona
Escuela Politécnica Superior
Grado en Ingeniería Informática

La seguridad como punto de partida del desarrollo web

Proyecto de final de grado

Autor
Sergi Bergillos Pedraza

Tutor
Antonio Bueno Delgado

Septiembre de 2021

CC BY 4.0

Atribución 4.0 Internacional

Agradecimientos

«Quiero agradecer primero a mi tutor, Antonio Bueno Delgado, por su interés y su disponibilidad durante los meses de verano. También al grupo BCDS por la comprensión y la flexibilidad de horarios que me ha permitido dedicarme estas últimas semanas por completo al proyecto. Por último a mi familia porque sin su apoyo y su paciencia no habría sido posible nada de esto.»

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La seguridad y la privacidad de los datos personales es un derecho del usuario y una responsabilidad de las empresas tal y como recogen numerosas leyes o reglamentos a nivel europeo o español como el *Reglamento general de protección de datos*[1]; la *Ley orgánica de Protección de Datos Personales y garantía de los derechos digitales*[2]; y la *Ley de servicios de la sociedad de la información y de comercio electrónico*[3].

Sin embargo, el conocimiento que los estudiantes del grado de ingeniería informática de la Universidad de Gerona (UdG) adquieren durante los cuatro años de carrera es insuficiente para preparar a los recién graduados para los nuevos desafíos del mundo laboral en ciberseguridad.

En concreto, las asignaturas relacionadas con la privacidad y la seguridad de los datos y los sistemas informáticos en el grado de ingeniería informática de la UdG son: «Legislación y ética profesional», «Criptografía» y «Seguridad y protección de datos». Si bien, solo la primera es obligatoria.

A pesar de que este conocimiento puede ampliarse de forma autónoma en internet, la gran cantidad de información disponible al respecto, aunque esta no siempre es reciente, fiable o en castellano, puede resultar abrumadora.

El objetivo de este proyecto es de carácter divulgativo y didáctico: documentar los diferentes tipos de ciberataques contra sitios web, cómo se producen y cómo protegerse; y proporcionar una serie de consejos y buenas prácticas para diseñar e implementar un sitio web seguro.

No obstante, este propósito no es simplemente un ejercicio teórico por lo que se implementará una plantilla de un sitio web con las funcionalidades básicas en los *stacks* MEVN (MongoDB, Express.js, Vue.js y Node.js) y LAMP (Linux, Apache, MySQL y PHP). Una vez acabado el proyecto, ambas plantillas podrán encontrarse en código abierto en GitHub y podrán usarse como un punto de partida de una sitio web seguro.

Además, al no tratarse de un proyecto de desarrollo de un producto o de un sistema informático, se han suprimido los capítulos de «Requerimientos del sistema» e «Implantación y resultados» y añadidos los capítulos «Recopilación de buenas prácticas para un desarrollo web seguro» y «Recopilación de ciberataques: cómo se producen y cómo protegerse» a la presente memoria del proyecto de final de grado. También se ha sustituido el capítulo de «Manual de usuario y/o instalación» por «Guía de contribución».

Por último, dado el carácter divulgativo y didáctico del proyecto, la documentación se ha hecho en castellano para llegar al mayor número de personas posibles.

Capítulo 2

Estudio de viabilidad

2.1. Viabilidad TELOS del proyecto

El estudio de viabilidad se ha realizado utilizando el sistema TELOS. TELOS es un acrónimo que define cinco factores esenciales para determinar la viabilidad de un proyecto:

Technical
¿Es el proyecto técnicamente posible?

Economical
¿Es el proyecto económicamente rentable?

Legal
¿Es el proyecto legal?

Operational
¿Cómo afecta el proyecto a las operaciones actuales?

Scheduling
¿Se puede terminar el proyecto a tiempo?

Viabilidad técnica

Para el correcto desarrollo del proyecto son necesarios los siguientes elementos:

- Ordenador personal con sistema operativo Linux y conexión a internet para instalar o utilizar el *software* adicional necesario.
- Servidor en la nube con sistema operativo Linux para probar las plantillas en un entorno de producción.
- Las tecnologías y *frameworks* que corresponden a los dos *stacks* escogidos: MongoDB, Express.js, Vue.js, Node.js, Ubuntu (Linux), Apache, MySQL y Laravel (PHP).
- Editor de código: Visual Studio Code.
- Repositorio de código en línea: GitHub.
- Editor de LaTeX: Overleaf.

- Herramienta de gestión de proyectos: Trello.

El proyecto es técnicamente viable porque ya se posee el *hardware* o puede ser alquilado a un bajo precio; y el *software*, las tecnologías y los *frameworks* escogidos son de código abierto o gratuitos (GitHub, Overleaf y Trello).

Viabilidad económica

El coste estimado del proyecto es 6.750€ o 375 horas de dedicación al proyecto.

La fórmula que se ha utilizado para calcularlo es la siguiente:

$$15 \text{ créditos ECTS } * 25^{1} \text{horas/crédito ECTS } = 375 \text{ horas } * 18^{2} €/\text{horas} = 6.750 €$$

Al tratarse de un proyecto de carácter divulgativo y didáctico, no se esperan beneficios monetarios.

Sin embargo, no se debe subestimar la importancia económica de los conocimientos adquiridos durante la realización del proyecto ya que, por ejemplo, incumplir los preceptos legales del *Reglamento General de Protección de Datos*[1] puede suponer multas de hasta veinte millones de euros o 4 % de la facturación global de la compañía.

Además, al implementarse diferentes plantillas web con las funcionalidades básicas, se acelerará la implantación de sitios web futuros lo que supondrá un ahorro significativo en tiempo y en dinero.

Por estos motivos, el proyecto se considera económicamente viable.

Viabilidad legal

Uno de los objetivos del proyecto es cumplir al pie de la letra la legislación aplicable, ya mencionada en el Capítulo 1:

- El *Reglamento General de Protección de Datos* (RGPD)[1].
- La *Ley orgánica de Protección de Datos Personales y garantía de los derechos digitales* (LOPDPGDD)[2].
- La *Ley de servicios de la sociedad de la información y de comercio electrónico* (LSSI)[3]

Viabilidad operacional

El proyecto actual puede compaginarse perfectamente con el trabajo a tiempo parcial del autor.

Viabilidad de la programación

- Fecha de inicio del proyecto: 1 de junio de 2021.
- Fecha de fin del proyecto: 1 de septiembre de 2021.

[1] De acuerdo al Ministerio de Educación y Formación Profesional el crédito ECTS equivale a 25 horas de trabajo del estudiante[4].

[2] Redondeo del salario promedio por hora en *Salario para full Stack en España - Salario Medio*[5].

Para cumplir la programación del proyecto debe dedicarse una media de cuatro horas diarias. Este es un número asumible por lo que la programación del proyecto es viable.

2.2. Acta de constitución del proyecto

Una vez estudiado la viabilidad del proyecto y considerado factible, se ha realizado el acta de constitución del proyecto o *project charter* en inglés.

Nombre del proyecto

La seguridad como punto de partida del desarrollo web

Objetivos del proyecto

El objetivo del proyecto es hacer una recopilación de consejos, buenas prácticas y medidas de protección contra diferentes ciberataques para un desarrollo web seguro y ofrecer una plantilla de ejemplo en código abierto con las funcionalidades básicas implementadas.

Riesgos del proyecto

- Conocimiento limitado del autor en diseño de interfaces web.
- Conocimiento limitado del autor en las tecnologías o *frameworks* MongoDB y Vue.js del *stack* MEVN.
- Conocimiento limitado del autor en las tecnologías o *frameworks* Apache, MySQL y Laravel (PHP) del *stack* LAMP.

Beneficios del proyecto

- Concienciar sobre la importancia de la privacidad y la seguridad de los datos en el desarrollo web.
- Ampliar el conocimiento de los programadores web en ciberseguridad.
- Reducir el tiempo necesario para la implantación de un sitio web.

Entregas del proyecto

- Recopilación de consejos y buenas prácticas para un desarrollo web seguro y legal.
- Recopilación de ciberataques: cómo se producen y cómo protegerse.
- Plantilla en código abierto con el *stack* MEVN.
- Plantilla en código abierto con el *stack* LAMP.

Límite del presupuesto del proyecto

No superar los 8.100€ o las 450 horas de dedicación al proyecto.

La fórmula que se ha utilizado para calcular el límite del presupuesto es:

$$15 \text{ créditos ECTS} * 30^{3}\text{horas/crédito ECTS} = 450 \text{ horas} * 18 \text{ €/horas} = 8.100 \text{ €}$$

Hitos del proyecto

- Diseño de las interfaces web y del protocolo REST: 15 de julio de 2021.
- Plantillas MEVN y LAMP: 15 de agosto de 2021.
- Memoria del proyecto con el recopilatorio de buenas prácticas y ciberataques: 1 de septiembre de 2021.
- Defensa del proyecto: 10 de septiembre de 2021.

[3]Incremento porcentual del 20 % de las horas por crédito ECTS.

Capítulo 3

Metodología

La metodología de desarrollo que se ha escogido para gestionar el proyecto es el método Kanban.

Los dos componentes claves del sistema, como puede observarse en la figura 3.1, son el tablero y las tarjetas. El tablero está compuesto por al menos tres columnas que definen los diferentes procesos del proyecto: «Por hacer», «En progreso» y «Hecho»; y la tarjeta contiene toda la información significativa de la tarea: nombre, descripción, fecha de creación o prioridad, por ejemplo.

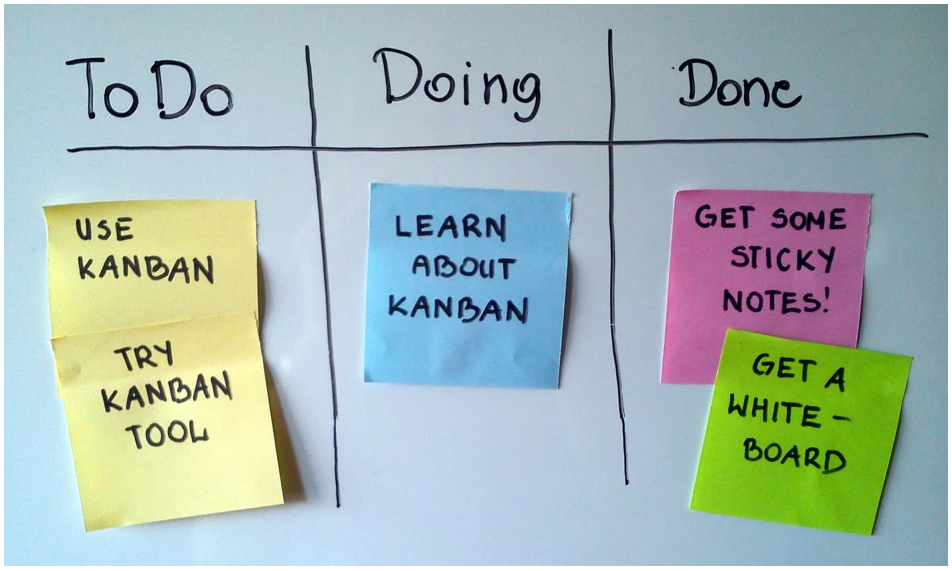

Figura 3.1: Tablero Kanban básico

Los cuatro principios básicos del método Kanban, definidos por David J. Anderson en su libro *Agile Management for Software Engineering*[6], son:

- Empiece con lo que hace ahora.
- Comprométase a buscar e implementar cambios incrementales y evolutivos.
- Respete el proceso, los roles, las responsabilidades y los cargos actuales.
- Anime actos de liderazgo en todos los niveles.

Anderson también define seis prácticas para una implementación con éxito del método:

- Visualiza el flujo de trabajo.
- Limita el trabajo en proceso.
- Gestiona el flujo.
- Fomenta la visibilidad del sistema.
- Transfiera el conocimiento.
- Mejora colaborando.

El sistema Kanban es un método ágil, visual y fácil de implantar que permite gestionar con fluidez el trabajo y conocer en detalle el estado del proyecto. Por estos motivos, se ha escogido Kanban como metodología para desarrollar el proyecto.

Concretamente, se ha utilizado la aplicación web Trello para implementar el tablero Kanban con el *power-up* Epic Card. La descripción de las columnas o listas del tablero[7], cuya visualización es pública, y su significado se encuentra en la tabla 3.1.

Además, dada la filosofía Kanban en que solo se puede empezar una tarea nueva después de haber finalizado la última, se ha restringido el número total de tareas en las columnas de *Doing*, *Code Review* y *Testing* a uno porque solo hay un miembro en el equipo de trabajo.

Columna	**Significado**
Epics	Lista de las tareas «épicas». Las tareas «épicas» corresponden a los diferentes hitos del proyecto.
New	Lista de las plantillas de tareas y las tareas a realizar que todavía no se han descrito y priorizado.
Backlog	Lista de las tareas a realizar, pero que no se pueden empezar porque dependen de otras que aún no se han finalizado.
To Do	Lista de las tareas a realizar que pueden ser empezadas en cualquier momento.
Bugs & Hotfixs	Lista de las tareas relacionadas con errores encontrados en producción a solucionar lo más pronto posible.
On Hold	Lista de las tareas empezadas, pero que han tenido que ser interrumpidas por la aparición de un *bug* urgente o un *hotfix*.
Doing	Lista de las tareas en progreso actualmente.
Code Review	Lista de las tareas cuyo formato del código tiene que ser aprobado. Al tratarse de un proyecto de código abierto, el código escrito tiene que seguir un patrón definido para mejor su legibilidad.
Testing	Lista de las tareas que tienen que verificarse su corrección.
Done	Lista de las tareas finalizadas.

Cuadro 3.1: Descripción del tablero Kanban implementado en Trello

Capítulo 4

Planificación

4.1. Descripción SMART de los objetivos

El primer paso que se ha realizado durante la planificación es la descripción SMART de los objetivos definidos en el «Acta de constitución del proyecto» en el Capítulo 2. SMART es un acrónimo que corresponde a:

Specific
El objetivo ha de especificarse tanto como sea posible.

Measurable
El objetivo ha de ser cuantificable o permitir un progreso mensurable.

Attainable
El objetivo ha de ser realista.

Relevant
El objetivo ha de merecer la pena.

Time-bound
El objetivo ha de tener una fecha límite o un final definido.

4.1.1. Recopilación de buenas prácticas para un desarrollo web seguro

Especifico

Se estudia cómo diseñar e implementar las siguientes funcionalidades básicas de una forma segura, ordenadas por prioridad:

1. Protocolo HTTPS.
2. Registro.
3. Tratamiento de los errores.
4. Validación de las entradas.
5. Subir archivos.

6. Almacenamiento de datos sensibles.
7. Creación de las cuentas de usuario.
8. Autenticación de los usuarios.
9. Gestión de las *cookies*.
10. Gestión de la sesión.
11. Recuperación de la contraseña de un usuario.
12. Control de accesos de los usuarios.
13. Serialización y deserialización de los objetos.
14. Seguridad de la base de datos.
15. Monitoreo.

Mensurable

El progreso del objetivo puede ser mesurado con el número de funcionalidades estudiadas.

Asumible

El objetivo es asumible.

Pertinente

La seguridad y la privacidad de los datos en la web empieza con un diseño correcto.

Limitado en el tiempo

El objetivo ha de estar finalizado el 1 de septiembre de 2021.

4.1.2. Recopilación de ciberataques: cómo se producen y cómo protegerse

Especifico

En concreto, los ataques, también ordenados por importancia, que se estudian son:

1. Denial of Service (DoS).
2. Ataque de credenciales.
3. Cross-Site Request Forgery (XSRF o CSRF).
4. Cross-Site Scripting (XSS).
5. Code Injection.
6. Client-State Manipulation.
7. Cross Site Script Inclusion (XSSI).

8. *Overflow* de enteros y del *buffer*.
9. Path Traversal.

Mensurable

El progreso del objetivo puede ser mesurado con el número de ataques analizados.

Asumible

El objetivo es asumible.

Pertinente

El primer paso para protegerse frente a los ciberataques es conocer cuáles son y cómo se producen.

Limitado en el tiempo

El objetivo ha de estar finalizado el 1 de septiembre de 2021.

4.1.3. Diseño e implementación de las plantillas de un sitio web seguro

Especifico

Diseño e implementación de la plantilla de un sitio web utilizando los *stacks* MEVN (MongoDB, Express.js, Vue.js y Node.js) y LAMP (Linux, Apache, MySQL y PHP) con las funcionalidades básicas siguientes:

- Creación de las cuentas de usuario.
- Autenticación y gestión de la sesión.
- Subir imágenes, ficheros y textos.
- Política de privacidad y gestión de las *cookies*.
- Protección frente los diferentes tipos de ataques estudiados.

Mensurable

El progreso del objetivo puede ser mesurado con el número de funcionalidades implementadas.

Asumible

El objetivo es asumible.

Pertinente

Puesta en práctica de los conocimientos adquiridos durante la recopilación de buenas prácticas y ciberataques en un sitio web real.

Limitado en el tiempo

El objetivo ha de estar finalizado el 21 de agosto de 2021.

4.2. Estimación de las tareas

Una vez especificados los objetivos del proyecto con el sistema SMART, se han desglosado en tareas más pequeñas que permitan la evaluación del trabajo para finalizarlas. Después, se ha analizado las dependencias entre ellas para priorizar aquellas cuyo número de tareas dependientes es mayor, aunque la decisión final del orden en que se realizan siempre es del autor.

La lista inicial con su estimación en horas se encuentra en la tabla 4.1 y cada una de estas es una tarjeta en el tablero Kanban. Si bien, la estimación total de horas es superior a la dedicación esperada para un proyecto de final de grado o la considerada al calcular el presupuesto en la Sección 2.1: «Viabilidad económica». Esto es debido al gran número de características, funcionalidades y ciberataques que deben tenerse en cuenta para diseñar e implementar una aplicación web segura.

Sin embargo, no se han podido realizar todas las tareas planificadas ya que además, durante la planificación del proyecto, tampoco se ha tenido en cuenta el tiempo necesario para la documentación y la propia planificación.

Inicio de la tabla 4.1

	Tarea	**Estimación**
Investigación buenas prácticas	Protocolo HTTPS	4 h
	Registro	8 h
	Tratamiento de los errores	8 h
	Almacenamiento de datos sensibles	8 h
	Validación de las entradas	8 h
	Subir archivos	8 h
	Serialización y deserialización	8 h
	Creación de las cuenta de usuario	12 h
	Autenticación de los usuarios	12 h
	Gestión de las *cookies*	8 h
	Gestión de la sesión	8 h
	Recuperación de la contraseña	8 h
	Control de accesos de los usuarios	8 h
	Seguridad de las bases de datos	8 h
	Monitoreo	12 h
Investigación ciberataques	Ataques de credenciales	8 h
	Ataque XSS	12 h
	Code Injection	12 h
	Client-State Manipulation	8 h
	Ataque XSRF	12 h
	Ataque XSSI	12 h
	Ataque DoS	8 h

Continuación de la tabla 4.1

	Tarea	**Estimación**
	Overflow de enteros y del *buffer*	8 h
	Path Traversal	8 h
Diseño	API	10 h
	Wireframe	10 h
	Mockup	20 h
Implementación MEVN	Creación del proyecto	2 h
	Registro y monitoreo	4 h
	Tratamiento de los errores	4 h
	Implementación HTTPS	4 h
	Página principal	4 h
	Creación de la cuenta	8 h
	Recuperación de la contraseña	4 h
	Autenticación de los usuarios	8 h
	Gestión de las *cookies*	8 h
	Gestión de la sesión	8 h
	Gestión de la cuenta: modificación y eliminación	8 h
	Subir imágenes y documentos	8 h
	Entrar textos	8 h
	Control de accesos de los usuarios	8 h
	Proteger frente ataques de credenciales	4 h
	Proteger frente ataques DoS	4 h
	Proteger frente ataques XSS	4 h
	Proteger frente ataques XSRF	4 h
	Proteger frente ataques XSSI	4 h
	Proteger frente ataques de Code Injection	4 h
	Proteger frente ataques de Client-State Manipulation	4 h
	Proteger frente ataques de *overflow*	4 h
	Proteger frente ataques de Path Traversal	4 h
Implementación LAMP	Creación del proyecto	2 h
	Registro y monitoreo	4 h
	Tratamiento de los errores	4 h
	Implementación HTTPS	4 h
	Página principal	4 h
	Creación de la cuenta	8 h
	Recuperación de la contraseña	4 h
	Autenticación de los usuarios	8 h
	Gestión de las *cookies*	8 h
	Gestión de la sesión	8 h
	Gestión de la cuenta: modificación y eliminación	8 h
	Subir imágenes y documentos	8 h
	Entrar textos	8 h
	Control de accesos de los usuarios	8 h

Continuación de la tabla 4.1

	Tarea	**Estimación**
	Proteger frente ataques de credenciales	4 h
	Proteger frente ataques DoS	4 h
	Proteger frente ataques XSS	4 h
	Proteger frente ataques XSRF	4 h
	Proteger frente ataques XSSI	4 h
	Proteger frente ataques de Code Injection	4 h
	Proteger frente ataques de Client-State Manipulation	4 h
	Proteger frente ataques de *overflow*	4 h
	Proteger frente ataques de Path Traversal	4 h
		Total: 488 h

Cuadro 4.1: Lista de tareas

4.3. Plan de trabajo

En resumen, el plan de trabajo seguido para el desarrollo del proyecto puede verse en la figura 4.1.

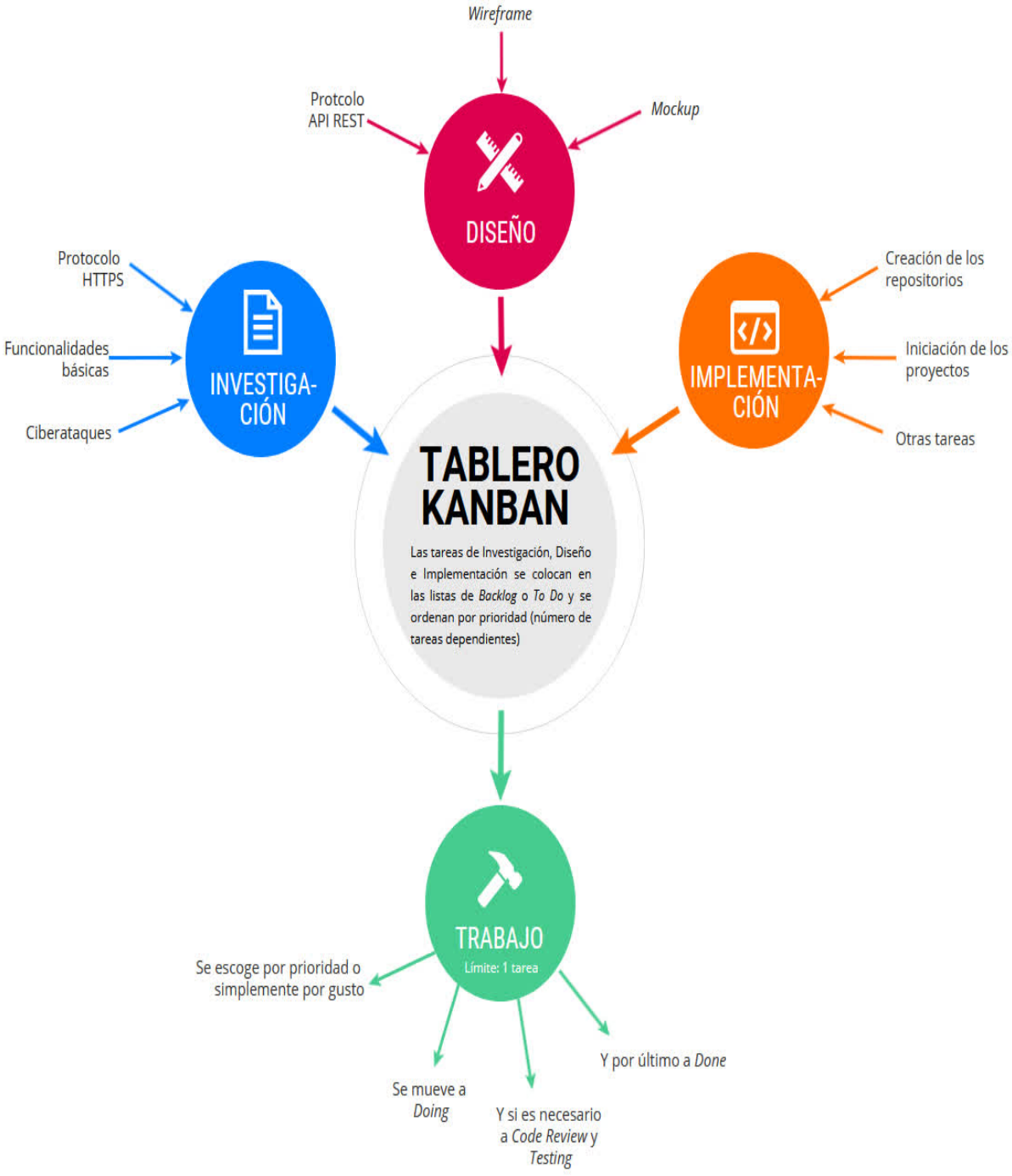

Figura 4.1: Diagrama del plan de trabajo

Capítulo 5

Marco de trabajo y conceptos previos

5.1. Marco de trabajo

La seguridad informática es una preocupación creciente entre los involucrados en el desarrollo de una aplicación web: desde los propios desarrolladores o los directivos de las empresas a los usuarios finales del producto. El 68 % de los líderes empresariales sienten que sus riesgos de ciberseguridad están aumentando[8].

Sin embargo, la mentalidad habitual en el sector es considerarla un plus, no un esencial. Si bien, esta es una concepción peligrosa, que afortunadamente está cambiando: la seguridad informática y la privacidad de los datos han de ser los pilares fundamentales del diseño y de la implementación del *software*. Un arquitecto no levantaría un rascacielos sobre suelos inestables; tampoco debería de hacerlo un programador sobre una aplicación insegura.

De acuerdo al *Informe sobre el coste de una brecha de datos 2020* [9] de IBM Security, el coste medio de una brecha de datos es de 3,86 millones de dólares y el plazo medio de identificación y contención es de 280 días. Otro estudio, por la Universidad de Maryland, señala que los piratas informáticos atacan 2.244 veces al día o cada 39 segundos[10]. Unas cifras nada desdeñables.

Además, los expertos en ciberseguridad están en demanda en todo el mundo: pronto habrá más de 3,5 millones de trabajos sin ocupar relacionados con la seguridad informática[11], aunque solo 1 de cada 4 de los candidatos están cualificados[12]. Asimismo, también son los mejor pagados con un salario medio de 140.000$[13].

5.2. Conceptos previos

A continuación, se describen brevemente los conceptos necesarios, ordenados alfabéticamente, para entender el proyecto.

5.2.1. AEPD

La Agencia Española de Protección de Datos (AEPD) es la autoridad pública independiente encargada de velar por la privacidad y la protección de datos de la ciudadanía. Su objetivo es fomentar que las personas conozcan sus derechos y que los desarrolladores tengan a su disposición un instrumento ágil que les facilite el cumplimiento de la normativa.

5.2.2. API

Una Interfaz de Programación de Aplicaciones (API) es un conjunto de definiciones y protocolos que se utiliza para desarrollar e integrar el software de las aplicaciones.

Las API permiten que sus productos y servicios se comuniquen con otros, sin necesidad de saber cómo están implementados. Esto simplifica el desarrollo de las aplicaciones y permite ahorrar tiempo y dinero[14].

API RESTful

Las API web que funcionan con las limitaciones REST (Transferencia de Estado Representacional) se llaman API RESTful que han de cumplir las seis limitaciones siguientes[14]:

- Arquitectura cliente-servidor: la comunicación se realiza con el protocolo HTTP.
- Sin estado: la información sobre el estado de la sesión se almacena en el cliente.
- Capacidad de caché: el almacenamiento en caché puede eliminar la necesidad de algunas interacciones cliente-servidor.
- Sistema en capas: las interacciones cliente-servidor pueden estar mediadas por capas adicionales, como equilibrio de carga, cachés compartidos o seguridad.
- Código de demanda (opcional): los servidores pueden extender las funciones de un cliente transfiriendo código ejectuable.
- Interfaz uniforme: esta limitación es fundamental e incluye cuatro aspectos.
    - Identificación de recursos en las solicitudes: los recursos se identifican en las solicitudes y se separan de las representaciones que se devuelven al cliente.
    - Administración de recursos mediante representaciones: los clientes reciben archivos que representan los recursos. Estas representaciones deben tener la información suficiente como para poder ser modificadas o eliminadas.
    - Mensajes autodescriptivos: cada mensaje que se devuelve al cliente contiene la información suficiente para describir cómo debe procesar la información.
    - Hipermedios es el motor del estado de la aplicación: después de acceder a un recurso, el cliente debe ser capaz de descubrir mediante hipervínculos todas las otras acciones que están disponibles en el momento.

5.2.3. Código abierto

El *software* de código abierto, o *open source software* (OSS) en inglés, es el *software* que cumple con las diez directrices de la Open Source Definition (OSD)[15]:

Redistribución libre
: La licencia no impedirá que ninguna de las partes venda o regale el *software* como un componente de una distribución de *software* agregada que contenga programas de varias fuentes diferentes.

Código fuente
: El programa debe incluir el código fuente y debe permitir la distribución tanto en código fuente como en forma compilada. Cuando alguna forma de un producto no se distribuya con el código fuente, debe haber un medio bien publicitado para obtener el código fuente.

Trabajos derivados
: La licencia debe permitir modificaciones y trabajos derivados, y debe permitir que se distribuyan bajo los mismos términos que la licencia del *software* original.

Integridad del código fuente del autor
: La licencia puede restringir la distribución del código fuente en forma modificada solo si la licencia permite la distribución de «archivos de parche» con el código fuente con el fin de modificar el programa en el momento de la compilación. La licencia debe permitir explícitamente la distribución de *software* creado a partir del código fuente modificado. La licencia puede requerir que los trabajos derivados tengan un nombre o número de versión diferente al del *software* original.

No discriminar a ninguna persona o grupo
: La licencia no debe discriminar a ninguna persona o grupo de personas.

No discriminar a ningún sector de actividad
: La licencia no debe restringir a nadie el uso del programa en un sector específico de actividad.

Distribución de la licencia
: Los derechos adjuntos al programa deben aplicarse a todos aquellos a quienes se redistribuye el programa sin la necesidad de que esas partes ejecuten una licencia adicional.

La licencia no debe ser específica de un producto
: Los derechos adjuntos al programa no deben depender de que el programa sea parte de una distribución de *software* en particular. Si el programa se extrae de esa distribución y se usa o distribuye dentro de los términos de la licencia del programa, todas las partes a quienes se redistribuye el programa deben tener los mismos derechos que los que se otorgan junto con la distribución original del *software*.

La licencia no debe restringir otro *software*
: La licencia no debe imponer restricciones a otro *software* que se distribuya junto con el *software* con licencia.

La licencia debe ser tecnológicamente neutral
: Ninguna disposición de la licencia puede basarse en ninguna tecnología o estilo de interfaz individual.

5.2.4. *Cookies*

Las *cookies* o galletas informáticas son pequeños fragmentos de texto enviados por los sitios web y almacenados en los navegadores de internet. Sus dos propósito principales son mantener un estado y conocer los hábitos de navegación de los usuarios del sitio web.

Se pueden hacer varias clasificaciones de las *cookies*. Según quien la gestione:

- Las *cookies* propias: el propietario es el servicio solicitado por el usuario.
- Las *cookies* de terceros: el propietario es otra entidad que trata los datos obtenidos con las *cookies*.

Según su finalidad:

- *Cookies* técnicas: permiten al usuario la navegación al sitio web.
- *Cookies* de preferencia o personalización: permiten personalizar la experiencia del usuario.
- *Cookies* de análisis o medición: permiten al propietario de la *cookie* el seguimiento y el análisis del comportamiento del usuario.
- *Cookies* de publicidad comportamental: almacenan información del comportamiento de los usuarios para mostrar publicidad personalizada.

Y según su duración:

- *Cookies* de sesión: recaban y almacenan los datos mientras el usuario accede al sitio web.
- *Cookies* persistentes: los datos siguen almacenados en el navegador durante un periodo definido por el propietario de la *cookie*.

Su uso está regulado por el RPGD[1] y la LSSI[3] ya que las *cookies* pueden utilizarse para rastrear la actividad de los usuarios en Internet y se han convertido en un peligro para la privacidad.

Los interesados en profundizar sobre el uso que los sitios web dan a las *cookies* en la actualidad y el grado de cumplimiento normativo pueden consultar el siguiente trabajo en catalán: *Les cookies i tecnologies similars a les pàgines web: Grau de compliment normatiu*[16].

5.2.5. Desarrollo web

El desarrollo web[17] se puede definir como las tareas asociadas con el desarrollo de sitios web para alojarlos a través de una intranet o Internet. El proceso de desarrollo web incluye diseño web, desarrollo de contenido web, *scripting* del lado del cliente o del lado del servidor y configuración de seguridad de la red, entre otras tareas. En un sentido más amplio, el desarrollo web abarca todas las acciones, actualizaciones y operaciones necesarias para crear, mantener y administrar un sitio web para garantizar que su rendimiento, experiencia de usuario y velocidad sean óptimos.

5.2.6. *Framework*

Un *framework*[18] o marco de trabajo se puede definir como una plataforma concreta o conceptual en la que los desarrolladores o usuarios pueden especializar o sobreescribir de forma selectiva código común con funcionalidad genérica. Los *frameworks* adoptan la forma de bibliotecas, donde una interfaz de programa de aplicación (API) bien definida se puede reutilizar en cualquier lugar dentro del software en desarrollo.

Ciertas características hacen que un *framework* sea diferente de otras formas de biblioteca, incluidas las siguientes:

- Comportamiento predeterminado: antes de la personalización, un *framework* se comporta de una manera específica a la acción del usuario.
- Inversión de control: a diferencia de otras bibliotecas, el flujo de control global dentro de un *framework* lo emplea el *framework* en lugar del llamador.
- Extensibilidad: un usuario puede extender el *framework* reemplazando selectivamente el código predeterminado con el código de usuario.
- Código de *framework* no modificable: un usuario puede ampliar el *framework*, pero no modificar el código.

El propósito del *framework* de *software* es simplificar el entorno de desarrollo, lo que permite a los desarrolladores dedicar sus esfuerzos a los requisitos del proyecto, en lugar de lidiar con las bibliotecas y funciones mundanas y repetitivas del *framework*.

5.2.7. Fundación OWASP

La Fundación OWASP[19] (Open Web Application Security Project®) es una fundación sin ánimo de lucro que trabaja para mejorar la seguridad del *software* en la web. Por este motivo, la Fundación OWASP da visibilidad, credibilidad, financiación y una comunidad amplia a los proyectos de código abierto relacionados con la seguridad de las aplicaciones web en su plataforma.

OWASP Top Diez

El proyecto OWASP Top Diez[20] es un documento de concienciación para desarrolladores con los riesgos de seguridad más críticos para las aplicaciones web:

A1:2017-Injection
La inyección de código, SQL, NoSQL, OS o LADP, ocurre cuando datos sin verificar son enviados a un intérprete como parte de un comando o una consulta.

A2:2017-Broken Authentication
Las funciones de autenticación y gestión de la sesión a menudo están implementadas incorrectamente.

A3:2017-Sensitive Data Exposure
Los datos sensibles de las aplicaciones web no estan protegidos de forma correcta.

A4:2017-XML External Entities (XXE)
Muchos de los procesadores XML más antiguos o mal configurados evalúan las referencias de entidades externas dentro de los documentos XML.

A5:2017-Broken Access Control
Las restricciones en lo que los usuarios autenticados pueden hacer a menudo no se aplican correctamente.

A6:2017-Security Misconfiguration
La mala configuración de la seguridad es el problema más común. Suele ser el resultado de configuraciones predeterminadas inseguras, configuraciones incompletas, almacenamiento en la nube abierto, encabezados HTTP mal configurados y mensajes de error detallados con información confidencial.

A7:2017-Cross-Site Scripting XSS
Los fallos XSS ocurren cuando una aplicación incluye datos que no son de confianza en una nueva página web, sin la validación o la codificación apropiadas, o actualiza una página web existente con datos proporcionados por el usuario.

A8:2017-Insecure Deserialization
La deserialización es la reconstrucción de un objeto a partir de la información obtenida y cuando se hace de un modo inseguro a menudo puede resultar en la ejecución remota de código.

A9:2017-Using Components with Known Vulnerabilities
Los componentes, como bibliotecas o *frameworks*, tienen los mismos privilegios que la aplicación por lo que aprovecharse de las vulnerabilidades conocidas de estos componentes puede suponer la pérdida de datos o del propio servidor.

A10:2017- Insufficient Logging & Monitoring
El registro y la supervisión insuficientes, junto con una integración ineficaz con la respuesta a incidentes, permite mantener el ataque durante más tiempo y manipular, extraer o destruir datos. La mayoría de los estudios muestran que el tiempo de detección de una brecha de seguridad es superior a los doscientos días[9].

OWASP Top Diez Controles Proactivos

El proyecto OWASP Top Diez Controles Proactivos[21] es un documento de concienciación para desarrolladores con los controles que deben ser implementados en todos los proyectos.

C1: Define Security Requirements
Los requisitos de seguridad se derivan de los estándares de la industria, las leyes aplicables y un historial de vulnerabilidades pasadas. Los requisitos de seguridad definen nuevas funciones o adiciones a las funciones existentes para resolver un problema de seguridad específico o eliminar una vulnerabilidad potencial.

C2: Leverage Security Frameworks and Libraries
Las bibliotecas de codificación seguras y los marcos de *software* con seguridad incorporada ayudan a los desarrolladores a protegerse contra fallas de implementación y diseño relacionadas con la seguridad.

C3: Secure Database Access
Las áreas a considerar son: consultas seguras, configuración segura, autenticación segura y comunicación segura.

C4: Encode and Escape Data
Codificar y escapar son técnicas defensivas destinadas a detener los ataques de inyección.

C5: Validate All Inputs
La validación de entrada es una técnica de programación que garantiza que solo los datos formateados correctamente puedan ingresar a un componente del sistema de *software*.

C6: Implement Digital Identity
La identidad digital es la representación única de un usuario (u otro sujeto) mientras realiza una transacción en línea. La autenticación es el proceso de verificar que una persona o entidad es quien dice ser. La gestión de sesiones es un proceso mediante el cual un servidor mantiene el estado de la autenticación de los usuarios para que el usuario pueda continuar utilizando el sistema sin volver a autenticarse.

C7: Enforce Access Controls
El control de acceso (o autorización) es el proceso de otorgar o denegar solicitudes específicas de un usuario, programa o proceso. El control de acceso también implica el acto de otorgar y revocar esos privilegios.

Cabe señalar que la autorización (verificar el acceso a funciones o recursos específicos) no es equivalente a la autenticación (verificar la identidad).

C8: Protect Data Everywhere
Los datos confidenciales como contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos comerciales requieren protección adicional, particularmente si esos datos están sujetos a las leyes de privacidad (Reglamento General de Protección de Datos de la UE, RGPD), reglas de protección de datos financieros como el Estándar de Seguridad de Datos PCI (PCI DSS) u otras regulaciones.

C9: Implement Security Logging and Monitoring
El registro es un concepto que la mayoría de los desarrolladores ya utilizan con fines de depuración y diagnóstico. El registro de seguridad es un concepto igualmente básico: registrar información de seguridad durante la operación en tiempo de ejecución de una aplicación. El monitoreo es la revisión en tiempo real de los registros de seguridad y aplicaciones mediante varias formas de automatización. Las mismas herramientas y patrones se pueden utilizar para operaciones, depuración y seguridad.

C10: Handle All Errors and Exceptions
El tratamiento de los errores y las excepciones es un concepto de programación que permite que una aplicación responda a diferentes estados de error (como red inactiva, falla en la conexión a la base de datos,...) de varias maneras. Manejar las excepciones y los errores correctamente es fundamental para que su código sea confiable y seguro.

5.2.8. Huella digital del dispositivo

La huella digital o fingerprint del dispositivo es la recopilación sistemática de información en un dispositivo remoto específico con el objetivo de identificar, singularizar y, por tan-

to, poder monitorizar la actividad de su usuario para el propósito de la elaboración de perfiles[22].

Si bien, esta técnica de identificación puede resultar invasiva ya que a menudo se toma la huella digital sin tener en cuenta el consentimiento del usuario. Por este motivo, la AEPD recomienda que el procesamiento de datos utilizando técnicas de toma de huellas digitales de dispositivos debe seguir los criterios contenidos en *Guía sobre el uso de las cookies*[23] y las disposiciones del RGPD[1] sobre el tratamiento de datos personales.

La información que se puede obtener para tomar la huella digital del dispositivo es:

- Dirección IP.
- Sistema Operativo.
- Navegador.
- Lenguaje.
- Resolución de pantalla.
- Fuentes instaladas.
- Y muchas más...

Un sitio web interesante para comprobar como de identificable se es en Internet es *AmIUnique: Learn how identifiable you are on the Internet*[24].

5.2.9. Legislación aplicable

Protección de los datos personales

La normativa aplicable en materia de protección de datos de carácter personal es el *Reglamento general de protección de datos* (RGPD)[1] y sus posteriores correcciones del 23 de mayo de 2018 y del 4 de marzo de 2021; y la *Ley orgánica de Protección de Datos Personales y garantía de los derechos digitales* (LOPDPGDD)[2].

En el RGPD, entre otras cosas, se describen los derechos del usuario y las obligaciones de aquellos que tratan datos (organizaciones, empresas y administraciones públicas).

Los derechos del usuario son: derecho de acceso, rectificación, oposición, supresión («derecho al olvido»), limitación del tratamiento, portabilidad y de no ser objeto de decisiones individualizas. Además, el usuario tiene derecho a ser informado cuando se recaban sus datos de carácter personal (derecho de información).

El RGPD también señala un conjunto de principios que los responsables y encargados del tratamiento deben cumplir: el principio de «licitud, transparencia y lealtad»; el principio de «limitación de la finalidad»; el principio de «minimización de datos»; el principio de «exactitud»; el principio de «limitación del plazo de conservación»; y el principio de «integridad y confidencialidad».

Los responsables de los tratamientos de datos también deben tener en cuenta el principio de «responsabilidad proactiva», o *accountability* en inglés: deben aplicar las medidas técnicas y organizativas apropiadas para garantizar y poder demostrar que el tratamiento de datos personales se lleva a cabo de conformidad con el RGPD. Una de estas medidas es la privacidad desde el diseño y por defecto.

Sociedades de la información

Los sitios web de empresas o profesionales que ofrecen información sobre productos o servicios o los sitios web de particulares que generen ingresos para el titular también deben cumplir la *Ley de servicios de la sociedad de la información y de comercio electrónico* (LSSI)[3]. En concreto, la LSSI se aplica a los siguientes servicios: comercio electrónico, contratación en línea, información y publicidad y servicios de intermediación.

Entre las obligaciones de la LSSI destacan dos sobre el uso de las *cookies* en los sitios web: la obligación de transparencia y la obligación de obtención del consentimiento.

5.2.10. Modelo OSI

El modelo de interconexión de sistemas abiertos (OSI por sus siglas en inglés: «Open System Interconnection») es un modelo de referencia para los protocolos de la red[25]. Está conformado por siete capas:

7. Capa de Aplicación
Define los protocolos que utilizan las aplicaciones para intercambiar datos, como SMTP[1], FTP[2] o HTTP.

6. Capa de Presentación
Se encarga de la representación de la información de manera que, aunque distintos equipos puedan tener tener diferentes representaciones internas, los datos lleguen de manera reconocible.

5. Capa de Sesión
Se encarga de mantener y controlar el enlace establecido entre dos computadores que están transmitiendo datos de cualquier índole.

4. Capa de Transporte
Se encarga del transporte de los datos de la máquina de origen a la de destino, independientemente del tipo de red física que esté utilizando. Los dos principales ejemplos son el protocolo TCP[3] y UDP[4].

3. Capa de Red
Se encarga de identificar el enrutamiento existente entre una o más redes. El protocolo más conocido de esta capa es el IP[5].

2. Capa de Enlace de Datos
Se encarga del direccionamiento físico, del acceso al medio, de la detección de errores, de la distribución ordenada de tramas y del control del flujo.

1. Capa Física
Se encarga de la topología de red y de las conexiones globales de la computadora

[1]Protocolo para transferencia simple de correo (SMTP) es un protocolo de red utilizado para el intercambio de mensajes de correo electrónico.

[2]Protocolo de transferencia de archivos (FTP) para la transferencia de archivos entre sistemas conectados.

[3]El Protocolo de control de transmisión (TCP) proporciona un transporte fiable y bidireccional de datos.

[4]El protocolo de datagramas de usuario (UDP) permite el envío de datagramas a través de la red sin establecer una conexión.

[5]El Protocolo de Internet (IP) es un protocolo de comunicación de datos digitales.

hacia la red, se refiere tanto al medio físico como a la forma en la que se transmite la información y de las redes.

5.2.11. Protocolo HTTP

El protocolo Hypertext Transfer Protocol (HTTP) es el fundamento de Internet[26]. HTTP es un protocolo de la capa de aplicación para el intercambio de información: normalmente un dispositivo cliente que hace una petición a un servidor web. Es un protocolo sin estado, lo que significa que el servidor no guarda ningún dato entre dos peticiones[27].

Petición HTTP

El cliente pide una información concreta al servidor web con una petición HTTP con los siguientes elementos:

Versión
La versión del protocolo HTTPS de la petición. Por ejemplo, «HTTP/1.1».

URL
La URL, acrónimo de Uniform Resource Locator y más comúnmente conocida como la dirección web, es la dirección del recurso solicitado.

Método
HTTP define un conjunto de métodos de petición para indicar la acción que se desea realizar para un recurso determinado. Aunque también pueden ser sustantivos, los métodos de solicitud a veces son llamados verbos HTTP[28].

Los más habituales son:

- GET: solicita información al servidor.
- HEAD: solicita información al servidor sin el cuerpo de respuesta.
- POST: envía un recurso al servidor.
- PUT: reemplaza un recurso del servidor con la información de la petición.
- DELETE: borra un recurso del servidor.

Cabeceras
Las cabeceras, *headers* en inglés, HTTP permiten al cliente enviar información adicional junto a la petición[29].

Cuerpo de la petición
El cuerpo de una petición HTTP contiene la información siendo enviada al servidor web.

Respuesta HTTP

Códigos de respuesta de estado HTTP
Los códigos de respuesta de estado HTTP indican si una petición específica HTTP se ha completado con éxito[30]. Están agrupados en cinco clases:

- Información (100-199).

- Éxito (200-299).
- Redirección (300-399).
- Error del cliente (400-499).
- Error del servidor (500-599).

Cabeceras HTTP
Las cabeceras en las respuestas HTTP cumplen la misma función que en las peticiones HTTP. Si bien, es el servidor el que envía la información adicional junto a la respuesta al cliente.

Cuerpo de la respuesta
El cuerpo de la respuesta HTTP contiene la información solicitada por una petición HTTP GET del cliente.

5.2.12. Seguridad informática

La seguridad informática o ciberseguridad puede dividirse en seguridad del *hardware*, del *software* y de las redes; y tiene como objetivo la protección de los datos en un sistema informático.

Los pilares básicos de la seguridad de los datos son la confidencialidad, la integridad, la disponibilidad y la autenticación. Es decir, solo los usuarios autorizados pueden acceder o modificar la información que debe estar disponible cuando sea necesaria.

Sin embargo, hay multitud de amenazas y no existe un sistema informático infalible, pero se puede reducir el riesgo o minimizar el efecto con el diseño y las herramientas adecuadas: seudonimización de los usuarios, algoritmos de encriptación, copias de seguridad,...

Cifrado

El cifrado es un procedimiento en el que los datos digitales son transformados utilizando algoritmos criptográficos con una clave de cifrado. Después, estos datos pueden ser descifrados con la clave de descifrado.

Los dos tipos de métodos de cifrado más conocidos son:

- Criptografía simétrica: se utiliza la misma clave para cifrar y descifrar el mensaje.
- Criptografía asimétrica: se utilizan dos claves llamadas pública y privada.

Función *hash* criptográfica

Una función *hash* criptográfica es un proceso que transforma cualquier conjunto arbitrario de datos en una nueva serie de caracteres con una longitud fija, independientemente del tamaño de los datos de entrada.

Además, una función de *hash* segura ha de cumplir las propiedades siguientes[31]:

- Permite ejecutarse sobre contenido digital de cualquier tamaño y formato.
- Sin importar el tamaño de la entrada, la salida debe tener el mismo tamño.
- La misma entrada debe resultar siempre en el mismo *hash*.

- Reconstruir el mensaje original debe ser extremadamente costoso o imposible.
- Una variación mínima en el mensaje original debe producir un *hash* totalmente distinto (difusión).
- Si se selecciona un mensaje de entrada, encontrar otro mensaje que tenga el mismo *hash* debe ser extremadamente costoso (colisión fuerte).
- Encontrar dos mensajes distintos que obtengan el mismo *hash* debe ser extremadamente costoso (colisión fuerte).
- Cualquier resultado de la función *hash* debe tener la misma probabilidad de ocurrencia que cualquier otro.

Vector de ataque

Un vector de ataque hace referencia a un agujero o falla presente en la red, equipos y/o seguridad establecida, que permiten a los ciberdelincuentes realizar sus ataques.

5.2.13. *Stacks* tecnológicos

El *stack*, o pila, hace referencia a la combinación de lenguajes de programación y tecnologías que permiten implementar un sitio web desde la interfaz de usuario al propio sistema operativo, pasando por la base de datos.

Capítulo 6

Estudios y decisiones

6.1. Elección de la licencia

Al tratarse de un proyecto de código abierto, una de las primeras, y más importantes decisiones, que se ha tomado es la licencia con la que se distribuye el *software* desarrollado. Porque de publicar las plantillas sin licencia automáticamente se aplican las leyes de derechos de autor, lo que significa que el creador conserva todos los derechos sobre el código fuente y nadie más puede reproducir, distribuir o crear trabajos derivados.

Esta no es la intención del proyecto: no solo busca recoger consejos, buenas prácticas y ejemplos para un desarrollo web seguro, sino que las organizaciones, empresas y particulares puedan utilizar las plantillas implementadas como semilla para sus propios sitios web y así garantizar un mínimo de seguridad a los usuarios.

Por este motivo se ha escogido la licencia MIT[32]. La licencia MIT, cuyo texto completo puede encontrarse más abajo, es una licencia permisiva y cuyas condiciones solo requieren la preservación de los avisos de licencia y *copyright*, mientras que permite que los trabajos con licencia, las modificaciones y los trabajos más grandes pueden distribuirse bajo diferentes términos y sin código fuente.

Además, como la mayoría de licencias de código abierto, la licencia también protege al autor del código al remarcar que el *software* se proporciona sin garantía de ningún tipo y que el autor será responsable de ningún modo de cualquier reclamo o daño provocado por o conectado con el *software* proporcionado.

Texto de la licencia MIT

```
MIT License

Copyright (c) year fullname

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
```

```
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

6.2. Elección de los estándares

La segunda decisión que se ha tomado antes de empezar la investigación, el diseño y la implementación ha sido adherirse a tres estándares o convenciones para mejorar la calidad y la legibilidad del código producido, además de facilitar la comprensión y la cooperación de terceros una vez terminado el proyecto de final de grado.

6.2.1. Estándar de Git

Git[1] es una herramienta imprescindible en la actualidad para el trabajo de un programador y aunque no existe un estándar concreto, sí que hay un conjunto de convenciones y buenas prácticas para el uso y la denominación de las ramas y la descripción de los mensajes de *commits*.

Git Branching

Cuando se trabaja con las ramas de Git hay que tener en cuenta dos cosas: la estructura y la nomenclatura de las ramas. La estructura[33], se puede ver un ejemplo del flujo de trabajo en la figura 6.1, que se ha utilizado para el desarrollo del proyecto es la siguiente:

- Tiene dos ramas principales:
  - *main*[2]: la rama con los últimos cambios de desarrollo revisados y aceptados.
  - *stable*: la rama con la última versión del código desplegado en producción.
- Y tres tipos de ramas de soporte o secundarias:
  - Ramas de *features*: se utilizan cuando se desarrolla una nueva característica o mejora. Empiezan en *main* y terminan en *main*.
  - Ramas de *bugs*: se utilizan cuando hay un error que puede esperar a la próxima versión. Empiezan en *main* y terminan en *main*.
  - Rama de *hotfixs*: se utilizan cuando se tiene que actuar de inmediato en la versión de producción. Empiezan en *stable* y terminan en *stable* y en *main*.

[1] Git es el sistema de control de versiones más utilizado actualmente.

[2] La rama *main* es llamada *master* en repositorios más antiguos. Este cambio se realizó a finales del año 2020 para evitar las connotaciones racistas de los términos *master* y *slave*.

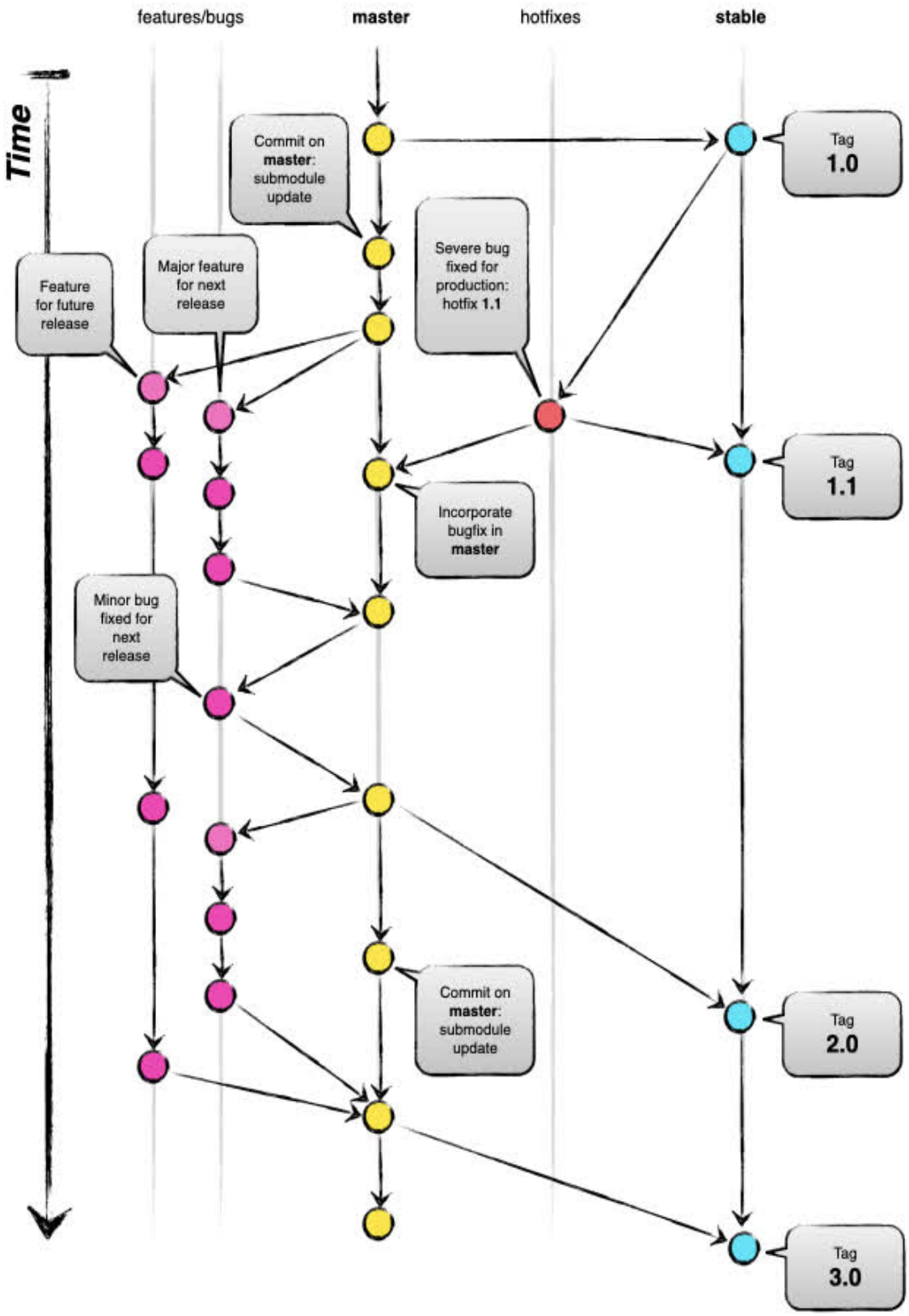

Figura 6.1: Flujo de trabajo Git[33]

Las reglas que se han escogido para la nomenclatura de las ramas de soporte son:

1. Empezar por el tipo de rama al que pertenecen: *feature*, *bug* o *hotfix*.
2. Seguir con el identificador numérico único de la tarea que se desarrolla en la rama.
3. Y acabar con una descripción corta que explique el propósito de la rama.
4. Por último, utilizar guiones bajos como separadores.

Algunos ejemplos de nomenclatura de posibles ramas son:

- *feature_1_add_log*: rama para añadir el *log* en la plantilla.
- *feature_2_add_error_handling*: rama para añadir el tratamiento de errores en la plantilla.
- *feature_3_add_https*: rama para añadir el certificado y la configuración HTTPS en la plantilla.

Git Commit

Los mensajes de *commit* tienen que ser concisos y precisos y explicar el porqué de los cambios a los otros desarrolladores del equipo. Tampoco existe un estándar para el formato de los mensajes de *commit*, pero sí hay siete normas[34] bien establecidas para escribir el asunto y el cuerpo del *commit*:

1. Separar el asunto del cuerpo con una línea en blanco.
2. Limitar el asunto a 50 caracteres.
3. Poner en mayúscula la primera letra del asunto.
4. No terminar el asunto con un punto.
5. Usar el imperativo en el asunto.
6. Las líneas del cuerpo tienen que ser como máximo de 72 caracteres.
7. Usar el cuerpo para explicar el «qué» y el «por qué» en vez del «cómo».

Un poco más abajo puede verse un mensaje de *commit* que sigue las siete normas anteriores obtenido del artículo web *How to Write a Git Commit Message*[34].

Ejemplo de un mensaje de commit de Git

```
commit eb0b56b19017ab5c16c745e6da39c53126924ed6
Author: Pieter Wuille <pieter.wuille@gmail.com>
Date:   Fri Aug 1 22:57:55 2014 +0200

   Simplify serialize.h's exception handling

   Remove the 'state' and 'exceptmask' from serialize.h's stream
   implementations, as well as related methods.

   As exceptmask always included 'failbit', and setstate was always
   called with bits = failbit, all it did was immediately raise an
   exception. Get rid of those variables, and replace the setstate
   with direct exception throwing (which also removes some dead
```

```
code).

As a result, good() is never reached after a failure (there are
only 2 calls, one of which is in tests), and can just be replaced
by !eof().

fail(), clear(n) and exceptions() are just never called. Delete
them.
```

6.2.2. Estándar de JavaScript

El estándar de código de JavaScript escogido es «JavaScript Standard Style»[35] que además cuenta con un paquete instalable con npm llamado standardJS. Sus tres ventajas son:

- No necesita configuración.
- Formatea automáticamente el código.
- Detecta problemas de estilo y errores del programador sin necesidad de revisar el código manualmente.

Para utilizarlo, se ha instalado standardJS localmente en el repositorio[3] como un programa de línea de comandos de Node.js:

```
$ npm install standard --save-dev
```

Esto permite ejecutarlo de la forma más simple, que revisa todos los archivos del directorio actual:

```
$ npx standard
```

Sin embargo, una buena práctica es añadirlo al «package.json» del repositorio como una dependencia:

```
{
  "name": "my-cool-package",
  "devDependencies": {
    "standard": "*"
  },
  "scripts": {
    "test": "standard && node my-tests.js"
  }
}
```

Así puede revisarse el formato del código con:

```
$ npm test
```

Además, también cuenta con una extensión de Visual Studio Code para hacer la revisión del código en tiempo real, «Standard JS - JavaScript Standard Style»[36], y otra con *snippets* para acelerar la programación, «JavaScript standardjs styled snippets»[37].

[3] Tiene que haberse instalado antes Node.js y npm

Algunas de las normas del estándar son:

- Usa 2 espacios para la sangría.
- Utiliza comillas simples para los *strings*, excepto para evitar que se escapen.
- Sin variables no utilizadas.
- Añada un espacio después de las palabras clave.

La lista completa con ejemplos correctos e incorrectos puede encontrarse en *JavaScript Standard Style*[38].

6.2.3. Estándar de PHP

El estándar de código de PHP escogido es el PSR: PHP Standard Recommendation de PHP-FIG. FIG es un acrónimo de Framework Interoperability Group[39] cuyo objetivo es encontrar puntos en común y formas de trabajar juntos entre los diferentes *frameworks* que forman parte del grupo.

La lista completa de las trece recomendaciones aceptadas, explicadas en detalle en el sitio web de PSR[40], son:.

PSR-1: Basic Coding Standard
: Comprende lo que deben considerarse los elementos de codificación estándar que se requieren para garantizar un alto nivel de interoperabilidad técnica entre el código PHP compartido: nomenclatura de ficheros, codificación, ...

PSR-3: Logger Interface
: Describe una interfaz común para las bibliotecas de registro.

PSR-4: Autoloading Standard
: Describe una especificación para clases de *autoloading* a partir de rutas de archivo. También describe dónde colocar los archivos que se cargarán automáticamente de acuerdo con la especificación.

PSR-6: Caching Interface
: Describe una interfaz común para las interfaces de almacenamiento en caché.

PSR-7: HTTP Message Interface
: Describe interfaces comunes para representar mensajes HTTP como se describe en RFC 7230 y RFC 7231, y URI para usar con mensajes HTTP como se describe en RFC 3986.

PSR-11: Container Interface
: Describe una interfaz común para contenedores de inyección de dependencia.

PSR-12: Extended Coding Style Guide
: Describe restricciones de código nuevas a PSR-1.

PSR-13: Hypermedia Links
: Tiene como objetivo proporcionar a los desarrolladores de PHP una forma simple y común de representar un enlace hipermedia independientemente del formato de serialización que se utilice

PSR-14: Event Dispatcher
Tiene como objetivo establecer un mecanismo común para la extensión y la colaboración basadas en eventos para que las bibliotecas y los componentes se puedan reutilizar más libremente entre varias aplicaciones y *frameworks*.

PSR-15: HTTP Handlers
Describe interfaces comunes para los controladores de solicitudes del servidor HTTP («controladores de solicitudes») y los componentes de *middleware* del servidor HTTP («*middleware*») que utilizan mensajes HTTP como se describe en PSR-7.

PSR-16: Simple Cache
Describe una interfaz simple pero extensible para un elemento de caché y un controlador de caché.

PSR-17: HTTP Factories
Describe un estándar común para las fábricas que crean objetos HTTP compatibles con PSR-7.

PSR-18: HTTP Client
Describe una interfaz común para enviar solicitudes HTTP y recibir respuestas HTTP.

Para revisar el formato del código de forma automática se ha utilizado «PHP CodeSniffer»[41] que cuenta además con una extensión de Visual Studio Code: «PHP Sniffer & Beautifier»[42].

Para poder utilizar la extensión mencionada se han seguido los pasos siguientes:

1. Primero, se ha instalado composer[43], un administrador de dependencias de PHP.
2. Después, se ha instalado «PHP CodeSniffer»[4] localmente en el repositorio:

```
composer require --dev squizlabs/php_codesniffer
```

3. Por ultimo, se ha añadido la extensión «PHP Sniffer & Beautifier» en Visual Studio Code

6.3. Elección de los *stacks*

6.3.1. LAMP

La elección del *stack* LAMP es fácil de justificar: fue el primero que apareció en 1998 y aunque la popularidad del PHP ha decaído en los últimos años[44] es un lenguaje de desarrollo web maduro y consolidado: el 79.1 % de los sitios web actuales lo utilizan como lenguaje de programación para el lado del servidor[45]; siendo la segunda opción ASP.NET con un 8.5 %.

Sin embargo, esto no quiere decir que todos los sitios web que utilizan PHP lo hagan con este *stack*: hay otras alternativas como WAMP o MAMP, cuyo sistema operativo es Windows o MacOS, respectivamente, en vez de Unix; o que sustituyen el servidor web Apache

[4] «PHP CodeSniffer» requiere la versión PHP 5.4.0 o superior

por Nginx, el más popular en la actualidad[46]; también hay los que prefieren otras bases de datos SQL, como PostgreSQL, o una base de datos NoSQL, como MongoDB.

Algunas de las ventajas[47] de LAMP son:

- Todos los componentes del *stack* son de código abierto.
- Cuenta con más de veinte años de historia por lo que hay un ecosistema de bibliotecas, extensiones y *frameworks* enorme y una comunidad muy amplia.
- Es escalable, seguro, personalizable y de desarrollo rápido.

Por estos motivos se ha escogido el *stack* LAMP con los componentes siguientes: Ubuntu Server como sistema operativo, Apache como servidor web, MySQL como base de datos y Laravel como *framework* de PHP. Laravel permite utilizar Vue.js, cuya justificación se encuentra en la subsección 6.3.2, como *framework* de interfaz de usuario por lo que solo ha tenido que diseñarse e implementarse una interfaz para los dos *stacks*.

6.3.2. MEVN

El uso del lenguaje JavaScript en el lado del servidor es más bien minoritario: solo un 1.4 % de los sitios web lo utilizan[45], aunque esto ya significa más de veinte millones de sitios web; pero hegemónico cuando se trata del lado del cliente: el 97.4 % de las interfaces de usuario están programadas con JavaScript[48] y no existe actualmente una alternativa factible.

Es por este motivo que la ventaja principal de escoger un *stack* MEAN, MERN o MEVN es la facilidad de aprendizaje y el aumento de la velocidad de desarrollo ya que solo es necesario dominar un lenguaje de programación para implementar todas las capas del *stack*: algo especialmente importante para *startups* ya que el equipo de desarrollo suele ser pequeño y el presupuesto limitado.

Los tres *stacks* anteriores tienen en común la base de datos no relacional, MongoDB, cuya afinidad con el lenguaje JavaScript es que ambos utilizan objetos JSON[5] para el intercambio de información; la infraestructura de aplicaciones web, Express.js; y el entorno de ejecución, Node.js; pero difieren en el *framework* para la construcción de interfaces de usuario: utilizan Angular, React y Vue.js respectivamente.

De los tres *frameworks* de interfaz de usuario anteriores, se ha escogido Vue.js por los siguientes motivos:

- Tiene una curva de aprendizaje más rápida que Angular, pero similar a React.
- Puede utilizarse como solución *frontend* para el *stack* LAMP. Igual que React.
- Ha crecido en popularidad en los últimos años dada su simplicidad y por ser «*developer friendly*»: la documentación de Vue.js es muy buena y el paquete «Vue CLI» y su interfaz de usuario gráfica permiten crear el proyecto a partir de numerosas plantillas.
- Y finalmente, por interés personal del autor del proyecto.

[5] JSON (JavaScript Object Notation - Notación de Objetos de JavaScript) es un formato ligero de intercambio de datos[49].

Otra ventaja de este tipo de *stack* es que Node.js es un entorno multiplataforma por lo que la misma aplicación puede implantarse en distintos sistemas operativos sin el menor cambio.

Capítulo 7

Recopilación de buenas prácticas para un desarrollo web seguro

A continuación se describen los consejos y buenas prácticas recopilados para el desarrollo seguro de un sitio web durante la realización del proyecto de final de grado.

7.1. El protocolo HTTPS

7.1.1. ¿Qué es el protocolo HTTPS?

Hypertext transfer protocol secure (HTTPS) es la versión cifrada de HTTP: el protocolo de aplicación estándar para la transferencia de datos entre el cliente y el servidor de un sitio web.

El protocolo HTTPS usa un canal cifrado con Transport Layer Security (TLS) o su predecesor, Secure Sockets Layer (SSL), para transmitir los datos de forma segura entre las dos partes.

También existe la política de seguridad HTTP Strict Transport Security (HSTS) para asegurar que los navegadores web solo puedan comunicarse con el servidor utilizando el protocolo HTTPS como respuesta a un tipo particular de ataque MITM (Man-In-The-Middle por sus siglas en inglés): el ataque SSL-stripping[50].

7.1.2. ¿Por qué es importante el protocolo HTTPS?

En la figura 7.1, obtenida de *What is HTTPS? Everything You Need to Know*[51], se puede ver las dos aproximaciones distintas de un posible ataque de intermediario (MITM por sus siglas en inglés: Man In The Middle) al flujo de una conversación HTTP.

En la primera aproximación, el atacante se conforma con espiar la información que se transmiten el cliente y el servidor; mientras que en la segunda, toma una dirección más activa y decide manipular la respuesta del servidor para inyectar código malicioso (*malware*) al navegador del cliente. Este tipo de ataque no es posible, o no tan fácilmente, si el servidor utiliza HTTPS y no permite al cliente comunicarse con él de otro modo.

Por este motivo, el protocolo HTTPS es un requisito fundamental de seguridad para los sitios web con autenticación de usuarios y otros datos sensibles, como pueden ser

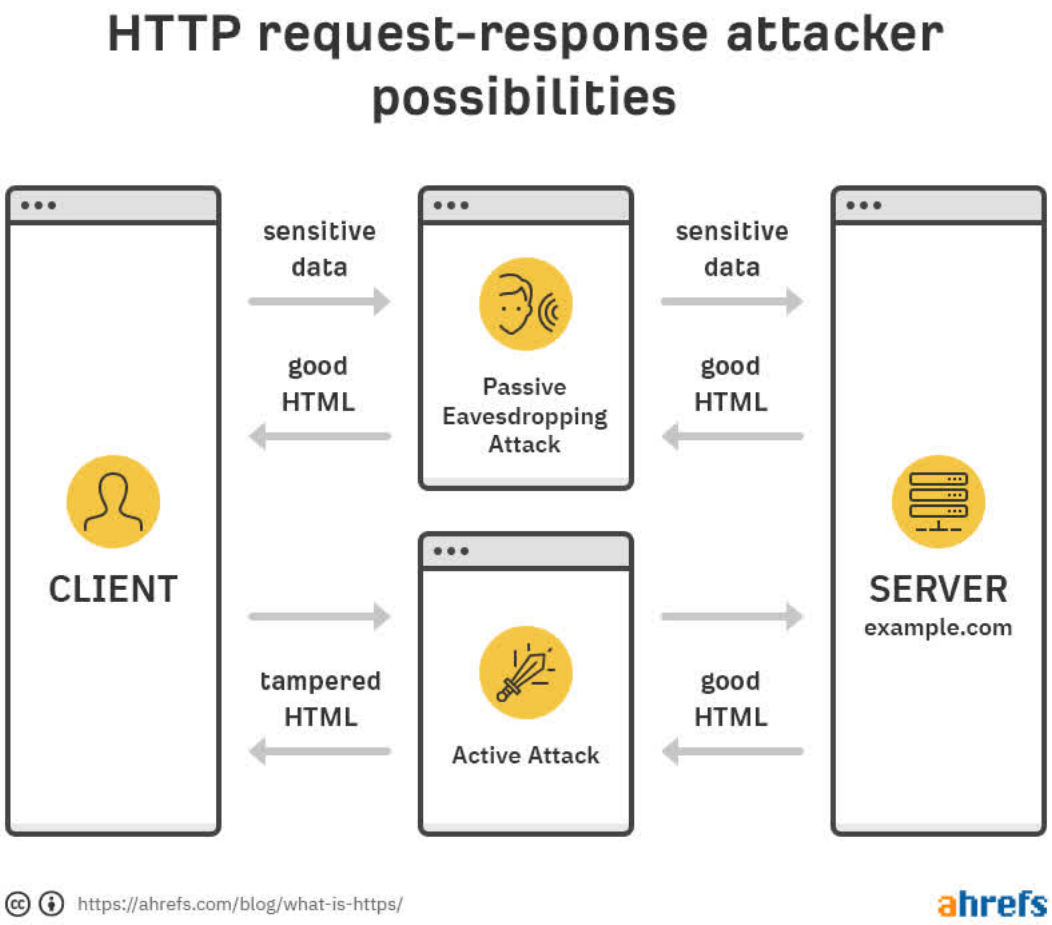

Figura 7.1: Comunicación HTTP petición-respuesta con un atacante de intermediario

detalles de pago, para asegurar la confidencialidad de las peticiones y respuestas HTTPS durante la transmisión. Además, también permite verificar la identidad del servidor frente el cliente.

7.1.3. ¿Cómo funciona el protocolo HTTPS?

Los pasos que un cliente y un servidor siguen para establecer el canal SSL/TLS y así poder comunicarse con el protocolo HTTPS son:

1. (Previo a la comunicación) El servidor solicita un certificado TLS a una entidad certificadora de su confianza.
2. Se establece la conexión SSL/TLS con el protocolo *handshake* o de acuerdo para que:
   - El cliente verifique la identidad del servidor.
   - Las dos partes se pongan de acuerdo en el *cipher suite* o conjunto de cifrado que incluye el algoritmo de encriptación y el tamaño de la llave.

- Las dos partes se intercambien o generan la misma clave[1] con un algoritmo de encriptación simétrico[2].

3. Finalmente, ya puede empezar la conversación HTTPS entre el cliente y el servidor.

7.1.4. ¿Cómo puede mejorarse la configuración del protocolo HTTPS?

Como se ha visto en la subsección anterior una parte muy importante de la seguridad del protocolo HTTPS radica en el canal cifrado SSL/TLS y en el protocolo de *handshake* por lo que es recomendable seguir los consejos siguientes:

Protocolos SSL/TLS seguros

Solo es recomendable utilizar los protocolos TLS 1.2 y TLS 1.3.

Los protocolos SSL 2.0 y SSL 3.0 tienen fallos conocidos (ataques DROWN y POODLE respectivamente) y los protocolos TLS 1.0 y TLS 1.1 (ataque BEAST) están considerados obsoletos por lo que deberían ser evitados. Sin embargo, esto significa dejar sin acceso al sitio web a los dispositivos más viejos. Por ejemplo, Internet Explorar 10 en Windows 7 solo soporta TLS 1.0.

***Cipher suites* seguros**

Un error habitual de la configuración SSL/TLS de los servidores web es no tener uno o más *cipher suites* preferidos.

> Se debe confiar principalmente en los *cipher suites* AEAD (Authenticated Encryption with Associated Data) que brindan autenticación sólida e intercambio de claves, secreto de reenvío y cifrado de al menos 128 bits. Es posible que aún se admitan otros *cipher suites* más débiles, siempre que se negocien solo con clientes más antiguos que no admitan nada mejor.
>
> (Qualys, Inc., *SSL and TLS Deployment Best Practices*)

Una lista de *cipher suites* de TLS 1.2 y TLS 1.3 con un nivel de compatibilidad intermedio y considerados seguros actualmente[54] es:

```
TLS_AES_128_GCM_SHA256
TLS_AES_256_GCM_SHA384
TLS_CHACHA20_POLY1305_SHA256
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-ECDSA-CHACHA20-POLY1305
ECDHE-RSA-CHACHA20-POLY1305
DHE-RSA-AES128-GCM-SHA256
DHE-RSA-AES256-GCM-SHA384
```

Además, Mozilla Foundation ofrece un sitio web para generar la configuración SSL/TLS (protocolos y *cipher suites*), con tres niveles de restricción, en diferentes *softwares* de servidores: *Mozilla SSL Configuration Generator*[55].

[1] El cliente genera una clave aleatoria y la cifra con la clave pública del servidor que se encuentra en el certificado TLS o el cliente y el servidor generan cada uno una parte de la clave que luego es transmitida y juntada de tal modo que ambos lados terminan con la misma clave.

[2] Se emplea una sola clave para cifrar y descifrar los mensajes[52]

Protocolo HTTPS

Los servidores web deben de comunicarse con sus clientes con el protocolo HTTPS. Algunas configuraciones o implementaciones adicionales ordenadas por importancia[56] son:

1. Todos los recursos, ya sean *scripts* de JavaScript o contenido estático, como imágenes, deben de ser cargados por el canal seguro HTTPS.
2. Debe redirigirse el tráfico del puerto 80 (HTTP) a HTTPS con el código de respuesta 301, si es al mismo recurso, o con el código 302, si se redirige a otro camino.
3. Debe añadirse el encabezado HTTPS «HTTP Strict Transport Security (HSTS)» que notifica a los navegadores web que el servidor solo acepta comunicaciones HTTPS.
4. Debe añadirse el encabezado HTTPS «X-Frame-Options» que permite que los sitios controlen cómo se puede enmarcar su sitio dentro de un *iframe*[3].
5. Debe añadirse el encabezado HTTPS «X-Content-Type-Options» que indica a los navegadores que no carguen *scripts* y hojas de estilo a menos que el servidor indique el tipo MIME[4] correcto.
6. Debe añadirse el encabezado HTTPS «Content-Security-Policy» (CSP) para controlar de dónde pueden ser cargados los recursos.

   Una política de CSP básica, pero bastante segura en que solo puede cargarse el contenido desde el propio dominio, es, por ejemplo:

   ```
   default-src 'none';
   frame-ancestors 'self';
   script-src 'self';
   style-src 'self';
   img-src 'self';
   connect-src 'self';
   base-uri 'self';
   form-action 'self';
   upgrade-insecure-requests;
   block-all-mixed-content;
   ```

   Sin embargo, una política de CSP segura es también más restrictiva por lo que a veces es una buena práctica implementar una política un poco más laxa, pero añadiendo el encabezado «Content-Security-Policy-Report-Only» que advierte de las violaciones de la política.

   Esta también es una buena forma de probar políticas nuevas o el comportamiento de una página web existente antes de añadir el encabezado de CSP.

Por último, la información anterior se ha extraído de, y los interesados pueden ampliarla con:

[3] Un *iframe* es un elemento HTML que permite incrustar otra página HTML en la página actual[57]

[4] El tipo MIME es un estándar que indica la naturaleza y el formato de un documento, archivo o variedad de bytes[58].

- *SSL and TLS Deployment Best Practices*[53].
- *Transport Layer Protection - OWASP Cheat Sheet Series*[59].
- *HTTP Strict Transport Security - OWASP Cheat Sheet Series*[60].
- *Web Security*[56].
- *Security/Server Side TLS - MozillaWiki*[54].
- *Content Security Poliy (CSP) - HTTP | MDN*[61].
- *Content-Security-Policy - HTTP | MDN*[62].
- *Content-Security-Policy-Report-Only - HTTP | MDN*[63].
- *Content Security Policy - OWASP Cheat Sheet Series*[64].

7.1.5. ¿Cómo puede comprobarse la configuración del protocolo HTTPS?

Se puede probar la configuración HTTPS del servidor en producción con la página web *SSL Server Test (Powered by Qualys SSL Labs)*[65][5] que evalúa cuatro características de la configuración de los protocolos HTTPS y SSL/TLS[67]:

Certificado
: Comprueba que el certificado es valido y de confianza: es decir, que está signado por una entidad certificadora conocida.

Soporte de protocolos
: Comprueba los protocolos SSL/TLS soportados por el servidor: SSL 2.0, SSL 3.0, TLS 1.0, TLS 1.1, TLS 1.2 y TLS 1.3.

Intercambio de llaves
: Comprueba la seguridad de la fase de intercambio de las llaves: los algoritmos utilizados y el tamaño de la llave para autenticar la identidad de las partes y asegurar la generación y el intercambio de las llaves utilizadas durante el resto de la sesión.

Fuerza del cifrado
: Comprueba la seguridad del algoritmo de cifrado que se utiliza para las comunicaciones. La fuerza del cifrado tendría que ser, como mínimo de 128 bits.

Además, para probar la configuración y la implementación de los encabezados de HTTPS puede utilizarse *Mozilla Observatory*[68]. Las dos siguientes herramientas también permiten evaluar la política de seguridad de CSP: *CSP Evaluator*[69] y *CSP Scanner*[70].

7.2. Registro

7.2.1. ¿Qué es el registro?

Un registro, o *log* en inglés, es un registro de los eventos que ocurren dentro de los sistemas y redes de una organización. Los registros se componen de entradas de registro;

[5] Otras herramientas para probar la configuración SSL/TLS y HTTPS del servidor pueden encontrarse en *Transport Layer Protection: Test the server configuration - OWASP Cheat Sheet Series*[66]

cada entrada contiene información relacionada con un evento específico que ha ocurrido dentro de un sistema o red.

Originalmente, los registros se usaban principalmente para solucionar problemas, pero ahora cumplen muchas funciones dentro de la mayoría de las organizaciones, como optimizar el rendimiento del sistema y la red, registrar las acciones de los usuarios y proporcionar datos útiles para investigar actividades maliciosas[71].

7.2.2. ¿Por qué es importante el registro?

Los registros proporcionan persistencia y trazabilidad a los eventos que ocurren en las aplicaciones para su estudio posterior y así detectar el origen de un problema o los pasos que ha seguido un atacante para vulnerar la seguridad de la aplicación. Sistemas más avanzados de gestión de registros pueden incluso avisar en tiempo real de los posibles problemas o atacantes.

Sin embargo, la seguridad no es el único beneficio de implementar un buen sistema de gestión de registros: también permite analizar el comportamiento de los usuarios de la aplicación para conocer las horas punta de actividad, desde qué dispositivo se conectan o qué partes de la aplicación visitan.

7.2.3. ¿Cómo puede mejorarse la gestión del registro?

Algunos recomendaciones para implementar un sistema de gestión de registros seguro y eficaz son:

- Los ficheros de registro deben protegerse para evitar la manipulación externa y así garantizar la fiabilidad de los datos registrados.
- Los ficheros de registro no deben ser eliminados ya que pueden necesitarse para investigar un problema o un ataque en el futuro[6].
- Las entradas del registro deben tener diferentes niveles de severidad. Como mínimo, los tres niveles siguientes (de más prioritario a menos):

  Error
  : Este nivel se utiliza cuando un problema grave impide que las funciones dentro de la aplicación funcionen de manera correcta.

  Advertencia
  : Este nivel se utiliza cuando se detecta un problema de aplicación inesperado.

  Información
  : Este nivel brinda información sobre lo que sucede mientras se ejecuta la aplicación.

- Las entradas del registro deben contener la información suficiente para saber «cuándo», «dónde», «quién» y «qué»[72].

  Cuándo

  - Fecha y hora del registro.

[6] Se necesita de media dos cientos días para detectar un ataque[9].

- Identificador de la transacción[7].

Dónde

- Identificador de la aplicación: nombre y versión.
- Dirección de la aplicación: *hostname* y puerto.
- URL de entrada y método HTTP.
- Localización en el código.

Quién

- Dirección de origen: identificador del dispositivo, dirección IP,...
- Identificador del usuario, si conocido.

Qué

- Severidad del registro.
- Descripción.

- Las entradas del registro deben seguir un estándar que permita a otra aplicación extraer la información con facilidad: un buen formato para las entradas puede ser JSON[73].
- Los eventos que deben registrarse son[72]:
  - Fallos en la validación de entradas. Por ejemplo, violaciones de protocolo, codificaciones no aceptadas,...
  - Fallos en la validación de salidas. Por ejemplo, discordancia en la base de datos, codificación de los datos inválidos,...
  - Autenticación de los usuarios: exitosos o no.
  - Fallos de autorización.
  - Fallos en la gestión de sesión. Por ejemplo, modificación de los valores de *cookies* de sesión.
  - Errores de la aplicación.
  - Arranques y cierres de la aplicación y sistemas relacionados, e inicialización del registro.
  - Uso de funciones de alto riesgo. Por ejemplo, adición o eliminación de usuarios, cambios de privilegios, uso de los privilegios del sistema de administración,...
  - *Opt-ins* legales o de otros tipos. Por ejemplo, permisos para utilizar características del teléfono, términos de uso, consentimiento de uso de datos personales,...
- Además hay cierta información que nunca debe registrarse[72]:
  - Código fuente de la aplicación.

[7]Permite enlazar las entradas del registro relacionadas del mismo usuario. Por ejemplo, diferentes fallos en validación de entradas en un espacio corto de tiempo podría suponer un intento de vulnerar el sistema.

- Identificadores de sesión.
- *Tokens* de acceso.
- Datos de carácter personal.
- Contraseñas de los usuarios.
- Llaves de encriptación y otros secretos.
- Cuentas bancarias o datos de tarjetas de pago.
- Información confidencial o ilegal de registrar.

- Por último, existe multitud de *software* de código abierto en internet que permite mantener un registro centralizado. Este tipo de *software* permite gestionar y analizar los registros de decenas de aplicaciones o servidores al mismo tiempo por lo que su uso en organizaciones de mayor tamaño es muy recomendado. Dos ejemplos son Graylog[74] o el *stack* Elastic[75][8].

7.3. Tratamiento de los errores

7.3.1. ¿Qué es el tratamiento de los errores?

El tratamiento de los errores se refiere a los procedimientos de respuesta y recuperación de las condiciones de error presentes en una aplicación de *software*. En otras palabras, es el proceso que comprende la anticipación, detección y resolución de los errores de aplicación, programación o comunicación. El tratamiento de los errores ayuda a mantener el flujo normal de ejecución del programa[76].

7.3.2. ¿Por qué es importante el tratamiento de los errores?

El tratamiento de errores en una aplicación de *software* es esencial porque los errores son inevitables, por muy extensas que sean las pruebas realizadas durante o después del desarrollo, y a veces pueden ser fatales.

Un tratamiento inadecuado de los errores en el *software* puede causar una leve molestia, frustración e incluso crear problemas que tardarían horas en resolverse. Además, la mala gestión de los errores hace que el producto parezca inacabado, crudo y descuidado: socava la confianza de los usuarios[77].

Otro motivo de peso es que el tratamiento de errores forma parte de la seguridad general de una aplicación. Un ataque siempre empieza con una fase de reconocimiento en la que el atacante intentará recopilar la mayor cantidad de información técnica posible sobre el objetivo, como el servidor de aplicaciones, los *frameworks*, las bibliotecas,... Los errores no tratados pueden ayudar a un atacante en esta fase inicial[78] ya que normalmente incluyen la pila de llamadas que ha causado el ataque.

Ejemplo de una excepción sin tratar

[8]Los components del stack son: Elasticsearch, el motor de búsqueda y analítica distribuido con base en JSON; Kibana, la interfaz de usuario; Beats, la plataforma para agentes ligeros que envían datos; y Logtash, la *pipeline* de recopliación de datos dinámico.

```
HTTP Status 500 - For input string: "null"

type Exception report

message For input string: "null"

description The server encountered an internal error that prevented it from fulfilling
    this request.

exception

java.lang.NumberFormatException: For input string: "null"
    java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
    java.lang.Integer.parseInt(Integer.java:492)
    java.lang.Integer.parseInt(Integer.java:527)
    sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.java:57)
    sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
    java.lang.reflect.Method.invoke(Method.java:606)
    com.opensymphony.xwork2.DefaultActionInvocation.invokeAction(DefaultActionInvocation.java:450)
    com.opensymphony.xwork2.DefaultActionInvocation.invokeActionOnly(DefaultActionInvocation.java:289)
    com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:252)
    org.apache.struts2.interceptor.debugging.DebuggingInterceptor.intercept(DebuggingInterceptor.java:256)
    com.opensymphony.xwork2.DefaultActionInvocation.invoke(DefaultActionInvocation.java:246)
    ...

note: The full stack trace of the root cause is available in the Apache Tomcat/7.0.56 logs.
```

7.3.3. ¿Cómo funciona el tratamiento de los errores?

El tratamiento de los errores o de las excepciones depende enormemente de la arquitectura y del lenguaje que se utilice, pero el procedimiento es similar en la mayoría de lenguajes[77]:

1. Un método se encuentra con un error, esperado o no, y lanza una excepción. La «excepción» contiene la información del error: dónde se ha lanzado, el tipo de error y el estado del programa.
2. Se recorre la pila de llamadas en orden para encontrar un método que sea capaz de tratar la excepción.
3. La excepción es «atrapada» por el método con el *handler* adecuado. De no existir uno, la excepción provoca el cierre inesperado del programa.

En las figuras 7.2 y 7.3 se puede ver el procedimiento habitual de tratamiento de los errores en Java.

7.3.4. ¿Cómo puede mejorarse el tratamiento de los errores?

Al tratarse de un tema muy dependiente del lenguaje utilizado, los consejos particulares de como implementar un sistema de tratamiento de errores eficaz pueden encontrarse en el capítulo 10, más concretamente en la subsección 10.2.5 para el *stack* MEVN; y en la subsección 10.3.5 para el *stack* LAMP.

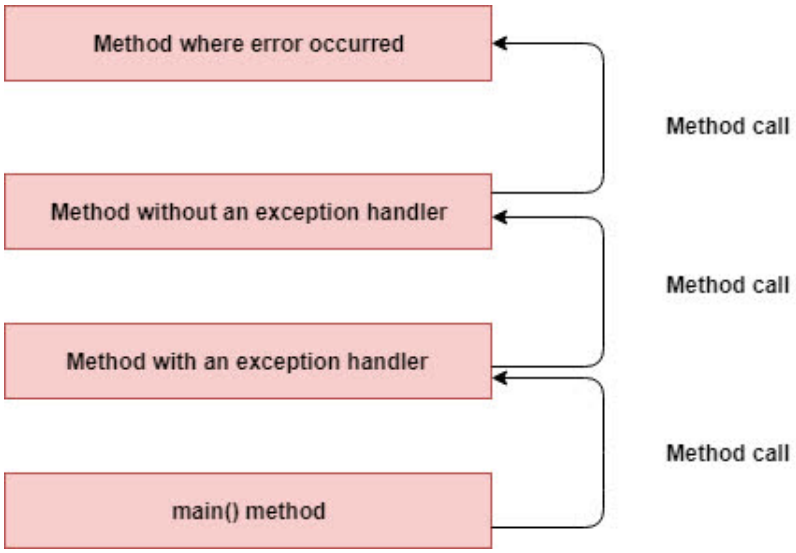

Figura 7.2: Pila de llamadas que provoca una excepción[79]

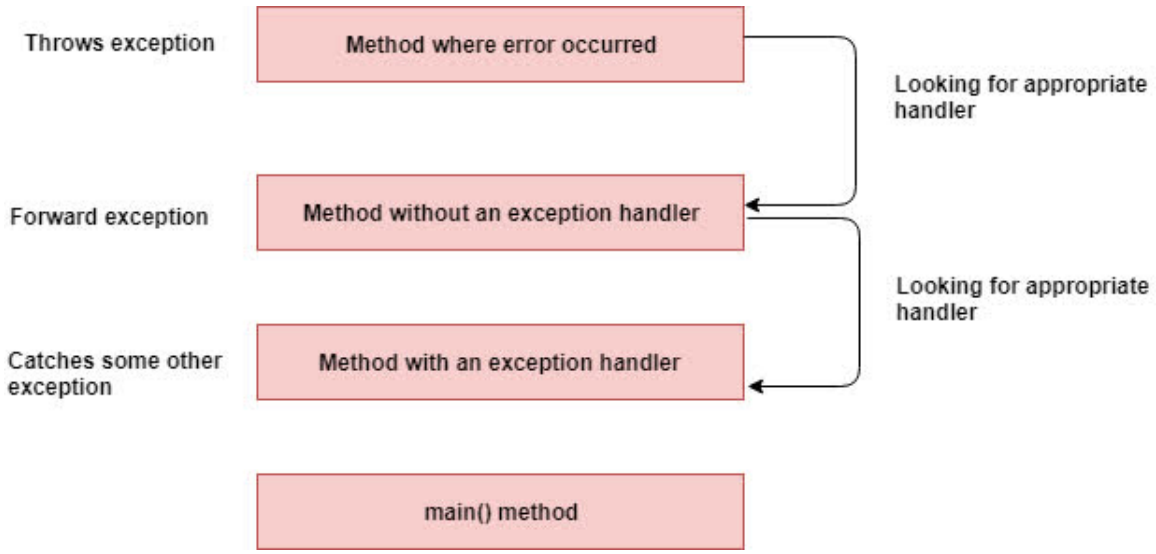

Figura 7.3: Tratamiento de la excepción lanzada[79]

Sin embargo, hay un número de prácticas independientes del lenguaje que ayudan a mantener un código limpio y a prueba de errores[77]:

Tener un estilo de código

Este punto ya se ha discutido en el capítulo 6, pero es esencial para los proyectos de mayor tamaño en el que trabajan centenares de programadores durante años desarrollar o utilizar un estilo de código estándar para reducir la dificultad de comprensión.

Del mismo modo, también puede resultar útil desarrollar una guía para el tratamiento de errores.

Utilizar tanto excepciones como códigos de estado

En algunas ocasiones se debe poder enviar el código de estado o la excepción.

Utilizar excepciones solo en casos extraordinarios

Se necesita una forma de distinguir los casos en los que usar estados y en los que usar excepciones. Una regla es «lanzar una excepción si y solo si no se puede hacer nada y el trabajo actual debe cancelarse por completo», pero puede haber otras.

Tener un conjunto de errores estándar y utilizarlo en todas partes

En la figura 7.4, puede verse un diagrama para decidir cuál es el estado o excepción apropiado para cada error que pueda surgir en la aplicación.

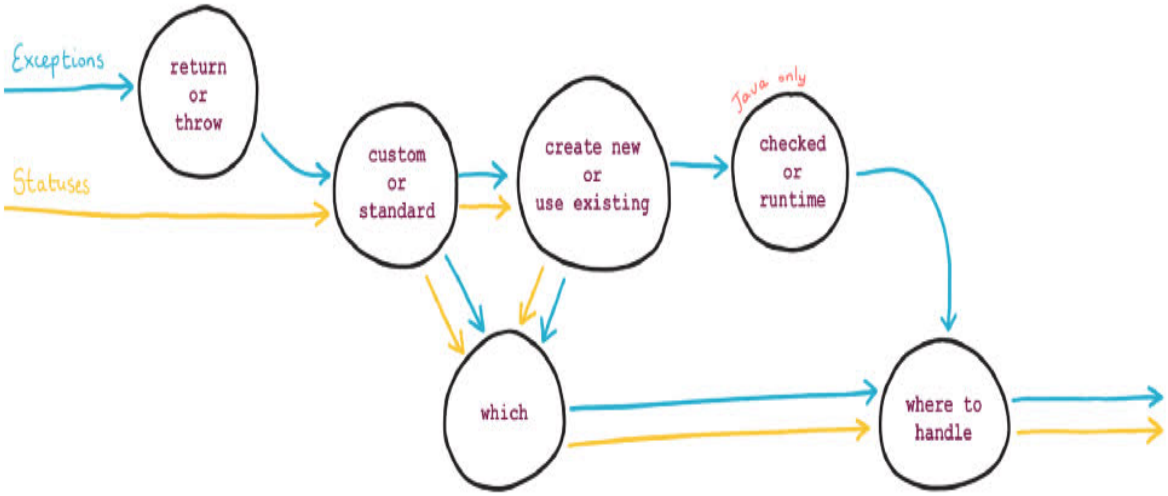

Figura 7.4: Preguntas que responder cuando tratar con errores[77]

Como puede verse también en la misma figura, utilizar códigos de estado o excepciones personalizados añade dificultad extra e innecesaria al sistema de tratamiento de errores.

En concreto, hay un conjunto de código de estado estándar que puede, y debe, utilizarse en las aplicaciones web, como la que se ha diseñado e implementado para el proyecto: los códigos de estado del protocolo HTTP que ya se han visto en la subsección 5.2.11.

No obstante, a veces estos no pueden dar la información suficiente para ser útiles. Por este motivo, la IETF (Internet Engineering Task Force) ha propuesto un segundo estándar para detallar los errores en el cuerpo de las respuestas HTTP[80].

El esquema que definen es utilizar un objeto JSON con los campos siguientes:

***type* (string)**

Una referencia URI que identifica el tipo de problema: tiene que proveer documentación leíble para un humano sobre el tipo concreto del problema.

***title* (string)**

Una descripción corta, leíble para humanos del tipo del problema. No debe de cambiar dependiendo de la ocurrencia del problema.

***status* (número)**

El código de estado HTTP.

***detail* (string)**
Una explicación leíble para un humano específica a la ocurrencia del problema.

***instance* (string)**
Una referencia URI que identifica la ocurrencia específica de la ocurrencia del problema.

Otros campos
Además, el desarrollador puede definir otros campos que sirvan para detallar el problema.

Un ejemplo de respuesta de error con el estándar anterior es:

```
HTTP/1.1 403 Forbidden
   Content-Type: application/problem+json
   Content-Language: en
   {
    "type": "https://example.com/probs/out-of-credit",
    "title": "You do not have enough credit.",
    "detail": "Your current balance is 30, but that costs 50.",
    "instance": "/account/12345/msgs/abc",
    "balance": 30,
    "accounts": ["/account/12345",
                 "/account/67890"]
   }
```

Sin embargo, el proyecto actual no se ha adherido a este último estándar de la IETF porque se ha considerado suficiente el conjunto de códigos de error estándar del protocolo HTTP.

Por último, los errores deben registrarse en el *log* para su estudio posterior y no deben ser retornados al usuario ya que, como se ha mencionado en la subsección 7.3.2, las excepciones pueden dar información técnica del sistema a un posible atacante en la fase de reconocimiento. Por este motivo, el mensaje de error devuelto debe ser genérico[78].

7.4. Almacenamiento de datos sensibles

7.4.1. ¿Qué es el almacenamiento de datos sensibles?

Los datos confidenciales como contraseñas, números de tarjetas de crédito, registros de salud, información personal y secretos comerciales requieren protección adicional, particularmente si esos datos están sujetos a las leyes de privacidad, como el Reglamento General de Protección de Datos de la Unión Europea (RGPD)[1]; reglas de protección de datos financieros, como el Estándar de Seguridad de Datos PCI (PCI DSS)[81]; u otras regulaciones[82].

7.4.2. ¿Por qué es importante el almacenamiento de datos sensibles?

Los atacantes pueden robar datos de la web y las aplicaciones de servicios web de varias formas. Por ejemplo, si se envía información confidencial a través de Internet sin seguridad en las comunicaciones, un atacante en una conexión inalámbrica compartida podría ver y

robar los datos de otro usuario. Asimismo, un atacante podría usar una inyección de código SQL para robar contraseñas y otras credenciales de una base de datos de aplicaciones y exponer esa información al público[82].

Además, el almacenamiento seguro de datos sensibles es un requisito legal en la actualidad y su incumplimiento puede acarrear multas millonarias. Varios artículos del RGPD[1] destacan la importancia de proteger los datos personales[9], en especial aquellos considerados sensibles, en reposo y durante el tratamiento[10]. En concreto:

- Define seis principios relativos al tratamiento de datos personales en el artículo 5, que puede encontrarse completo en el apéndice A.1. Estos principios son:
  - licitud, lealtad y transparencia;
  - limitación de la finalidad;
  - minimización de datos;
  - exactitud;
  - limitación del plazo de conservación;
  - e integridad y confidencialidad.

  Además, el responsable del tratamiento[11] debe garantizar y poder demostrar que los tratamientos realizados cumplen con los principios anteriores (responsabilidad proactiva)
- Define dos categorías especiales de datos en el artículo 9, que puede encontrarse completo en el apéndice A.2, en las que se debe tener particularmente cuidado:
  - Datos sobre el origen étnico o racial, opiniones políticas, convicciones religiosas o filosóficas, o la afiliación sindical.
  - Datos genéticos, datos biométricos dirigidos a identificar de forma unívoca a una persona física, datos relativos a la salud o datos relativos a la vida sexual o la orientación sexual de una persona física.
- Define varias medidas técnicas en el artículo 25 y en el 32, que pueden encontrarse en los apéndices A.3 y A.4 respectivamente, que el responsable del tratamiento puede aplicar para cumplir con su responsabilidad:
  - Protección de datos desde el diseño y por defecto.
  - Seudonimización.
  - K-anonimidad.
  - Minimización de datos.
  - Cifrado.

[9] Toda información sobre una persona física identificada o identificable.

[10] Cualquier operación o conjunto de operaciones realizadas sobre datos personales o conjuntos de datos personales.

[11] La persona física o jurídica que determine los fines y medios del tratamiento.

7.4.3. ¿Cómo puede mejorarse el almacenamiento de datos sensibles?

Para un almacenamiento de datos sensible seguro se tiene que tener en cuenta tres cosas[82]:

Clasificación de los datos

Es fundamental clasificar los datos en el sistema y determinar a qué nivel de sensibilidad pertenecen. Luego, cada categoría de datos se puede asignar a las reglas de protección necesarias para cada nivel de sensibilidad. Una clasificación de los niveles de sensibilidad habituales con ejemplos puede verse en la tabla 7.1.

Nivel	**Ejemplos**
Alto	Números de tarjetas de crédito (PCI) u otros números de cuentas financieras, datos personales de los usuarios, credenciales privilegiadas para sistemas de IT, información médica protegida, propiedad intelectual,...
Medio	Información de servicios de IT, registros de educación de estudiantes, información de sistemas de telecomunicaciones,...
Bajo	Contenido de sitios web públicos, comunicados de prensa, materiales de *marketing*,...

Cuadro 7.1: Clasificación del nivel de sensibilidad de los datos con ejemplos

Los datos de nivel de sensibilidad bajos pueden considerarse públicos y no es necesario ningún trabajo especial para almacenarlos en la base de datos. Sin embargo, sí se tienen que establecer ciertas reglas de protección para los niveles medio y alto.

Protección de los datos en tránsito

Los datos sensibles deben ser protegidos cuando están en tránsito por la red, ya sea pública o privada: el protocolo criptográfico para comunicaciones seguras más utilizado es el TLS por lo que la implementación de una configuración SSL/TLS y HTTPS segura evita la gran mayoría de problemas que puedan surgir cuando los datos están en tránsito.

Para las recomendaciones del autor en materia de la configuración SSL/TLS y HTTPS, puede referirse a la subsección 7.1.4 y a las implementaciones concretas en el capítulo 10.

Protección de los datos en reposo y durante el tratamiento

En la subsección anterior, se ha mencionado diversos principios o métodos técnicos y organizativos que pueden utilizarse para garantizar la protección de los datos y cumplir con la responsabilidad proactiva del responsable del tratamiento.

Sin embargo, dado su carácter jurídico y no técnico, el RGPD no da más que unas pinceladas de estas técnicas por lo que a continuación se han resumido algunas aproximaciones para su implementación de distintas guías de la AEPD y otros sitos. Concretamente:

Protección de los datos desde el diseño y por defecto

Hay ocho estrategias de diseño de la privacidad que deben seguirse para garantizar la seguridad de los datos desde el diseño y por defecto y que así el responsable del tratamiento cumpla así con su «responsabilidad proactiva».

Estrategias orientadas a datos

Las cuatro estrategias orientadas a datos son:

Minimizar

Debe recogerse y tratar la mínima cantidad de datos posible para así limitar los posibles riesgos en la privacidad.

Ocultar

Debe limitarse la exposición de los datos, estableciendo las medidas necesarias para garantizar la confidencialidad y la desvinculación de los datos.

Separar

Debe evitarse el riesgo que durante el procesamiento de diferentes datos personales pertenecientes a un mismo individuo y utilizados en tratamientos independientes, se pueda realizar un perfilado completo del sujeto.

Abstraer

Debe limitarse al máximo el detalle de los datos personales que son tratados.

Estrategias orientadas a procesos

Las cuatro estrategias orientadas a procesos son:

Informar

Debe informarse a los usuarios del procesamiento de sus datos: qué información se procesa, con qué propósito, a quién se comunica y el plazo de conservación de los datos. El consentimiento del usuario debe ser libre, específico, informado e inequívoco. Además, cualquier modificación debe ser comunicada, incluyendo las posibles brechas de seguridad.

Controlar

Debe implementarse mecanismos que permitan al usuario el ejercicio de sus derechos: acceso, rectificación, supresión, oposición, portabilidad y limitación al tratamiento. Así como la prestación y la retirada del consentimiento.

Cumplir

Debe definirse un marco de privacidad y una estructura de organización involucrada desde la alta dirección, así como los roles y responsables que velen por su cumplimiento.

Demostrar

El responsable del tratamiento ha de poder demostrar, tanto a los usuarios como a las autoridades de supervisión, el cumplimiento de la política de protección de datos que está aplicando, así como del resto de requisitos y obligaciones legales del RGPD.

La información anterior se ha obtenido de los sitios siguientes:

- *Guía de Privacidad desde el Diseño*[83].
- *Guía de Protección de Datos por Defecto*[84].

Seudonimización

La seudonimización es el tratamiento de datos personales de tal manera que ya no puedan atribuirse a un usuario concreto sin utilizar información adicional: los posibles campos que puedan identificar al usuario se sustituyen por seudónimos o identificadores.

La seudonimización, por lo tanto, reduce significativamente el peligro de posibles brechas de seguridad ya que la información del usuario no puede asociarse fácilmente a un individuo concreto.

Sin embargo, se deben tomar las medidas apropiadas para que solo los responsables y encargados del tratamiento autorizados puedan revertir la seudonimización ya que la seudonimización no exime de las otras obligaciones legales del RGPD: los datos seudonimizados aún son considerados personales.

Hay múltiples técnicas que pueden utilizarse para implementar la seudonimización: desde utilizar un identificador único incremental, aleatorio, generado por una función *hash* criptográfica o por código de autenticación de mensaje (MAC) o simplemente cifrando la información de identificación del usuario.

Su implementación correcta puede resultar compleja y su explicación larga, por lo que los interesados en las técnicas mencionadas anteriormente pueden consultar los dos documentos siguientes:

- *Introducción al hash como técnica de seudonimización de datos personales*[31].
- *Pseudonymisation techniques and best practices*[85].

***K*-anonimidad**

La K-anonimidad es una propiedad de los datos anonimizados que permite cuantificar hasta qué punto se preserva la anonimidad de los sujetos presentes en un conjunto de datos en el que se han eliminado los identificadores.

Los atributos de los registros pueden clasificarse según su naturaleza:

- Atributos clave o identificadores: identifican unívocamente a los sujetos de los datos. Deben ser eliminados de los registros anonimizados.
- Cuasi-identificadores: si bien por si mismos y de forma aislada no identifican a un individuo, agrupados con otros atributos «cuasi-identificadores» pueden señalar de forma unívoca a un sujeto. Este es el tipo de datos que las técnicas de anonimización eliminan, agregan o generalizan.
- Atributos sensibles: los datos que podrían tener un mayor impacto en la privacidad de un individuo concreto.

Un individuo es k-anónimo dentro del conjunto de datos en el que se encuentra incluido si, y sólo si, para cualquier combinación de los atributos cuasi-identificadores asociados, existen al menos otros K-1 individuos que comparten con él los mismos valores para esos mismos atributos.

Existen dos métodos para implementar la K-anonimización que no perturban los datos: la generalización y la eliminación.

Generalización
: Consiste en hacer que el valor de los atributos cuasi-identificadores sea menos precioso, transformándolos o generalizándolos dentro de un conjunto o intervalo que comparte los mismos valores.

Eliminación
: Aquellos registros con valores muy poco usuales deben ser eliminados dado que aumentan significativamente la probabilidad de reidentificación.

La información anterior se ha obtenido del siguiente sitio:

- *La K-anonimidad como medida de la privacidad*[86].

Aquellos interesados en el tema también pueden consultar:

- *10 malentendidos relacionados con la anonimización*[87].

Cifrado
: El cifrado es una medida técnica apropiada para la protección de los datos personales y así lo recomienda el RGPD. Sin embargo, antes de usar un algoritmo de cifrado y un tamaño de clave concretos se debe tener en cuenta si estos aún son considerados seguros porque los algoritmos más antiguas pueden tener vulnerabilidades conocidas o los tamaños de clave más pequeñas pueden romperse con ataques de fuerza bruta.

  En concreto, los algoritmos de cifrado aprobados por NIST para el uso del gobierno y las instituciones norteamericanas son[88]:

  - AES (Advanced Encryption Standard): un algoritmo de cifrado de clave simétrica que utiliza claves de tamaño 128, 192 o 256 bits.

  - Triple-DES (Triple Data Encryption Standard): un algoritmo de cifrado de clave simétrica que utiliza tres claves distintas para cifrar el mismo bloque de 64 bytes. Esta particularidad ocasiona que el algoritmo pierda efectividad respecto al tamaño de la clave, que puede ser de 112 o 168 bits.

  De ambos, la mejor opción es AES: no se ha encontrado todavía una vulnerabilidad suya y los distintos tamaños de clave proporcionan una seguridad más que razonable, incluso cuando la clave es «solo» de 128 bits.

  Sin embargo, la utilización de esta técnica para la protección de los datos personales añade las cuestiones de cómo generar, almacenar, utilizar y destruir las claves de forma segura porque la generación de una clave débil y su uso posterior compromete significativamente el sistema; y el robo de la clave permite a los atacantes descifrar todos los datos personales cifrados.

  Para las recomendaciones de NIST de este tema se puede consultar la página web *Key Management | CSRC*[89] que contiene distintos enlaces a documentos sobre la gestión de claves.

7.5. Validación de las entradas

7.5.1. ¿Qué es la validación de las entradas?

La validación de las entradas es el proceso de probar los datos recibidos por la aplicación para verificar su cumplimiento con un estándar definido dentro de la aplicación.

Puede ser tan simple como escribir estrictamente un parámetro y tan complejo como usar expresiones regulares o lógica empresarial para validar la entrada.

Hay dos tipos diferentes de enfoques de validación de entrada: validación de lista blanca (a veces denominada validación de inclusión o positiva) y validación de lista negra (a veces conocida como validación de exclusión o negativa)[90].

7.5.2. ¿Por qué es importante la validación de las entradas?

La validación de entrada se realiza para garantizar que solo los datos correctamente formados ingresen en el flujo de trabajo en un sistema de información, evitando que los datos mal formados persistan en la base de datos y provoquen un mal funcionamiento en los componentes posteriores[91].

Asimismo, la validación de las entradas puede[92]:

- Reducir la superficie de ataque de las aplicaciones y dificultar los ataques contra una aplicación.
- Proporcionar seguridad a determinadas formas de datos, específicas de determinados ataques.

Sin embargo, no debe usarse como el método principal para prevenir ataques XSS, explicado en la sección 8.4, inyección de código, explicado en la sección 8.5, u otros tipo de ataque, aunque sí que puede contribuir a su prevención con la implementación correcta.

7.5.3. ¿Cómo puede mejorarse la validación de las entradas?

La validación de las entradas debe aplicarse tanto en el nivel sintáctico como en el semántico.

- Validación sintáctica: la entrada está en la forma esperada.
- Validación semántica: la entrada está dentro del rango aceptable para la funcionalidad de la aplicación y el contexto

Tal y como ya se ha comentado, hay dos aproximaciones a la validación de las entradas:

- Lista de exclusión: la validación de la entrada se hace detectando caracteres o patrones posiblemente peligros.
- Lista de inclusión: la validación de la entrada intenta comprobar que los datos de entrada coinciden con un conjunto de reglas.

Como mínimo debería implementarse la lista de inclusión cuando se desarrolla *software* seguro: la lista de inclusión ayuda a limitar la superficie del ataque al asegurar la corrección

de la validez sintáctica y semántica. Por otro lado, la lista de exclusión ayuda a detectar y detener ataques obvios.

Una forma de implementar la lista de inclusión es utilizando expresiones regulares para validar las entradas, pero debe de tenerse cuidado a la hora de diseñarlas y añadirlas a la aplicación porque pueden ser vulnerables al ataque de denegación de servicio ReDoS[93].

Para evitar esta vulnerabilidad, la mejor forma de realizar la validación de las entradas por inclusión es utilizar librerías de validación que cubran los tipos básicos de datos o que permitan la definición y validación de «JSON Schemas»[12], como pueden ser «joi» o «validatorjs» para JavaScript.

Un caso muy especial es la validación de las direcciones de correo electrónico ya que su validación sintáctica es muy difícil, especialmente si se trata de hacer con una expresión regular[95].

Por este motivo, la mejor forma de validar un correo electrónico es con una validación inicial simple y enviar un correo de confirmación con un enlace o un *token* para confirmar la propiedad.

Asimismo, para la mitigación de los ataques de inyección de código, como puede ser el ataque XSS, explicado en la sección 8.4, una parte fundamental de la validación de las entradas es la codificación y el escape de los caracteres.

Por último, la validación de las entradas siempre debe hacerse en el lado del servidor, independientemente de que también se haga en el lado del cliente, ya que un atacante puede sobrepasarla y realizar la petición directamente al servidor web.

La información anterior se ha obtenido de los sitios siguientes:

- *C5: Validate All Inputs | OWASP*[92].
- *Input Validation | OWASP Cheat Sheet Series*[91].

[12] Json Schema es un vocabulario que permite anotar y validar documentos JSON[94].

Capítulo 8

Recopilación de ciberataques: qué son, cómo se producen y cómo protegerse

8.1. Ataque DoS

8.1.1. ¿Qué es un ataque DoS?

El ataque de denegación de servicio (DoS por sus siglas en inglés: «Denial-of-Service») es uno de los ataques informáticos más comunes en la actualidad[96]; aunque habitualmente se realiza de forma distribuida (DDoS: «Distributed Denial-of-Service») en el que el ataque se lanza desde múltiples dispositivos.

Un ataque de denegación de servicio es un intento malintencionado de afectar la disponibilidad del sistema atacado, como por ejemplo, un sitio web o una aplicación, para usuarios legítimos. Los atacantes suelen generar grandes volúmenes de paquetes o requerimientos para sobrecargar el sistema objetivo[97].

8.1.2. ¿Cómo se produce un ataque DoS?

Los vectores de ataques de un posible ataque de denegación de servicio son múltiple: se puede realizar sobre diferentes capas del modelo OSI, como puede verse en la tabla 8.1[97].

	Capa	**Vector de ataque**
7	Aplicación	Inundación HTTP[1], inundación DNS[2], ReDoS[3]
6	Presentación	Abuso SSL[4]
5	Sesión	N/A
4	Transporte	Inundación SYN[5]
3	Red	Inundación ICMP[6], ataques de reflexión UDP[7]
2	Enlace de datos	N/A
1	Física	N/A

Cuadro 8.1: Modelo OSI con los vectores de ataque para realizar un ataque de denegación de servicio

Los más comunes son los ataques en las capas tres y cuatro y se consideran ataques en la capa de infraestructura, pero también son los más fáciles de detectar. Sin embargo, los ataques a las capas seis y siete se consideran ataques a la capa de aplicación son más sofisticados y tienden a concentrarse en partes particularmente «caras» de la aplicación.

Cuando el ataque es realizado de forma distribuida, los atacantes se aprovechan de una red de *bots*[8] para conectarse al mismo tiempo al servidor que quieren colapsar. A veces, utilizan los aparatos del Internet de las Cosas (IoT) con conexión a Internet y, a menudo, una seguridad débil o inexistente[98]. Este gran número de dispositivos a su disposición les permite lanzar ataques con un gran volumen de información. En 2020, AWS Shield registró un ataque de denegación de servicio de 2.3Tbps[99].

Asimismo, también hay ataques de denegación de servicio multivectoriales: el atacante utiliza varias rutas para sobrecargar el servidor de destino, para así burlar o desviar los esfuerzos de mitigación[100].

8.1.3. ¿Cómo protegerse de un ataque DoS?

La preocupación principal al mitigar un ataque DoS es diferenciar entre el tráfico de ataque y el tráfico normal, pero esta es una tarea muy difícil: cuanto más complejo es el ataque, más difícil es distinguir el tráfico normal del malicioso ya que el objetivo del atacante es disimularlo tanto como hacerla posible y hacer la mitigación ineficaz. Además, el ataque también se puede modificar y adaptar para contrarrestar las medidas de protección tomadas[100].

La mayoría de las posibles técnicas de mitigación DoS quedan fuera del objetivo de este trabajo porque han de implantarse después del desarrollo de la aplicación web. Igualmente, se explican la idea básica a continuación:

Enrutar a agujeros negros
: Una posible solución es crear una ruta de agujero negro y dirigir el tráfico hacia ella. Sin embargo, si la discriminación entre el tráfico de ataque y el tráfico legítimo no es la correcta, o inexistente, el atacante consigue su objetivo: la red se vuelve inaccesible para todo el mundo[101].

Balancear la carga
: Un *proxy* inverso[9] actúa como un equilibrador de carga para enrutar las solicitudes

[1]Los atacantes «inundan» de peticiones HTTP al servidor objetivo.

[2]Los atacantes «inundan» los servidores de un dominio en concreto con resoluciones DNS.

[3]El ataque de denegación de servicio de expresión regular es un ataque que aprovecha el hecho de que la mayoría de las implementaciones de expresiones regulares pueden llegar a situaciones extremas que hacen que funcionen muy lentamente en función del tamaño de la entrada[93].

[4]El protocolo TLS permite a los clientes renegociar ciertos aspectos del TLS, lo que puede provocar un abuso por parte de los atacantes. El protocolo TLS 1.3 ya no lo permite.

[5]SYN es el primero de los tres pasos en la negociación de la conexión del protocolo TCP.

[6]Se basa en el envío masivo de mensajes *ping* del protocolo ICMP al ordenador objetivo.

[7]El atacante modifica la IP de origen de un datagrama UDP y lo envía a un servidor de intermediario que responde a la víctima.

[8]Dispositivos infectados previamente por algún tipo de *malware* que permite su control remoto.

[9]Un *proxy* inverso es un servidor que se encuentra frente a los servidores web y reenvía las solicitudes del cliente a esos servidores web. Los *proxies* inversos generalmente se implementan para mejorar la seguridad, el rendimiento y la fiabilidad[102].

de los clientes a diferentes servidores para maximizar la velocidad y la utilización de la capacidad y así garantizar que ningún servidor esté sobrecargado de trabajo.

Si un solo servidor deja de funcionar, el equilibrador de carga redirige el tráfico a los servidores en línea restantes. Cuando se agrega un nuevo servidor al grupo de servidores, el equilibrador de carga comienza a enviarle solicitudes automáticamente[103].

Limitar la frecuencia

Otra solución es limitar la cantidad de solicitudes que el servidor HTTP puede aceptar de una dirección IP durante un periodo determinado de tiempo. Si bien, es probable que por sí sola no sea suficiente para controlar de forma efectiva un ataque DDoS complejo[100].

Esta estrategia, además, puede proteger contra los intentos de inicio de sesión por fuerza bruta, explicados en la sección 8.2 de este mismo capítulo.

Reducir el área de superficie de ataque

Al reducir el área de superficie de ataque, se limitan las opciones del atacante lo que permite regular y monitorear los vectores de ataque restantes para mejorar la seguridad. Las aplicaciones no deben exponerse a puertos, protocolos u otras aplicaciones desde donde no esperan ninguna comunicación[104].

Varios ejemplos son:

- Deshabilitar la renegociación del cliente del protocolo TLS.
- Deshabilitar la funcionalidad ICMP.

***Firewall* de aplicaciones web**

Un *firewall* de aplicaciones web (WAF) es una herramienta que puede ayudar a mitigar ataques DoS: filtra y monitorea el tráfico HTTP que recibe la aplicación web al actuar como un *proxy* inverso.

Un WAF opera a través de un conjunto de reglas denominadas políticas. Estas políticas tienen como objetivo proteger la aplicación al filtrar el tráfico malicioso. El valor de un WAF proviene en parte de la velocidad y la facilidad con la que se modificar estas políticas. Esto permite una respuesta más rápida a los diferentes vectores de un ataque DDoS. La limitación de velocidad, por ejemplo, se puede implementar modificando las políticas WAF[105].

Asimismo, un WAF puede proteger contra otros tipos de ciberataques como: CSRF, explicado en la sección 8.3; XSS, explicado en la sección 8.4; o inyección de código, explicado en la sección 8.5.

Distribución de red Anycast

Anycast es un método de direccionamiento y enrutamiento de red en el que las solicitudes entrantes se pueden enrutar a una variedad de ubicaciones diferentes.

En el contexto de una CDN[10], Anycast generalmente enruta el tráfico entrante al centro de datos más cercano con la capacidad de procesar la solicitud de manera eficiente.

[10]Una red de entrega de contenido (CDN) está formada por un grupo de servidores distribuidos geográficamente que trabajan juntos para ofrecer una entrega rápida de contenido de Internet[106].

El enrutamiento selectivo permite que una red Anycast sea resistente frente al alto volumen de tráfico, la congestión de la red y los ataques DDoS[107].

Sin embargo, estos métodos de mitigación no son excluyentes entre ellos, por lo que una defensa efectiva los implantaría la mayoría, sino todos. Esta puede ser una tarea colosal para un equipo pequeño de desarrolladores por lo que la mejor opción es dejar este trabajo a los expertos y contratar una de las empresas que ofrecen este servicio como Cloudflare o Amazon Web Services.

8.2. Ataque de credenciales

8.2.1. ¿Qué es un ataque de credenciales?

El objetivo del ataque de credenciales es la obtención ilegal de las contraseñas de una organización o un individuo con la intención de acceder y abusar del acceso o conseguir datos e información críticos.

Asimismo, es, a menudo, una etapa temprana de un ataque cibernético: el robo de credenciales permite a los atacantes operar sin ser detectados en una red, restablecer contraseñas y causar estragos dentro de una organización[108].

8.2.2. ¿Cómo se produce un ataque de credenciales?

Los ataques de credenciales pueden producirse de diferentes formas[109], como puede verse en la tabla 8.2 con una breve descripción de la estrategia o estrategias que el atacante puede siguir.

Ataque	**Descripción**
Fuerza bruta	Prueba sistemáticamente todas las contraseñas posibles, palabras de diccionario con reglas de modificación (añadir números o sustituir letras) o *hash cracking*
Credential Stuffing	Prueba de pares de nombre de usuario / contraseña obtenidos de las brechas de seguridad de otros sitios
Password Spraying	Prueba de una única contraseña débil contra una gran cantidad de cuentas diferentes
Phishing	Se consigue el nombre de usuario y la contraseña haciéndose pasar por una persona, empresa o servicio de confianza
Registrador de teclas	Se consigue el nombre de usuario y la contraseña con un *malware* que registra las pulsaciones realizadas en el teclado

Cuadro 8.2: Vectores de ataque para la realización de un ataque de credenciales

8.2.3. ¿Cómo protegerse de un ataque de credenciales?

La mejor defensa posible contra este tipo de ataques es la implementación de la Autenticación Multifactor (MFA). Un buen sistema de MFA exige dos o más de los siguientes

métodos de autenticación[110]:

- Algo que conoce, normalmente una contraseña.
- Algo que tiene, como un dispositivo de confianza.
- Algo que forma parte del usuario, como la huella dactilar o un reconocimiento facial.

La correcta implementación de un sistema MFA es muy compleja y seguramente más allá de las capacidades de un equipo pequeño por lo que la aproximación habitual es la autenticación en dos pasos (2FA): después de la autenticación habitual del usuario con su nombre de usuario y la contraseña, el usuario recibe un código adicional vía dirección de correo electrónico, SMS o llamada telefónica para finalizar el inicio de sesión.

No obstante, para evitar el hastío del usuario y que acabe desactivando la 2FA o la MFA, es recomendable solicitarla solo en aquellos casos que la autenticación es sospechosa como:

- Un nuevo navegador, dispositivo o dirección IP.
- Un país o ubicación inusual.
- Una dirección IP que aparece en listas de bloqueo conocidas.
- Una dirección IP que ha intentado iniciar sesión en varias cuentas.

Además, esta no debe ser la única estrategia que se utilice para proteger las cuentas del usuario: no siempre es posible de implementar o los usuarios de la aplicación no la habilitan, ya sea porque no quieren compartir su número de teléfono o, simplemente, porque no conocen sus beneficios. Menos del 10 % de los usuarios de Microsoft utilizan MFA[111].

Por este motivo, existen otras defensas alternativas, y aunque individualmente no son tan efectivas como la 2FA o la MFA, si se implementan múltiples defensas por capas pueden proporcionar un grado de protección aceptable para los ataques de fuerza bruta, *credential stuffing* y *password spraying*.

No obstante, las estrategias siguientes no funcionan cuando el ataque se realiza con *phishing* o con un registrador de teclas porque el usuario proporciona al atacante su nombre de usuario y contraseña.

CAPTCHA

Requerir al usuario de resolver un CAPTCHA por cada intento de autenticación ayuda a prevenir ataques de credenciales automatizados, lo que reduce la velocidad de los ataques de *credential stuffing* o de *password spraying*.

Sin embargo, esta estrategia no es tan efectiva como pueda parecer en el caso de *password spraying* ya que es un ataque que se caracteriza por su lentitud: cuanto más rápidos son los intentos de acceso, más fácil son de detectar en las otras capas de las defensas[109].

Además, para mejorar la usabilidad, se puede requerir el CAPTCHA solo en aquellos casos considerados sospechosos, del mismo modo que con 2FA o MFA.

Bloqueo de IPs y cuentas

Debe ser posible detectar al menos dos escenarios:

- Intentos fallidos por cuenta: protección frente ataques de fuerza bruta.
- Intentos fallidos por IP: protección frente ataques menos sofisticados de *credential stuffing* y *password spraying*.

Cuando se detecta uno o ambos casos, debe bloquearse la cuenta y/o la IP, aunque el bloqueo automático es recomendable que sea temporal para evitar bloquear permanentemente usuarios legítimos.

Un sitio web interesante es *AbuseIPDB - IP addresses abuse reports - Making the Internet safer, one IP at a time*[112] en el que se reportan miles de direcciones IP sospechosas cada hora. Además cuenta con una API pública para reportar y comprobar si una IP ha sido ya reportada como maliciosa para bloquearla permanentemente o no.

Identificar contrseñas vulnerables

El ataque de *credentials stuffing* abusa de pares de nombre de usuario y contraseñas que se han obtenido de brechas de seguridad anteriores y el ataque de *password spraying* prueba las contraseñas más comunes.

Por este motivo, una buena práctica para mitigar el riesgo de ambos ataques es utilizar herramientas como *Have I Been Pwned: Pwned Passwords*[113] para identificar las contraseñas vulnerables y advertir a los usuarios; y prohibir las contraseñas más comunes. Por ejemplo, las contraseñas más comunes de 2021 son[114]:

- 123456
- 123456789
- qwerty
- password
- 12345
- qwerty123
- 1q2w3e
- 12345678
- 111111
- 1234567890

Notificar a los usuarios sobre eventos de seguridad inusuales

Cuando se detectan actividades inusuales o sospechosas, puede ser apropiado advertir al usuario. Además, también se debería mostrar información relativa a las autenticaciones actuales y anteriores.

Una fuente fundamental para la información anterior es *Credential Stuffing Prevention - OWASP Cheat Sheet Series*[115].

8.3. Ataque CSRF

8.3.1. ¿Qué es un ataque CSRF?

El ataque de falsificación de petición en sitios cruzados (CSRF, por las siglas en inglés de *Cross-site request forgery* y conocido también como XSRF) es un ataque que obliga a un usuario final a ejecutar acciones no deseadas en una aplicación web en la que está autenticado actualmente.

Con un poco de ayuda de la ingeniería social (como enviar un enlace por correo electrónico o chat), un atacante puede engañar a los usuarios de una aplicación web para que ejecuten acciones de su elección.

Si la víctima es un usuario normal, un ataque CSRF exitoso puede obligar al usuario a realizar solicitudes de cambio de estado, como transferir fondos, cambiar su dirección de correo electrónico,... Si la víctima es una cuenta administrativa, CSRF puede comprometer toda la aplicación web[116].

8.3.2. ¿Cómo se produce un ataque CSRF?

Este tipo de ataques son posibles porque los navegadores web envían automáticamente ciertos tipos de autenticación con cada petición a los sitios web. Por ejemplo, cuando la autenticación se hace a través de *cookies*[117]. Esto se debe a que:

- Los navegadores web guardan las *cookies* emitidas por los sitios web.
- Las *cookies* almacenadas incluyen *cookies* de sesión para usuarios autenticados.
- Los navegadores envían todas las *cookies* asociadas al dominio del sitio web sin importar de donde se ha generado la petición.

Además también afecta a los tipos de autenticación «Basic» y «Digest». La autenticación «Basic» consiste en que el cliente envía una petición HTTP con el encabezado de «Authorization» que contiene la palabra «Basic» seguida por un espacio y la codificación en base 64 del *string* «username:password». La autenticación «Digest» difiere de «Basic» en que a la contraseña se le aplica una función de *hash* antes de ser enviada.

El flujo de un ataque CSRF habitual es:

1. Un usuario se autentica en el sitio web legítimo: «good-banking-site.com». El usuario es autenticado por el servidor web y emite una *cookie* de sesión. Además, el servidor web confía en todas las peticiones que incluyan dicha *cookie*.
2. El usuario visita el sitio malicioso que incluye, por ejemplo, el código HTML siguiente:

```
<h1>Congratulations! You're a Winner!</h1>
<form action="http://good-banking-site.com/api/account" method="post">
    <input type="hidden" name="Transaction" value="withdraw">
    <input type="hidden" name="Amount" value="1000000">
    <input type="submit" value="Click to collect your prize!">
</form>
```

3. El usuario presiona el botón de enviar del formulario y el sitio web malicioso realiza la petición HTTP en nombre del usuario legítimo con la *cookie* de autenticación correcta.

4. El sitio web legítimo autentica la petición del atacante y realiza la petición solicitada.

Un segundo tipo de ataque CSRF es el «Login CSRF» en el que el atacante fuerza al usuario a autenticarse en la cuenta del atacante para así obtener información privada del usuario legítimo como el historial de actividad o datos de pago que haya podido usar.

8.3.3. ¿Cómo protegerse de un ataque CSRF?

Los tipos de autenticación vulnerables al ataque CSRF son, tal y como se ha comentado en la subsección anterior, «Autenticación basada en *cookie* de sesión», «Autenticación Basic» y «Autenticación Digest» por lo que la mejor forma de evitar esta vulnerabilidad es utilizar una «Autenticación basada en *token*», siendo el Json Web Token (JWT) el más famoso de este tipo, porque, una vez el *token* es enviado al cliente en un encabezado de la respuesta, este se guarda en el almacén local del navegador[117].

Sin embargo, en aquellos casos en que esta no sea opción o que se considere inapropiada para la aplicación en cuestión, se pueden tomar otras medidas para evitar, o al menos mitigar, los ataques CSRF[116].

Los dos siguientes métodos de protección previenen, si se han implementado de forma correcta, la vulnerabilidad de la aplicación a los ataques CSRF, si bien es recomendable buscar que opciones de seguridad ofrece el *framework* en el que se está desarrollando la aplicación ya que es muy probable que ya incorpore, de un modo u otro, la protección CSRF.

Por ejemplo, algunos *frameworks* que han implementado la protección frente ataques de CSRF son:

- Laravel. Utiliza *tokens* de sincronización[118].
- Django. Utiliza el doble envío de *cookies*[119], aunque el valor aleatorio y la *cookie* se generan en el lado del servidor.
- ASP.NET Core. Utiliza un híbrido entre las dos técnicas anteriores[117][120].
- NodeJS con el paquete «csurf». Depende de si se utilizan *cookies* o sesión[121].

***Tokens* de sincronización**

Un *token* de CSRF es un valor único, secreto e impredecible que genera el lado del servidor de la aplicación y se transmite al cliente para que que se incluya en una petición HTTP posterior. Luego, el servidor web valida que la petición lo incluye y rechaza la solicitud si el *token* falta o es inválido[122].

Estos *tokens* deben generarse por petición o por sesión. Si bien, por temas de seguridad es preferible hacerlo por petición, ya que el tiempo de abusar de un *token* robado es mucho menor, esto puede provocar problemas de usabilidad.

Por ejemplo, en algunos casos se puede inutilizar el botón de «Atrás» del navegador ya que la página anterior puede contener un *token* de CSRF inválido.

Asimismo, los *tokens* de CSRF no deben transmitirse dentro de una *cookie* ya que entonces la aplicación volvería a ser vulnerable. Por este motivo, se debe hacer con encabezados, parámetros o campos ocultos de los formularios. No obstante, entonces debe tenerse cuidado de no dejar rastro del *token* en los registros de la aplicación o en las direcciones URL.

Para protegerse de ataques de «Login CSRF» con esta técnica, se deben utilizar presesiones para usuarios no autenticados. Sin embargo, las presesiones deben eliminarse una vez los usuarios se han autenticado y crear una de nueva para evitar otro tipo de vulnerabilidad: fijación de sesión[11].

Cookies de doble envío

Una alternativa para evitar guardar la información de los *tokens* en el lado del servidor es utilizar *cookies* de doble envío.

En esta técnica, el lado del cliente de la aplicación genera un valor aleatorio criptográficamente fuerte y lo guarda como una *cookie* en el navegador del usuario separada de la *cookie* de sesión: se hace en el cliente para prevenir el ataque de «login CSRF».

Después, al realizar una petición HTTP, el lado del cliente envía la dos *cookies* al servidor web y el valor generado como un encabezado, un parámetro o un campo oculto de un formulario. El servidor entonces comprueba que los dos valores coincidan.

Sin embargo, esta técnica solo funciona completamente si el servidor web, y sus subdominios, solo aceptan conexiones HTTPS[123].

Además, la *cookie* puede encriptarse o firmarse con un HMAC[12] para que el atacante no pueda recrear el valor de la *cookie*.

Además, también son posibles las siguientes mitigaciones, aunque un ataque XSS, explicado en la sección 8.4 de este mismo capítulo, las inutilizaría:

Atributo de _cookies_ «SameSite»

«SameSite» es un atributo de las *cookies* que ayuda al navegador a decidir si tiene que enviar la *cookie* en una petición en sitios cruzados[125].

Si el valor es «Strict», la *cookie* no se enviará en ningún caso. En cambio, si el valor es «Lax» entonces se enviará en aquellos métodos HTTP considerados seguros, como GET y HEAD. Aquellas *cookies* en que el valor de «SameSite» sea «None» siempre serán enviadas, sin importar de dónde se origine la petición o qué método sea.

Lo más recomendable es que el atributo de «SameSite» de las *cookies* sea al menos «Lax» para garantizar un mínimo de seguridad a los usuarios.

Confirmación del usuario de peticiones potencialmente peligrosas

Del mismo modo que las autenticaciones sospechosas pueden verificarse con el uso de 2FA, MFA o CAPTCHA, el usuario también debe poder confirmar peticiones potencialmente peligrosas, como un cambio de contraseña o una transferencia de

[11] El atacante obliga al usuario a utilizar una *cookie* de sesión con un identificador conocido para que cuando la próxima vez que el usuario se autentique en la aplicación web se asocie esta a su sesión.

[12] Un código de autentificación de mensajes en clave-hash (HMAC) es una construcción específica para calcular un código de autentificación de mensaje (MAC) que implica una función *hash* criptográfica en combinación con una llave criptogràfica secreta[124].

dinero, y así prevenir operaciones no autorizadas. Algunas técnicas que pueden utilizarse son:

- Entrar de nuevo la contraseña o un código secundario.
- *Token* de un solo uso.
- Además de los ya mencionadas: 2FA, MFA o CAPTCHA.

Utilización de encabezados HTTP personalizados

Una defensa alternativa a los *tokens* de CSRF y al envío doble de *cookies*, es utilizar encabezados HTTP personalizados. Esta técnica es realmente efectiva para AJAX o los *endpoints* de una API.

Esto se debe a que los navegadores web no permiten realizar peticiones entre sitios cruzados con encabezados personalizados: así se añade protección frente ataques CSRF sin hacer cambios en la interfaz de usuario y sin que el servidor guarde los *tokens* generados.

Verificación del origen de la petición

Se examinan los encabezados HTTP de la petición para determinar que el origen y el objetivo de la solicitud[29].

- El origen de la solicitud puede encontrarse en los encabezados HTTP «Origin» o «Referer».
- El objetivo de la solicitud es un poco más complicado. Puede configurarse la aplicación para que simplemente lo sepa o utilizar en encabezado de «Host», cuando no se está detrás de un *proxy*, o el encabezado de «X-Forwarded-Host», cuando si se está detrás de un *proxy*.

Sin embargo, a menudo los encabezados de «Origin» o «Referer» no están incluidos por temas de privacidad, por lo que está mitigación no siempre es posible.

8.4. Ataque XSS

8.4.1. ¿Qué es un ataque XSS?

El ataque de secuencia de comandos en sitios cruzados (XSS por sus siglas en inglés de «Cross-Site Scripting») es un tipo de ataque de inyección de código porque el atacante inserta código malicioso en forma de *scripts* en sitios webs legítimos.

El navegador que no tiene forma de saber que el *script* no es de confianza, le da acceso a las *cookies*, *tokens* de sesión y otra información sensible. El atacante puede incluso modificar el contenido HTML de la página[126].

Típicamente el objetivo de este ataque es el robo de credenciales, sesiones o cuentas, sobrepasar sistemas de autenticación multifactor o realizar otros ataques contra el navegador del usuario como descargas de *software* malicioso, registradores de teclas, minado de criptomonedas,...

Además, es una vulnerabilidad muy extendida y fácil de explotar: puede encontrarse en dos tercios de todas las aplicaciones web y existen herramientas automáticas para detectar y abusar de ella[127].

8.4.2. ¿Cómo se produce un ataque XSS?

Los ataques XSS ocurren cuando la aplicación web acepta datos de fuentes sin verificar. Después, esta información es incluida como contenido dinámico en el cliente sin ser tratada previamente.

Las dos formas más conocidas de realizar un ataque XSS son:

Ataque XSS Reflected

Los ataques XSS Reflected inyectan temporalmente un *script* que luego es reflejado por el servidor web, como en un mensaje de error, resultado de búsqueda o cualquier otra respuesta que incluya parte o toda la entrada enviada al servidor como parte de la solicitud.

Los ataques XSS Reflected se envían a las víctimas a través de otra ruta, como en un mensaje de correo electrónico o en otro sitio web. Cuando el usuario hace clic en el enlace malicioso, el código inyectado viaja al sitio web vulnerable que después lo refleja en el navegador del usuario. Luego, el navegador lo ejecuta.[126]

Ataque XSS Stored

Esta es la forma más peligrosa del ataque XSS: el *script* inyectado es guardado permanentemente en el servidor web, ya sea como un mensaje de foro, un comentario de una foto o un producto,. . . . Luego, la víctima carga el *script* malicioso del servidor cuando solicita la información almacenada.

A diferencia de los ataques XSS Reflected en el que el usuario tiene que pulsar el enlace malicioso, en este no es necesario: el ataque XSS Stored se produce tan pronto como el usuario accede a la página web.

Por último, el sitio web *XSS Filter Evasion Cheat Sheet - OWASP*[128] de la Fundación OWASP recopila un gran número de variantes que pueden usar para eludir ciertos filtros defensivos XSS.

8.4.3. ¿Cómo protegerse de un ataque XSS?

A pesar de que hay un gran número de vectores de ataque para realizar un ataque XSS, la validación de las entradas, que ya se ha comentado en la sección 7.5.3 del capítulo anterior, es la primera y más efectiva forma de proteger a los usuarios de la aplicación web de este tipo de ataques.

En concreto, la implementación de unas pocas reglas de codificación y escape de los datos de entrada y dónde y cómo insertarlos puede mitigar significativamente esta vulnerabilidad. La lista completa de las reglas con una explicación detallada y ejemplos puede encontrarse en *Cross Site Scripting Prevention - OWASP Cheat Sheet Series*[129].

Por otro lado, dos métodos efectivos para la mitigación, no la prevención, de los ataques XSS son:

- Utilizar el atributo HTTPOnly[13] en las *cookies*.

[13]HTTPOnly es un atributo de las *cookies* que advierte a los navegadores que dicha *cookie* no puede ser accedida por un *script* del lado del cliente[125]

- Implementar una «Content Security Policy»[14].

8.5. Ataque de inyección de código

8.5.1. ¿Qué es un ataque de inyección de código?

Existen muchos tipos de ataques de inyección de código, aunque todos tienen en común que se aprovechan del tratamiento insuficiente de las entradas que no son de confianza por parte de las aplicaciones[130].

Luego, estos datos sin tratar son interpretados o ejecutados por la aplicación ya sea para realizar una consulta SQL, o NoSQL, a una base de datos o como parte de un comando al sistema operativo.

Además, es una vulnerabilidad todavía muy frecuente, en especial en aplicaciones más antiguas, y aunque es fácil de descubrir al examinar el código fuente, también existen herramientas automatizadas que pueden ayudar a los posibles atacantes a detectar esta vulnerabilidad[131].

8.5.2. ¿Cómo se produce un ataque de inyección de código?

Casi cualquier origen de datos puede ser un vector de ataque para una inyección de código: desde variables de entorno, parámetros, servicios web internos y externos y todo tipo de información recibida de los usuarios[131].

Algunos de estos posibles vectores son[132]:

Lenguajes de consulta
: El más famoso de este tipo es la inyección de código SQL en la que un atacante puede modificar consultas ya existentes a la base de datos.

  Esto puede provocar la lectura de información sensible, la modificación de los datos existentes, la ejecución de operaciones de administración,....

Lenguajes de *scripting*
: Todos los lenguajes utilizados en aplicaciones web tienen alguna función *eval* que recibe código en tiempo de ejecución y lo ejecuta. Cuando el código utiliza datos sin validar y sin escapar de un usuario, es cuando un ataque de inyección de código es posible.

Comandos del sistema operativo
: Este ataque es posible cuando los datos sin asegurar son enviados a una consola del sistema. En este caso los comandos se ejecutan con los privilegios de la aplicación vulnerable. Por este motivo, nunca se debería de dar permisos de administrador a las aplicaciones web.

[14] Esta técnica ya se ha comentado en la sección 7.1.4 como una buena práctica a la hora de implementar una configuración HTTP segura.

8.5.3. ¿Cómo protegerse de un ataque de inyección de código?

La principal defensa frente a los ataques de inyección de código es la validación de las entradas recibidas y la codificación y el escape de los caracteres tal y como se ha explicado en la subsección 7.5.3.

Sin embargo, esta no siempre es una defensa 100 % segura porque algunas aplicaciones requieren de caracteres especiales en las entradas[132].

Por este motivo, existen otras defensas para cada tipo de ataque de inyección de código, como la utilización de *prepared statements*[133] para la inyección SQL o simplemente evitar las llamadas al sistema operativo directamente utilizando una librería o módulo de confianza, como por ejemplo el módulo Child Process[134] de Node.js.

Una lista completa de las posibles defensas frente a los diferentes ataques de inyección de código puede encontrarse en los recursos siguientes:

- *Injection Prevention - OWASP Cheat Sheet Series*[132].
- *SQL Injection Prevention - OWASP Cheat Sheet Series*[135].
- *LDAP Injection Prevention - OWASP Cheat Sheet Series*[136].
- *OS Command Injection Prevention - OWASP Cheat Sheet Series*[137].
- *Injection Prevention in Java - OWASP Cheat Sheet Series*[138].

Capítulo 9

Análisis y diseño del sistema

9.1. Análisis

Tal y como se ha comentado en numerosas ocasiones, el propósito de este proyecto de final de grado es de carácter divulgativo y didáctico, pero no simplemente teórico. Por este motivo, se ha diseñado e implementado una aplicación web sencilla para poner en práctica los consejos explicados, documentar el proyecto con ejemplos reales y como un punto de partida de un sitio web seguro.

Como recordatorio, esta es la lista de funcionalidades que se ha decidido incluir a la aplicación web:

- Creación de las cuentas de usuario con verificación de correo electrónico.
- Recuperación de las cuentas de usuario.
- Autenticación y gestión de la sesión.
- Subir imágenes, ficheros y textos.
- Política de privacidad y gestión de las *cookies*.
- Protección frente los diferentes tipos de ataques estudiados.

9.1.1. Modelo de datos

El modelo de datos es muy simple: la aplicación solo necesita almacenar la información de los usuarios y de los ficheros subidos y la relación entre ellos porque en este momento del desarrollo no se contempla la compartición de los ficheros entre usuarios ni la posterior descarga. El diagrama entidad-relación de los datos puede verse en la figura 9.1.

9.1.2. Modelo de procesos

Los procesos que se han descrito con un diagrama de actividad porque son más complejos son:

- Creación de las cuentas de usuario con verificación de correo electrónico. El diagrama puede verse en la figura 9.2.

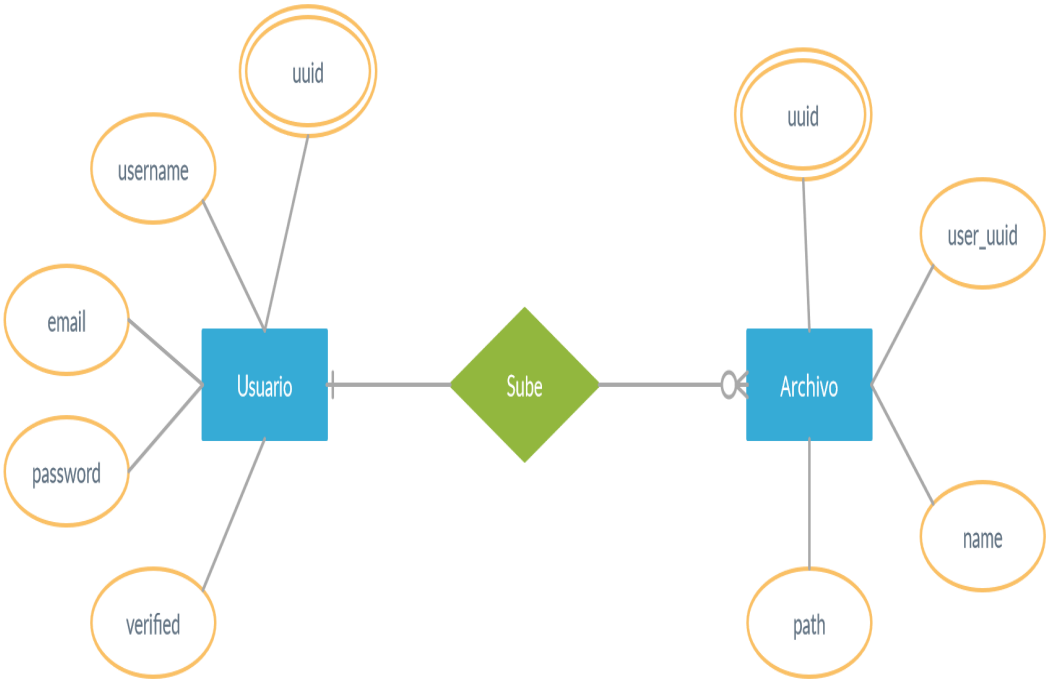

Figura 9.1: Diagrama entidad-relación de los datos

- Recuperación de las cuentas de usuario. El diagrama puede verse en la figura 9.3.

9.2. Diseño del sistema

El diseño habitual de este tipo de aplicaciones es la típica arquitectura cliente-servidor y es como se ha diseñado e implementado en el proyecto, a pesar del reciente crecimiento en uso y en popularidad de la arquitectura de microservicios[1] gracias a sus numerosos beneficios[2].

De haberse utilizado la arquitectura de microservicios, la aplicación se habría dividido en servicios más pequeños: por ejemplo, creación, verificación y recuperación de las cuentas de usuario; autenticación; y almacenaje de ficheros. Todos independientes entre ellos, y quizá incluso implementados con *stacks* diferentes, pero con una interfaz de aplicación pública que permita su comunicación y con una interfaz de usuario que enlace los diferentes componentes.

9.2.1. Diseño del servidor

Una vez comentado el diseño general de la aplicación, a continuación se ha explicado el diseño específico del servidor web.

Se ha escogido la arquitectura y el protocolo REST para la definición de la API del ser-

[1] La arquitectura de microservicios estructura una aplicación como una colección de servicios.

[2] Sus beneficios incluyen: facilidad de mantenimiento, acoplamiento débil, desplegado independiente,... [139]

vidor porque es la más utilizada en la actualidad, aunque las API que utilizan GraphQL[3] han ganado recientemente en popularidad, y porque, en un proyecto en el que el autor tenía, a priori, un conocimiento superficial de la mayoría de buenas prácticas y ataques informáticos recopilados en los dos capítulos anteriores, limitaba la carga de aprendizaje, y por lo tanto disminuía las cosas que podían salir mal al realizar el proyecto, al ya conocer el funcionamiento de las aplicaciones REST y las herramientas necesarias para implementarlas.

Asimismo, al ofrecer una interfaz de aplicación común para los dos servidores a implementar, MEVN y LAMP, se simplifica la implementación de la interfaz de usuario.

En concreto, para los servidores se ha utilizado la arquitectura de tres capas que separa la aplicación en tres niveles lógicos o capas: el controlador, en el que se definen las rutas de entrada típicas de REST; la capa de servicio, en la que se ejecuta la lógica del negocio; y la capa de acceso a los datos, en la que se accede, se modifica y se elimina los datos de las bases de datos.

Un aspecto importante a tener en cuenta al desarrollar servidores web con esta arquitectura es el principio de separación de intereses, en inglés *separation of concerns*, para facilitar las mejoras o reutilizaciones de las diferentes capas. Por ejemplo: las peticiones y respuestas HTTP deben quedarse en la capa del controlador y en la capa de la aplicación no debe haber *queries* a la base de datos.

Por último, la API diseñada e implementada para la aplicación puede encontrarse en el Apiary del autor[141].

9.2.2. Diseño del cliente

Una vez analizado el sistema a implementar en el lado del servidor, se ha diseñado la interfaz de usuario a desarrollar en Vue.js.

El primer paso ha sido realizar el diagrama de la interfaz web, que puede verse en la figura 9.4, con las páginas web a implementar y las relaciones entre ellas.

Después, tendría que haberse diseñado un *wireframe* simple de la interfaz de usuario para luego terminar de complementarlo con un *mockup* más completo. Sin embargo, desafortunadamente, no se ha podido realizar estas dos tareas previas a la implementación de la interfaz de usuario.

[3]GraphQL es un lenguaje de consulta para APIs: proporciona una descripción completa y comprensible de los datos en la API, les da a los clientes el poder de pedir exactamente lo que necesitan y nada más. Además, facilita la evolución de las API con el tiempo[140].

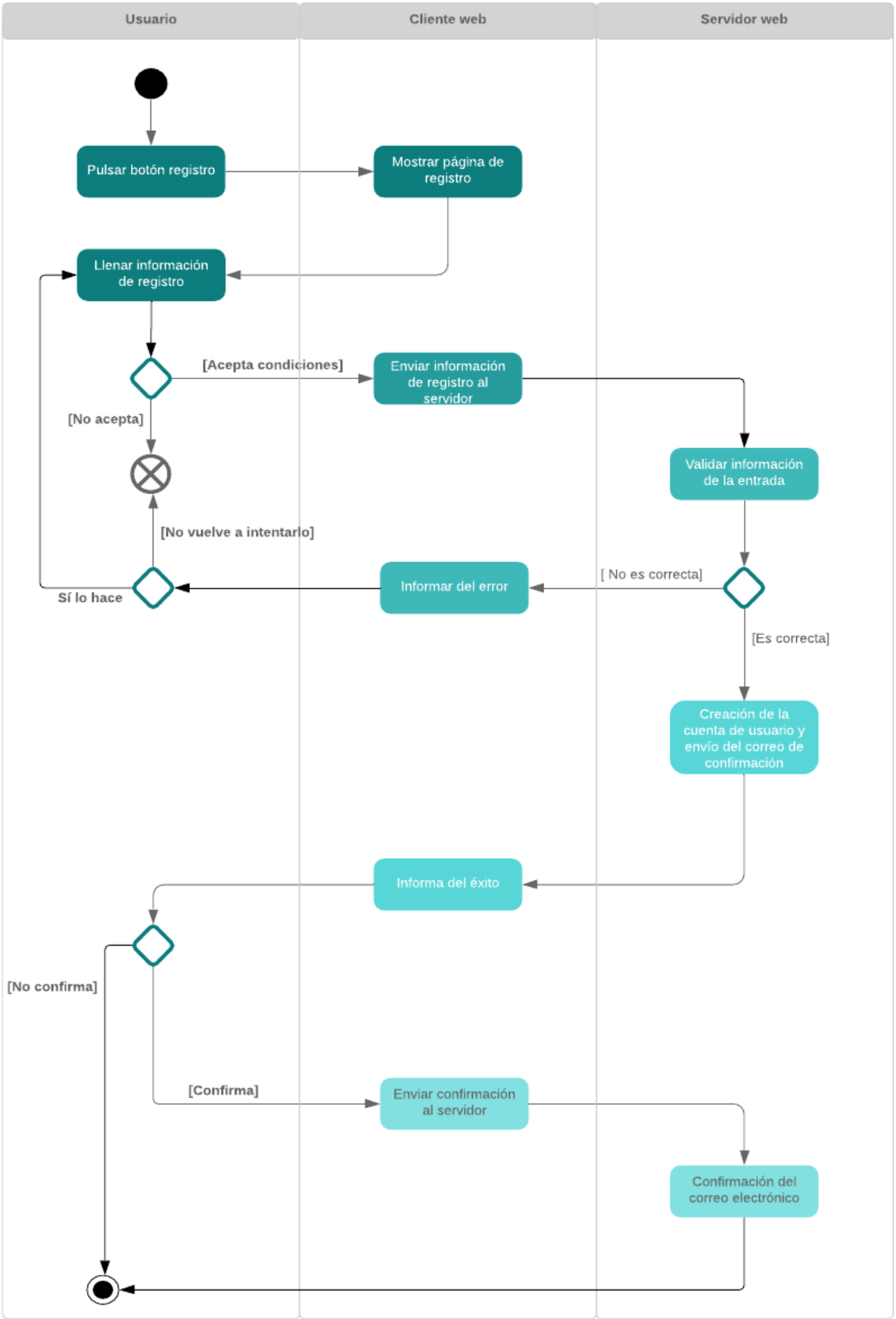

Figura 9.2: Diagrama de actividad para la creación de una cuenta de usuario en Cardona

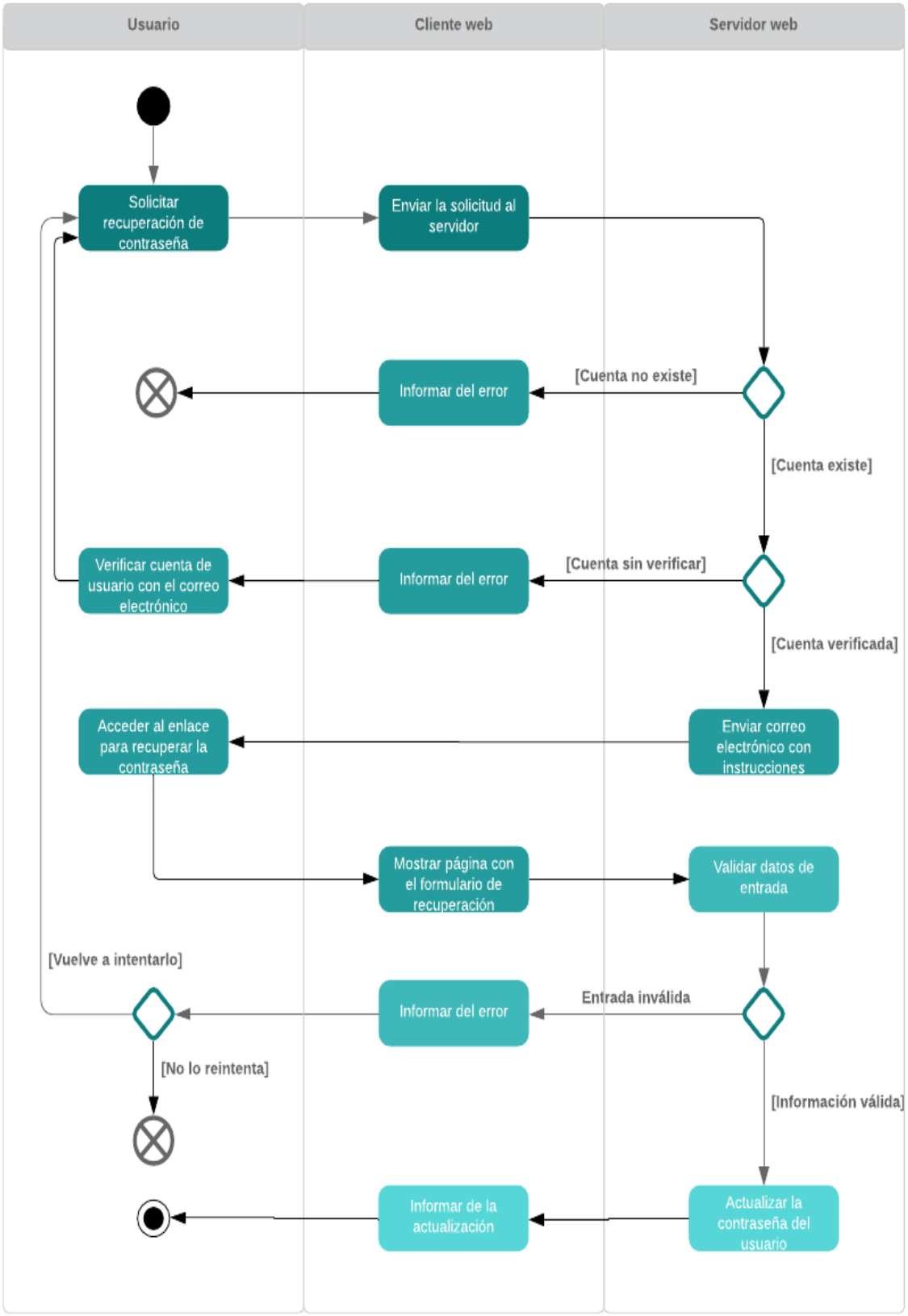

Figura 9.3: Diagrama de actividad para la recuperación de la contraseña de un usuario en Cardona

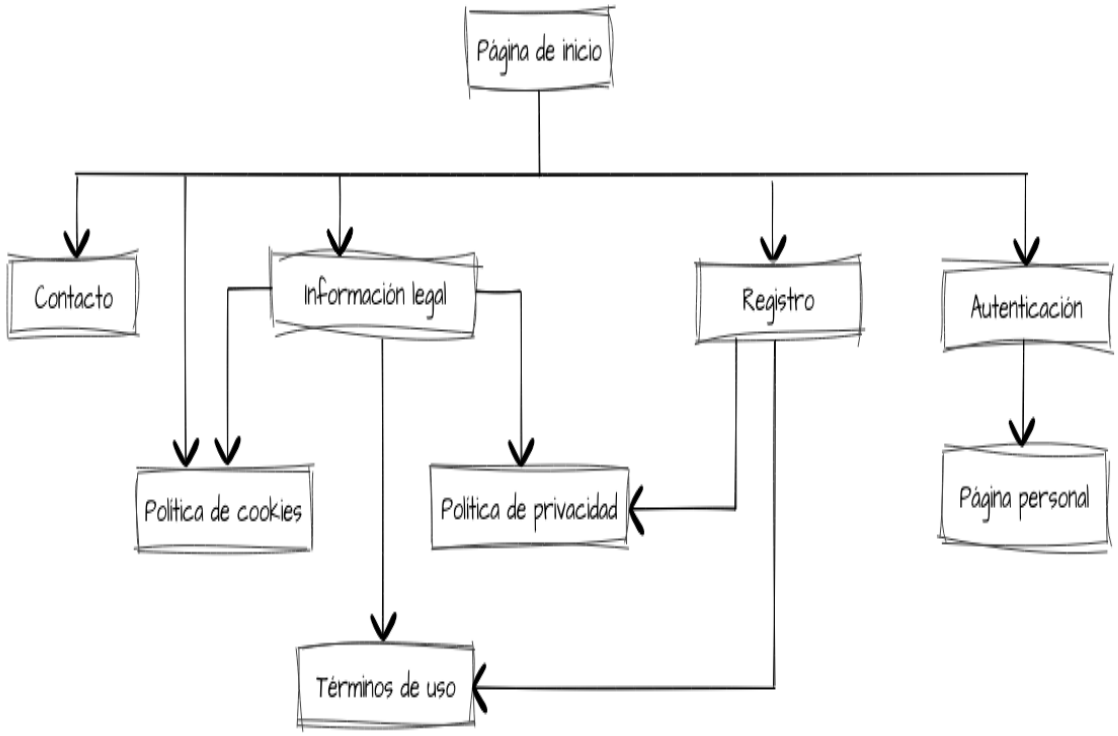

Figura 9.4: Diagrama de la interfaz gráfica de usuario: componentes y enlaces

Capítulo 10

Implementación y pruebas

10.1. Solicitud certificado TLS

10.1.1. Certificado TLS Let's Encrypt

En esta subsección se explican los pasos necesarios para solicitar un certificado TLS a Let's Encrypt. Si ya posee uno, ya sea del propio Let's Encrypt o de otra entidad certificadora de su confianza, puede seguir a la siguiente subsección.

La solicitud del certificado TLS a Let's Encrypt para el servidor en producción se ha hecho con el *software* Certbot[142]. Los pasos que se han seguido en Ubuntu Server 20.04[1] son:

1. Se ha conectado al servidor en la nube con una conexión *ssh*.
2. «snapd» ya viene por dejecto en Ubuntu Server 20.04, pero se ha comprobado que está actualizado con el comando:

   ```
   $ sudo snap install core; sudo snap refresh core
   ```

3. Se ha instalado Certbot con:

   ```
   $ sudo snap install --classic certbot
   ```

   y ejecutado el comando siguiente para asegurarse de que el Certbot puede ser ejectuado correctamente:

   ```
   $ sudo ln -s /snap/bin/certbot /usr/bin/certbot
   ```

4. Se ha ejecutado Certbot con:

   ```
   $ sudo certbot certonly --standalone
   ```

[1]Si utiliza un sistema operativo distinto, había instalado Certbot con anterioridad o el puerto 80 de su servidor está en uso, los pasos aquí descritos no son los correctos por lo que es mejor que acceda a la página web de Certbot y siga las instrucciones que se adaptan mejor a su caso.

Que después de entrar la información básica del certificado, como correo electrónico de contacto o dominio del servidor, guarda el certificado en dos ficheros distintos que se han necesitado más adelante para la implementación HTTPS.

Certbot guarda por defecto estos dos ficheros[2] en la siguiente localización:

```
Certificate is saved at: /etc/letsencrypt/live/{dominio}/fullchain.pem
Key is saved at:         /etc/letsencrypt/live/{dominio}/privkey.pem
```

5. Por último, los certificados Let's Encrypt caducan a los tres meses, pero Certbot se encarga automáticamente de su renovación.

10.1.2. Certificado TLS para desarrollo local

En algunas ocasiones, puede ser necesario trabajar con el protocolo HTTPS en local[143]. Por este motivo, se ha utilizado la herramienta «mkcert» para crear un segundo certificado TLS para el desarrollo en local[144]:

1. Se ha instalado el requisito «certutils».

```
$ sudo apt install libnss3-tools
```

2. Se ha descargado el binario de «mkcert» desde GitHub.

```
$ wget https://github.com/FiloSottile/mkcert/releases/download/
    v1.4.3/mkcert-v1.4.3-linux-amd64
$ sudo cp mkcert-v1.4.3-linux-amd64 /usr/local/bin/mkcert
$ sudo chmod +x /usr/local/bin/mkcert
```

3. Se ha creado la entidad certificadora de «mkcert» en local que se añade en el *trustore* de Mozilla y Chrome.

```
$ mkcert -install
```

4. Por último, se ha creado el certificado TLS para desarrollo en local con el comando:

```
$ mkcert localhost
```

Que crea los dos mismos ficheros que Let's Encrypt en el directorio actual. Este certificado solo puede utilizarse en el dispositivo que se ha creado y con el dominio «localhost».

[2] Ambos archivos deben mantenerse seguro, especialmente el segundo que contiene la llave privada del certificado TLS.

10.2. Implementación *stack* MEVN

10.2.1. Instalación de los requisitos

Los pasos previos, realizados con Ubuntu 21.04, a la creación del proyecto para el desarrollo de la aplicación del lado del servidor de la plantilla MEVN han sido:

1. Actualizar el sistema operativo.

```
$ sudo apt update && sudo apt upgrade -y
```

2. Instalar git.

```
$ sudo apt install git
```

3. Instalar nvm[145]: un administrador de versiones de Node.js.

```
$ curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.38.0/install.sh | bash[3]
```

   Una vez instalado nvm, tiene que cerrarse la terminal i abrir una nueva para que los cambios hagan efecto.

4. Instalar Node 14.17.3[4] LTS[5][146] con la última versión de npm: el administrador oficial de paquetes de Node.

```
$ nvm install --lts --latest-npm
```

5. Añadir las dos extensiones de Visual Studio Code: «Standard JS - JavaScript Standard Style» y «JavaScript standardjs styled snippets» mencionadas en la subsección 6.2.2.

6. Instalar la última versión de MongoDB, la base de datos NoSQL.

```
$ wget -qO - https://www.mongodb.org/static/pgp/server-5.0.asc | sudo apt-key
add -
$ echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu
focal/mongodb-org/5.0 multiverse" | sudo tee
/etc/apt/sources.list.d/mongodb-org-5.0.list
$ sudo apt-get install -y mongodb-org
```

[3]Debe tenerse extremadamente cuidado cuando se descarga un *script* de internet con *curl* y se ejecuta directamente con *bash*.

[4]Se ha escogido esta versión de Node JS porque es la recomendada a 22 de julio de 2021, la fecha de creación del repositorio.

[5]LTS significa «long-term support».

10.2.2. Creación del proyecto

Los pasos para crear el proyecto han sido:

1. Crear el repositorio «cardona-node» Git con la interfaz gráfica de Github con los ficheros «.gitignore», «LICENSE» y «README.md».
2. Clonar el repositorio en el directorio escogido:

```
$ git clone git@github.com:SBergillos/cardona-node.git
```

3. Cambiar al directorio del repositorio.

```
$ cd cardona-node
```

4. Inicializar proyecto de Node.js con:

```
$ npm init
```

   Que pide la información básica para configurarlo. Al tratarse de un repositorio Git detecta ya la mayoría de información por defecto. El único cambio realizado es renombrar el fichero principal de «index.js» a «app.js».

5. Crear el fichero «.npmrc» con el contenido:

```
save=true
save-exact=true
```

   Para que las dependencias de Node.js se guarden directamente en el «package.json» y con la versión exacta para que todos los contribuidores tengan las mismas versiones de los paquetes.

6. Crear la estructura básica de la aplicación teniendo como base el modelo de tres capas: controlador, servicios y modelos[147].

```
root
|   app.js          # Fichero principal
|---api             # Controlador con las rutas y middlewares
|---config          # Variables de entorno y configuraciones
|---loaders         # Modulos previos a la inicializacion
|---models          # Modelos de la base de datos
+---services        # Logica del servicio
```

7. Instalar las primeras dependencias del proyecto: «standard»[6], «express» y «compression»[7].

[6] «standard» se ha instalado como una dependencia de desarrollo al proyecto tal y como se ha explicado en la subsección 6.2.2.

[7] «compression» es un *middleware* de Express que permite la compresión de las respuestas en gzip para optimizar el rendimiento de la aplicación[148] de un modo muy simple.

```
$ npm install express
$ npm install compression
```

8. Por último, se ha programado la versión «Hello World» de Node.js para probar la inicialización de la aplicación y la creación del servidor HTTP, separando la lógica de la aplicación Express de la del servidor HTTP[149].

   - Implementación «app.js».

```
1  // NPM Dependencies
2  import express from 'express'
3  import compression from 'compression'
4
5  // Loaders Dependencies
6  import expressLoader from './loaders/express.js'
7
8  // Routes Dependencies
9  import statusRoutes from './api/routes/status.js'
10
11 const startApp = async () => {
12   // Create Express APP
13   const app = express()
14   // Add gzip compression for optimization
15   app.use(compression())
16
17   // Mount API endpoints
18   app.use('/status', statusRoutes)
19   app.get('/', (req, res) => {
20     res.send('Hello World!')
21   })
22
23   // Wait for Loaders
24   await expressLoader(app)
25 }
26
27 startApp()
```

   - Implementación «express.js».

```
1  // NPM Dependencies
2  import http from 'http'
3
4  const expressLoader = async (app) => {
5    // Get port
6    const port = 8000
7    app.set('port', port)
8
9    // Create HTTP Server
10   const server = http.createServer(app)
11
12   // Listen on port
13   server.listen(port)
14 }
15
16 export default expressLoader
```

Para ejecutar la aplicación en el entorno de desarrollo simplemente se tiene que ejecutar el comando siguiente:

```
$ npm start
```

El código completo en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.0-alpha».

10.2.3. Ciclo de vida de las peticiones

Antes de explicar el ciclo de vida de las peticiones en la aplicación, es necesario realizar dos comentarios sobre el propio *framework*:

- Express integra un servidor HTTP, o HTTPS, por lo que no es necesario utilizar un servidor web externo como Nginx o Apache, aunque por motivos de rendimiento es recomendable utilizar en producción un *reverse proxy* para gestionar los ficheros estáticos, codificación gzip, SSL,... [151]
- Un *middleware* de Express es una función con acceso al objeto de la petición, al objeto de la respuesta y a la funcion *next* para ejecutar el siguiente *middleware*. Los *middlewares* pueden ejecutar código, modificar la petición y la respuesta, terminar el ciclo o llamar al siguiente *middleware*. Para una explicación detallada de como programarlos y utilizarlos, se puede consultar la documentación de Express siguiente:
  - *Writing middleware for use in Express apps*[152].
  - *Using Express middleware*[153].

Una vez hecha la explicación anterior, ya puede describirse el ciclo de vida de una petición en Express:

1. El servidor HTTP integrado en el propio *framework* recibe la petición HTTP y crea dos objetos: «req» y «res» con la información de la petición y respuesta, que aún debe llenarse, respectivamente.
2. Después, esta petición es tratada por todos los *middlewares* definidos hasta que uno da por finalizada la petición y devuelve la respuesta completa, ya sea con éxito o no.

Un punto fundamental de la explicación anterior es que el orden en el que se añaden los *middlewares* a Express importa y mucho, ya que, por ejemplo, añadir un *middleware* que retorna un error 404 antes de definir las propias rutas del servidor provocará que cualquier intento de acceder a una ruta correcta devuelva un «NotFound».

10.2.4. Registro

Para la implantación del registro en la aplicación, se han utilizado dos paquetes: «pino» para las entradas de la aplicación; y «pino-http» para las entradas del servidor HTTP. Además, también se ha instalado «pino-pretty» como dependencia de desarrollo para formatear las entradas de un modo más comprensible para los humanos durante la implementación y pruebas.

La instalación de los tres paquetes se hace con los comandos siguientes:

```
$ npm install pino[8]
$ npm install pino-http
$ npm install pino-pretty --save-dev
```

El formato de escritura que utilizan ambos paquetes es el JSON y la información por defecto es:

pino

- El nivel de severidad. Tiene siete niveles, de más severo a menos: *fatal*, *error*, *warn*, *info*, *debug*, *trace* y *silent*. Sin embargo, cuando registran el nivel, lo hacen como un entero: 60 para *fatal*; 50 para *warn*; y sigue así hasta 0 para *silent*.
- La fecha y la hora del registro. Pueden escogerse tres formatos internacionales distintos: milisegundos o segundos desde «Unix epoch»[9] o en el formato ISO 8601 UTC[10].
- El identificador del proceso y el nombre del dispositivo en el que se está ejecutando la aplicación.

pino-http

- La misma información base que «pino»: nivel de severidad, fecha y hora del registro, identificador del proceso y el nombre del dispositivo.
- Petición HTTP: URL de entrada, método HTTP, encabezados HTTP,...
- Respuesta HTTP: código de estado y encabezados HTTP.
- Tiempo de respuesta.
- Mensaje «*request completed*» o «*request errored*».
- Error en caso de la petición HTTP falle.

Si bien la información que dan es satisfactoria para registrar las peticiones y respuestas HTTP en Express, aún hay mucha información útil que debe añadirse para una implantación correcta y eficaz del sistema de gestión de registros.

En la subsección 7.2.3 se puede encontrar una lista con los elementos a registrar recomendados en *Logging - OWASP Cheat Sheet Series*[72]. De estos, se han añadido los siguientes a ambos tipos de registro:

- Identificador de la transacción.
- Identificador de la aplicación: nombre y versión.

Y a los registros de la aplicación con «Pino»:

- Identificador del usuario, si conocido.

[8] Al haber creado el fichero «.npmrc» en la subsección 10.2.2 con la opción «save=true» npm guarda los paquetes como dependencias por defecto en el repositorio actual.

[9] El número de segundos o milisegundos que han pasado desde el 1 de enero de 1970.

[10] El formato es el siguiente: DD-MM-YYYY HH:MM:SS. Por ejemplo, 25-07-2021 19:37:01.

- Descripción del evento.

Un punto muy importante de la implantación del sistema de registros es el identificador de la transacción ya que permite enlazar diferentes entradas del registro al mismo usuario y a la misma acción. La implementación se ha hecho utilizando un módulo experimental de Node.js, «async hooks»[154], y siguiendo el ejemplo de *Assign TransactionId to each log statement*[155]. Además, el identificador de la transacción se envía al *frontend* de la aplicación como un encabezado de HTTP.

```
1  // NPM Dependencies
2  import { AsyncLocalStorage } from 'async_hooks'
3  import { v4 as uuidv4 } from 'uuid'
4
5  export const asyncLocalStorage = new AsyncLocalStorage()
6
7  // Set incoming requests TransactionId
8  export const transactionIdMiddleware = (req, res, next) => {
9    // The first asyncLocalStorage.run argument is the initialization of the store
       state, the second argument is the function that has access to that store
10   asyncLocalStorage.run(new Map(), () => {
11     // Try to extract the TransactionId from the request header, or generate a new
       one if it doesn't exist
12     const transactionId = req.get('x-transaction-id') || uuidv4()
13
14     // Set the TransactionId inside the store and as a response header
15     asyncLocalStorage.getStore().set('transactionId', transactionId)
16     res.set('X-Transaction-Id', transactionId)
17     next()
18   })
19 }
```

Después, se han utilizado tres estrategias distintas para añadir la información faltante a las entradas del registro:

- Modificando la opción «base» del constructor para añadir la información estática a las entradas (identificador de la aplicación: nombre y versión)[156].

```
1  // Ejemplo de utilizacion de la opcion 'base' del constructor de pino
2  // NPM Dependencies
3  import pino from 'pino'
4  import os from 'os'
5
6  // This information should not be here but in the configuration
7  const app_info = {
8    name: 'cardona-node',
9    version: 'v0.1-alpha'
10 }
11
12 // The logger options: add app name and version to each log
13 const options = {
14   base: {
15     pid: process.pid,
16     hostname: os.hostname(),
17     app: app_info
18   }
19 }
```

```
20
21  // Create pino logger
22  const logger = pino(options)
```

- Añadiendo la opción «mixin» del constructor para que ejecute la función definida y así añadir la información dinámica a las entradas (identificador de la transacción)[156].

```
1   // Ejemplo de utilizacion de la opcion 'mixin' del constructor de pino
2   // NPM Dependencies
3   import pino from 'pino'
4   import os from 'os'
5
6   let n = 0
7
8   // the mixin function is called each time one of the active logging methods is
        called
9   const options = {
10    mixin () {
11      return { line: ++n }
12    }
13  }
14
15  // Create pino logger
16  const logger = pino(options)
```

- Pasando la información como un argumento de la función de registro.

```
1   // Dos ejemplos de como llamar la funcion de registro
2   // NPM Dependencies
3   import pino from 'pino'
4
5   // Create pino logger
6   const logger = pino()
7
8   // Do the log with a simple message
9   logger.info("El log mostrara la informacion base y este mensaje")
10
11  // Or with a JSON object
12  logger.info({
13    id: 'ejemplo',
14    msg: 'El log mostrara la informacion base, el id y este mensaje'
15  })
```

Un ejemplo de una entrada de registro de la aplicación con «pino» después de las modificaciones anteriores es:

```
1  {
2    "level": 30,
3    "time": "2021-07-27T19:51:34.674Z",
4    "pid": 8625,
5    "hostname": "sergi-VivoBook",
6    "app": {
7      "name": "cardona-node",
8      "version": "v0.1-alpha"
9    },
```

```
  "transactionId": "8859b4f5-31d1-4195-bba7-f8351775d50d",
  "msg": "Doing operation 1 of transaction X"
}
```

Y la entrada del registro con la petición y la respuesta HTTP correspondiente con «pino-http»:

```
{
  "level": 30,
  "time": "2021-07-27T19:51:34.680Z",
  "pid": 8625,
  "hostname": "sergi-VivoBook",
  "app": {
    "name": "cardona-node",
    "version": "v0.1-alpha"
  },
  "req": {
    "id": 1,
    "method": "GET",
    "url": "/api/v1/",
    "query": {},
    "params": {},
    "headers": {
      ...
    },
    "remoteAddress": "::ffff:127.0.0.1",
    "remotePort": 55644
  },
  "transactionId": "8859b4f5-31d1-4195-bba7-f8351775d50d",
  "res": {
    "statusCode": 200,
    "headers": {
      ...
    }
  },
  "responseTime": 7,
  "msg": "request completed"
}
```

Por último, un punto importante a destacar de la implementación del sistema de registro en la aplicación es que por cuestiones de rendimiento se han tomado las siguientes decisiones: registrar las entradas de forma asíncrona y que la escritura del registro sea por el canal estándar de salida, es decir, por el STDOUT de la consola[157][158]. Donde se guardan los registros, ya sea en el mismo sistema de ficheros del dispositivo o con el *stack* Elastic, por ejemplo, es una decisión de «DevOps».

El código en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.1-alpha».

10.2.5. Tratamiento de los errores

Los errores en Node.js pueden dividirse en dos grupos[159]:

Errores operacionales

Los errores operacionales representan problemas en tiempo de ejecución esperados y

que pueden ser tratados de manera adecuada. Dos ejemplos de errores operacionales son: fallo en la autenticación de un usuario o abrir un fichero que no existe.

Hay diferentes estrategias que pueden seguirse para recuperarse de un error operacional:

- Tratar con el error directamente.
- Propagar el error al cliente.
- Reintentar la operación.
- Registrar el error e ignorarlo.
- Y en los casos más extremos, detener la ejecución del programa.

Errores de programación

Los errores de programación representan errores inesperados en un código mal escrito. Significan que el código en sí tiene algunos problemas que resolver y se codificó incorrectamente.

Los errores de programación no pueden ser tratados: la mejor opción es detener la ejecución del servidor después de registrar el error.

Asimismo, el tratamiento de los errores en Node.js y Express depende de donde se han producido[160].

Los errores que ocurren en el código síncrono dentro de los controladores de ruta y los *middlewares* no requieren trabajo adicional. Si el código síncrono arroja un error, Express lo detectará y procesará. Por ejemplo, un *middleware* de Express que se ejecuta cuando la URL solicitada por la petición HTTP no coincide con ningún *endpoint* definido en la API:

```
1  // When no API endpoint is called, return an HTTP 404
2  app.use(function (req, res, next) {
3    throw createHttpError.NotFound("The URL doesn't exist!")
4  })
```

Sin embargo, cuando el error ocurre en el código asíncrono es trabajo del programador atrapar y pasar los errores ocurridos a Express con la función *next*. Esto puede hacerse con los bloques habituales de *try...catch* o con el *catch* de una *Promise*[11] de JavaScript. Por ejemplo:

```
1  app.get(baseEndpoint + 'error', (req, res, next) => {
2    Promise.resolve().then(function () {
3      throw createHttpError.InternalServerError("That's not supposed to happen!")
4    }).catch(next)
5  })
```

El controlador de errores predeterminado de Express se encarga de enviar el error al cliente con la información siguiente:

- El código de estado HTTP.

[11] El objeto Promise representa la eventual finalización (o fallo) de una operación asíncrona y su valor resultante[161].

- El mensaje acorde al código de estado. «Not Found» para el código 404, por ejemplo.
- La pila de llamadas o un HTML con el mensaje del código de estado, dependiendo del valor de la variable de entorno.
- Los encabezados HTTP especificados.

El comportamiento del controlador de errores centralizado puede ser personalizado con la implementación de un *middleware* propio que debe ser añadido último para que funcione correctamente. Por ejemplo:

```
1  // Custom error handler
2  app.use(function (err, req, res, next) {
3    errorHandler(err, res)
4  })
5
6  const errorHandler = (err, res) => {
7    // If it is not an HttpError, it probably is an error due a bug
8    // so the best way to deal with it, it's to log the error and crash
9    if (!createHttpError.isHttpError(err)) {
10     logger.fatal(err)
11     process.exit(1)
12   }
13
14   // Return the error response to the client.
15   return res.status(err.statusCode).send({
16     name: err.name,
17     message: err.message
18   })
19 }
```

Sin embargo, saber como tratar los errores es inútil si no se sabe cuáles, y dónde, pueden producirse en el código y si la aplicación puede recuperarse de ellos. Por ejemplo, los errores que se han detectado en la versión actual de la aplicación son:

- No existe el fichero «.env» con las variables de entorno de la aplicación.

  Código

  El código, que puede encontrarse en el fichero «config/index.js», es:

  ```
  1  // Read environment file
  2  const envFound = dotenv.config()
  3  if (envFound.error) {
  4    // This error should crash whole process and can't
  5    // be logged because config is a logger's dependency
  6    throw new Error("Couldn't find .env file")
  7  }
  ```

  Comentario

  Este error no puede ser tratado ni registrado: el fichero «config/index.js» es una dependencia del fichero «loaders/loggers.js» que inicializa los dos objetos encargadas del registro en la aplicación.

  La única opción posible es detener la inicialización de la aplicación y que el programador cree el fichero «.env» con las variables de entorno. Existe un

ejemplo de fichero con las variables de entorno que deben ser definidas llamado «.env.example» en la raíz del proyecto.

- El servidor HTTP no puede escuchar las peticiones HTTP en el puerto o en la *pipe* especificada en el fichero «.env».

  Código

  El código, que puede encontrarse en el fichero «bin/www.js», es:

```
1  // HTTP Server listen on port
2  server.listen(port)
3  server.on('error', onError)
4
5  // Event listener for HTTP server "error" event
6  function onError (error) {
7    if (error.syscall !== 'listen') {
8      throw error
9    }
10
11   const bind = typeof port === 'string'
12     ? 'Pipe ' + port
13     : 'Port ' + port
14
15   // handle specific listen errors with friendly messages
16   switch (error.code) {
17     case 'EACCES':
18       logger.fatal(bind + ' requires elevated privileges')
19       process.exit(1)
20     case 'EADDRINUSE':
21       logger.fatal(bind + ' is already in use')
22       process.exit(1)
23     default:
24       throw error
25   }
26 }
```

  Comentario

  Este error tampoco puede ser tratado, pero sí preservado: el registro ya ha sido inicializado y puede utilizarse para anotar el error antes de detener la inicialización de la aplicación porque, lo más probable, el puerto especificado en el fichero «.env» ya está en uso.

- La URL solicitada por la petición HTTP no existe.

  Código

  El código, que puede encontrarse en el fichero «app.js», es:

```
1  // When no API endpoint is called, return an HTTP 404
2  app.use(function (req, res, next) {
3    throw createHttpError.NotFound("The URL doesn't exist!")
4  })
```

  Comentario

  Este es un error muy común para un servidor web y su tratamiento es igual de fácil: se ha utilizado un *middleware* de Express para detectar estos casos y

lanzar un error HTTP con el código de estado 404 que es recogido y tratado sin problemas por el *handler* de errores personalizado implementado.

- Pérdida de entradas del registro.

  Código

  El código, que puede encontrarse en el fichero «loaders/logger.js», es:

```
1  // asynchronously flush every 10 seconds to keep the buffer empty
2  // in periods of low activity
3  setInterval(function () {
4    logger.flush()
5  }, 10000).unref()
6
7  // use pino.final to create a special logger that
8  // guarantees final tick writes
9  const handler = pino.final(logger, (error, finalLogger, event) => {
10   finalLogger.info(
11     event + " caught on pino.final to make sure to flush all logs'
       registries")
12   if (error) finalLogger.fatal(error, 'error caused exit')
13   process.exit(error ? 1 : 0)
14 })
15
16 // catch all the ways node might exit
17 process.on('beforeExit', () => handler(null, 'beforeExit'))
18 process.on('exit', () => handler(null, 'exit'))
19 process.on('uncaughtException', (err) => handler(err, 'uncaughtException'))
20 process.on('unhandledRejection', () => handler(null, 'unhandledRejection'))
21 process.on('SIGINT', () => handler(null, 'SIGINT'))
22 process.on('SIGQUIT', () => handler(null, 'SIGQUIT'))
23 process.on('SIGTERM', () => handler(null, 'SIGTERM'))
```

  Comentario

  Las entradas del registro se escriben de forma asíncrona por lo que el cierre inesperado de la aplicación puede provocar la pérdida de las últimas entradas que todavía no han sido registradas.

  Por este motivo, se hace un volcado de las entradas que todavía no se han guardado cada diez segundos y en caso de recibir un evento que detendrá la ejecución de la aplicación.

- Dos errores artificiales en dos *endpoints* distintos de la aplicación.

  Código

  El código, que puede encontrarse en el fichero «app.js», es:

```
1  app.get(baseEndpoint + 'error', (req, res, next) => {
2    Promise.resolve().then(function () {
3      throw createHttpError.InternalServerError("That's not supposed to
       happen!")
4    }).catch(next)
5  })
6  app.get(baseEndpoint + 'goodbye', (req, res, next) => {
7    throw Error('Goodbye World!')
8  })
```

Comentario

Ambos errores han sido introducidos en el programa por fines puramente educativos y se han borrado en versiones posteriores de la aplicación.

El primer error es lanzado desde una *Promise* de JavaScript en código asíncrono, pero su tratamiento es el correcto y el *handler* personalizado de los errores puede atraparlo y tratarlo correctamente.

El segundo error, aunque también es atrapado por el *handler* personalizado de los errores, provoca el cierre de la aplicación ya que es un error genérico de JavaScript en vez de un error HTTP. Por motivos de diseño esto se ha considerado un error de programación, no operacional, y se detiene la ejecución del programa.

Por último, un documento imprescindible sobre el tratamiento de errores en Node.js, que ya ha sido citado, es *Error Handling in Node.js*[159] por lo que su lectura completa es obligatoria para aquellos interesados en profundizar en el tratamiento de errores en Node.js.

El código en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.2-alpha».

10.2.6. Protocolo HTTPS

Para implementar el protocolo HTTPS con una configuración segura en Node.js se han seguido los siguientes pasos:

- Se ha solicitado el certificado TLS a Let's Encrypt tal y como está explicado en la subsección 10.1.1.
- Se han añadido tres variables de entorno nuevas a «.env» relacionadas con la configuración HTTPS:

```
# Protocol
HTTPS = true

# TLS Certificate
# The TLS Certificate is only used if HTTPS = true
SSL_CERT_FILE = 'PATH/TO/CERTIFICATE.pem'
SSL_CERT_KEY = 'PATH/TO/CERTIFICATE_KEY.pem'
```

- Se ha modificado el fichero «bin/www.js» para crear el servidor acorde a la variable de entorno HTTPS. Además, cuando las dos variables de entorno HTTPS == true y NODE_ENV == 'production' se crea un segundo servidor HTTP en el puerto 80 que redirige las peticiones.

```
1 // Set port
2 const port = normalizePort(config.port)
3 app.set('port', port)
4
5 // Create server variable
6 let server
7
```

```
if (config.https) {
    if (config.env === 'production') {
        // In production, we have to set a dummy HTTP server
        // to redirect to HTTPS
        http.createServer((req, res) => {
            res.writeHead(307, {
                Location: 'https://' + req.headers.host + req.url
            })
            res.end()
        }).listen(80)
    }

    // Get TLS data: certificate and key
    const tls = {
        cert: fs.readFileSync(config.cert.file),
        key: fs.readFileSync(config.cert.key)
    }

    // Create HTTPS Server
    server = https.createServer(tls, app)
} else {
    // Create HTTP Server
    server = http.createServer(app)
}

server.listen(port)
server.on('error', onError)
server.on('listening', onListening)
```

- Se ha instalado el paquete «Helmet»[162] con:

```
$ npm install helmet
```

La configuración por defecto de «Helmet» resuelve las configuraciones de los encabezados HTTPS recomendados en la subsección 7.1.4. Sin embargo, se ha modificado la política del encabezado «Content-Security-Policy» a la mencionada en la anterior subsección[12].

```
app.use(helmet({
  contentSecurityPolicy: {
    useDefaults: false,
    directives: {
      'default-src': ["'none'"],
      'frame-ancestors': ["'self'"],
      'script-src': ["'self'"],
      'style-src': ["'self'"],
      'img-src': ["'self'"],
      'connect-src': ["'self'"],
      'base-uri': ["'self'"],
      'form-action': ["'self'"],
```

[12]Debe tenerse extremadamente cuidado al instalar y utilizar «Helmet» en una página web existente ya que uno de los encabezados que añade es el «HTTP Strict Transport Security[60]» lo que puede provocar que ciertas páginas que carguen el contenido tanto con HTTP como HTTPS dejen de funcionar, aunque no es recomendable hacerlo en ningún caso ya que puede provocar ciertas vulnerabilidades a ataques como SSL-stripping y otros propios del HTTP.

```
13        'upgrade-insecure-requests': [],
14        'block-all-mixed-content': []
15      }
16    }
17  }))
```

- No obstante, no se ha modificado la configuración por defecto del protocolo SSL/TLS porque ya es lo suficientemente buena, pero de considerarse necesario la documentación oficial de Node.js explica cómo[163].

Algunos de los resultados obtenidos con las herramientas de prueba mencionadas en la subsección 7.1.4 pueden encontrarse en las figuras 10.1, 10.2 y 10.3.

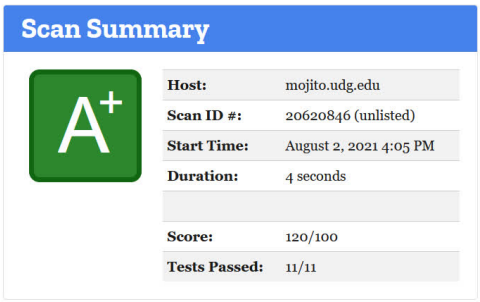

Figura 10.1: Puntuación de la configuración HTTPS con Mozilla Observatory.

Figura 10.2: Puntuación de la política de seguridad de CSP con CSP Scanner.

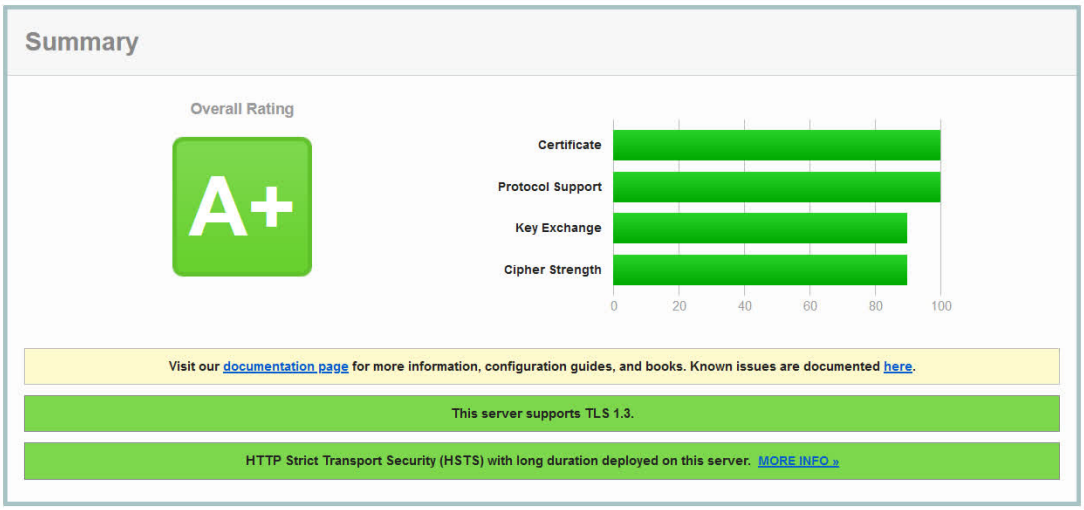

Figura 10.3: Puntuación de la configuración SSL/TLS con SSL Server Test.

La puntuación es muy buena en las cuatro pruebas realizadas, pero «CSP Scanner», en la figura 10.2 ha detectado dos posibles problemas en la configuración del encabezado HTTP «Content-Security-Policy»:

- La política de seguridad CSP que se ha implementado no tiene definida la directiva «report-uri» para recibir informes de infracción por los navegadores. No obstante, en este punto del desarrollo no se ha considerado necesario ya que añade dos nuevos problemas: cómo y dónde recibir estos informes.

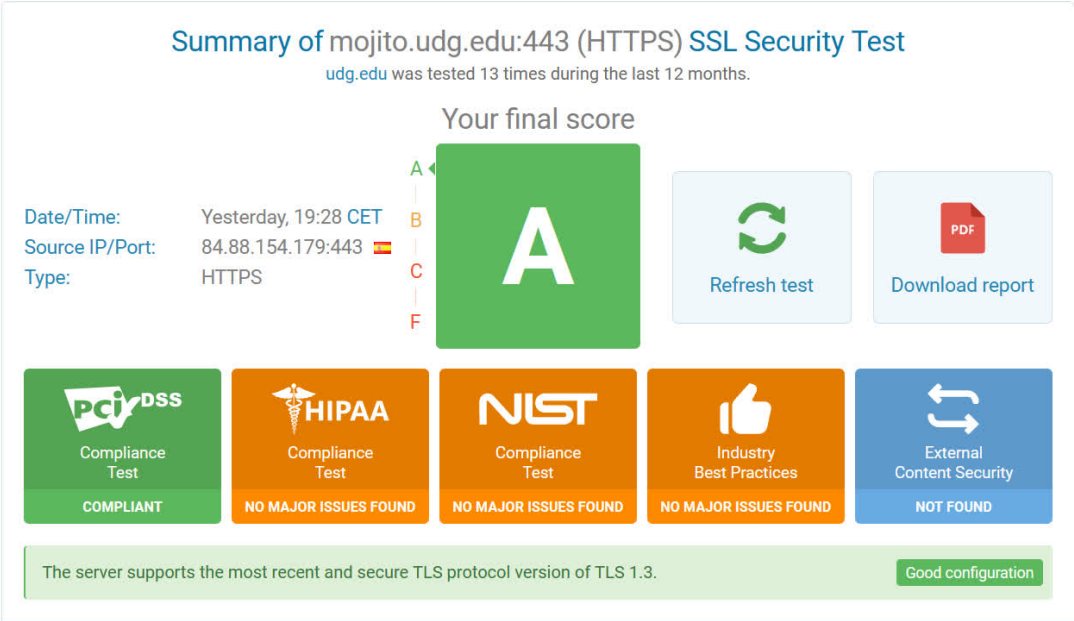

Figura 10.4: Puntuación de la configuración SSL/TLS con ImmuniWeb.

- La directiva «script-src 'self'» puede causar problemas con JSNP, Angular o con archivos cargados por el usuario. Este problema es más preocupante porque «Cardona» ha de permitir a los usuarios subir sus propios archivos al servidor, pero también se ha dejado para más adelante.

«ImmuniWeb», en la figura 10.4 también ha detectado dos posibles problemas en la configuración SSL/TLS:

- El servidor no admite «OCSP *stapling*» para el certificado RSA. OCSP *stapling* es un estándar para comprobar el estado de revocación de certificados digitales X.509[13].
- El servidor admite una renegociación segura iniciada por el cliente que puede ser insegura y permitir ataques de denegación de servicio.

Se ha solucionado el primero de los dos problemas anteriores instalando el paquete npm «ocsp» y añadiendo el código siguiente al fichero «bin/www.js» después de la creación del servidor HTTPS:

```
1  // Create HTTPS Server
2  server = https.createServer(tls, app)
3
4  // Initiate OCSP stapling
5  const cache = new ocsp.Cache()
6  server.on('OCSPRequest', function (cert, issuer, cb) {
7      ocsp.getOCSPURI(cert, function (err, uri) {
8        if (err) return cb(err)
9        if (uri === null) return cb()
10
11       const req = ocsp.request.generate(cert, issuer)
12       cache.probe(req.id, function (err, cached) {
13         if (err) return cb(err)
14         if (cached !== false) return cb(null, cached.response)
```

[13]X.509 es un estándar para definir el formato de la llave pública de certificados digitales.

```
15
16          const options = {
17            url: uri,
18            ocsp: req.data
19          }
20
21          cache.request(req.id, options, cb)
22        })
23      })
24  })
```

Sin embargo, se ha ignorado de momento el segundo problema ya que Node.js lo trata por defecto al limitar el número de renegociaciones iniciadas por el cliente a tres en diez minutos[163]. Aunque esta no es una solución perfecta, aún podría realizarse un ataque de denegación de servicio distribuido, sí que se ha considerado lo suficientemente buena por ahora: el tiempo dedicado a la investigación y a la solución de este problema en concreto supondría un retraso importante frente a otras tareas consideradas por el autor más apremiantes.

Si se repite la prueba con ImmuniWeb, el servidor «cardona-node» cumple ahora también con los estándares HIPAA[14] y NIST[15], además de con el PCI DSS[16], al menos en cuestión de configuración SSL/TLS, como puede verse en la figura 10.5.

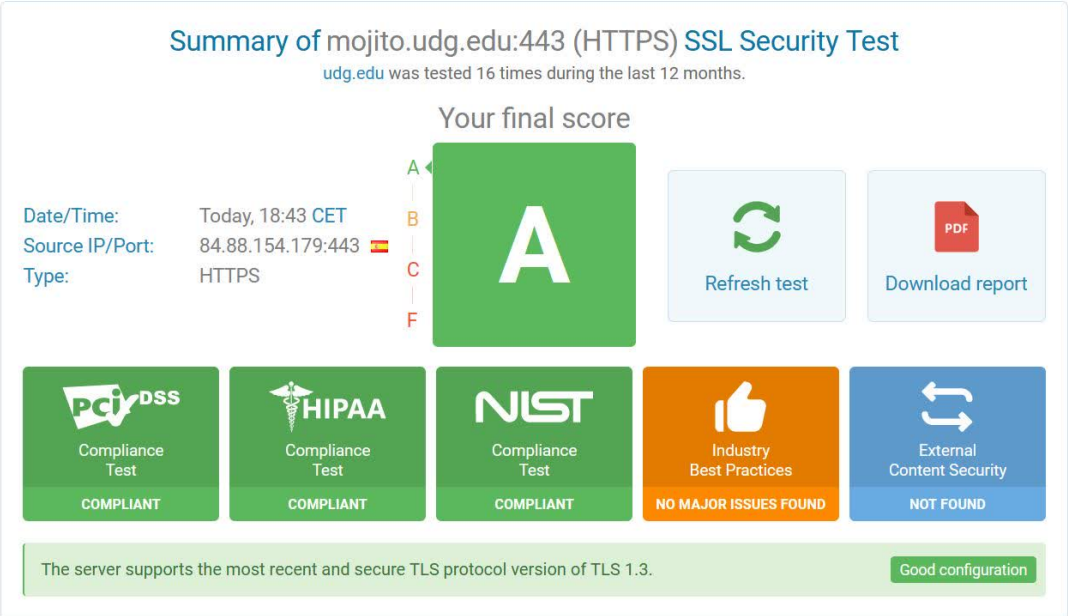

Figura 10.5: Puntuación de la configuración SSL/TLS con ImmuniWeb tras el cambio.

[14]Health Insurance Portability y Accountability Act (HIPAA) establece el estándar para la protección de datos confidenciales de pacientes[164] en EE.UU.

[15]National Institute of Standards and Technology (NIST) es una agencia gubernamental estadounidense no reguladora que desarrolla tecnología, métricas y estándares para impulsar la innovación y la competitividad económica en organizaciones con sede en los EE.UU. en la industria de la ciencia y la tecnología[165].

[16]PCI Data Security Standard (PCI DSS) es un estándar de seguridad que define el conjunto de requerimientos para gestionar la seguridad, definir políticas y procedimientos de seguridad, arquitectura de red, diseño de software y todo tipo de medidas de protección que intervienen en el tratamiento, procesado o almacenamiento de información de tarjetas de crédito[81].

Por último, una advertencia: una configuración de SSL/TLS y HTTPS buena ahora, no significa que lo siga siendo en el futuro ya que continuamente se inventan algoritmos más seguros o se descubren nuevas vulnerabilidades por lo que es una buena práctica repetir regularmente las pruebas para asegurarse de que la configuración sigue siendo segura.

El código en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.3-alpha».

10.2.7. Protección frente ataques DoS

Tal y como se ha comentado en la subsección 8.1.3, la mayoría de métodos de mitigación de ataques de denegación de servicio descritos están fuera del ámbito de este proyecto por su complexidad y porque su implantación es posterior al desarrollo.

Sin embargo, hay dos métodos para proteger la capa de aplicación o a la de representación, capas siete y seis del modelo OSI, que pueden implementarse en este punto:

- Limitar la frecuencia de las peticiones HTTP.
- Deshabilitar la renegociación del cliente del protocolo TLS.
- Revisar las expresiones regulares implementadas.

El primer punto puede implementarse con diferentes paquetes instalables con «npm». Los dos ejemplos que se han analizado son:

«express-rate-limiter»
: «express-rate-limiter» es un *middleware* de Express que permite limitar la frecuencia de las peticiones HTTP de un modo muy simple, como puede verse en el código de ejemplo más abajo obtenido de la documentación oficial del paquete, pero su simplicidad puede ser insuficiente para contrarrestar ataques de denegación de servicio distribuidos más complejos[166].

```
1  const rateLimit = require("express-rate-limit");
2
3  const apiLimiter = rateLimit({
4    windowMs: 15 * 60 * 1000, // 15 minutes
5    max: 100
6  });
7
8  // only apply to requests that begin with /api/
9  app.use("/api/", apiLimiter);
```

«rate-limiter-flexible»
: «rate-limiter-flexible» es más flexible, pero también más complejo. Permite limitar el número de acciones por un identificador, no por un punto de entrada como «express-rate-limiter»[167].

  Se puede implementar como un *middleware* de Express que limita a seis las peticiones HTTP en un segundo por dirección IP:

```
1  // NPM Dependencies
2  import { RateLimiterMemory } from 'rate-limiter-flexible'
```

```
3  import createHttpError from 'http-errors'
4
5  // Rate limiter options
6  // Points: number of HTTP requests
7  // Duration: size of the fixed window in seconds
8  const opts = {
9    points: 6,
10   duration: 1
11 }
12
13 // Create rate limiter
14 const rateLimiter = new RateLimiterMemory(opts)
15
16 const rateLimiterMiddleware = (req, res, next) => {
17   rateLimiter.consume(req.ip)
18     .then((rateLimiterRes) => {
19       res.set('X-RateLimit-Limit', opts.points)
20       res.set('X-RateLimit-Remaining', rateLimiterRes.remainingPoints)
21       res.set('X-RateLimit-Reset',
22         new Date(Date.now() + rateLimiterRes.msBeforeNext))
23       next()
24     })
25     .catch((rateLimiterRes) => {
26       res.set('Retry-After', rateLimiterRes.msBeforeNext / 1000)
27       next(createHttpError.TooManyRequests('Too many requests!'))
28     })
29 }
30
31 export default rateLimiterMiddleware
```

Sin embargo, en sistemas más complejos la dirección IP no puede ser suficiente como identificador para el limitador. Por ejemplo, varios usuarios de la aplicación pueden tener la misma dirección IP.

Una alternativa es usar el identificador de usuario, cuando está autenticado, o la huella digital del dispositivo del cliente.

En cuanto el segundo método, ya está implementado parcialmente por defecto en Node.js, como se ha comentado en la subsección 10.2.6 de este mismo capítulo. La estrategia que sigue Node.js es la misma que la implementada para mitigar la inundación HTTP: limitar la frecuencia de las renegociaciones del cliente[163].

Por último, en este momento del desarrollo de la aplicación no se ha implementado ninguna expresión regular por lo que la aplicación no es vulnerable a un ataque ReDoS[93].

El código en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.4-alpha».

10.2.8. Validación de las entradas

La validación de las entradas se ha implementado con el paquete npm «Joi»[168], una poderosa herramienta para validar los datos de entrada en JavaScript utilizando JSON Schema[94].

Para enseñar su funcionamiento, se han implementado dos rutas provisionales en la API para validar la entrada de la creación de cuentas de usuario y la autenticación de los

usuarios.

Validación de la creación de usuarios

El JSON Schema que se ha implementado para la validación de las entradas para la creación de las cuentas de usuario es:

```
1  import Joi from 'joi'
2
3  export const userCreationSchema = Joi.object({
4    username: Joi.string()
5      .trim()
6      .alphanum()
7      .min(3)
8      .max(30)
9      .required(),
10   email: Joi.string()
11     .trim()
12     .email({
13       tlds: { allow: false }
14     })
15     .required(),
16   password: Joi.string()
17     .min(8)
18     .max(64)
19     .required()
20 })
```

El esquema anterior define las siguientes restricciones sobre los valores de entrada:

username

Las restricciones sobre el nombre de usuario son:

- Es una cadena de caracteres requerida para la validación.
- Puede tener espacios al principio y al final de la cadena, pero se eliminan para la validación posterior.
- Solo con caracteres alfanuméricos.
- Y un tamaño entre tres y treinta caracteres.

email

Las restricciones sobre la dirección de correo electrónico son:

- Es una cadena de caracteres requerida para la validación.
- Puede tener espacios al principio y al final de la cadena, pero se eliminan para la validación posterior.
- Y parece un correo electrónico[17].

password

Las restricciones sobre la contraseña son:

- Es una cadena de caracteres requerida para la validación.

[17] Tal y como se ha comentado en la subsección 7.5.3, la validación sintáctica del correo electrónico es muy compleja por lo que la aproximación habitual es una validación sintáctica simple y después enviar un correo de confirmación.

- Con un tamaño mínimo de ocho caracteres.
- Y un tamaño máximo de sesenta y cuatro caracteres[18].

Si la entrada cumple con todas las restricciones anteriores, entonces se considera válida.

Validación de la autenticación de usuarios

El JSON Schema que se ha implementado para la validación de las entradas para la autenticación del usuario es:

```
1  import Joi from 'joi'
2
3  export const userLoginSchema = Joi.object({
4    username: Joi.string()
5      .trim(),
6    email: Joi.string()
7      .trim()
8      .email({
9        tlds: { allow: false }
10     }),
11   password: Joi.string()
12     .required()
13 })
14   .xor('username', 'email')
```

El esquema anterior define las siguientes restricciones sobre los valores de entrada:

username

Las restricciones sobre el nombre de usuario son:

- Es una cadena de caracteres.
- Puede tener espacios al principio y al final de la cadena, pero se eliminan para la validación posterior.

email

Las restricciones sobre la dirección de correo electrónico son:

- Es una cadena de caracteres.
- Puede tener espacios al principio y al final de la cadena, pero se eliminan para la validación posterior.
- Y parece un correo electrónico.

password

Las restricciones sobre la contraseña son:

- Es una cadena de caracteres requerida para la validación.

Además, el *username* o el *email* deben encontrarse en los datos de entrada: se necesita solo uno de los dos para la autenticación del usuario porque ambos han de ser únicos.

[18]Es recomendable añadir un máximo al tamaño de la contraseña para prevenir ciertos ataques de denegación de servicio al rechazar contraseñas, por ejemplo, de cuatro mil caracteres.

Sin embargo, debe tenerse en cuenta que en este punto del desarrollo todavía no se han analizado las funcionalidades anteriores y que los esquemas de los datos de entrada son susceptibles a cambios posteriores.

La comprobación en ambos casos se hace a través de un *middleware* de las rutas:

```
// NPM Dependencies
import createHttpError from 'http-errors'

// Input validation middleware
const inputValidation = (schema) => {
  return (req, res, next) => {
    const { error, value } = schema.validate(req.body)
    if (error) {
      throw createHttpError.BadRequest(error.message)
    }
    res.locals.value = value
    next()
  }
}

export default inputValidation
```

Que se invoca como:

```
// Schema dependencies
import {
  userCreationSchema,
  userLoginSchema
} from '../schemas/user.js'

// Middleware dependencies
import validateInput from '../middlewares/input_validation.js'

const router = Router()

// Create user account
router.post('/register',
    validateInput(userCreationSchema),
    (req, res) => {
        return res.status(200).send('user creation validated')
})

// Login user
router.post('/login',
    validateInput(userLoginSchema),
    (req, res) => {
        return res.status(200).send('user login validated')
})
```

Como puede verse en el código anterior, las dos rutas implementadas para esta subsección son completamente provisionales ya que solo validan la entrada obtenida.

El código en este punto del desarrollo puede encontrarse en el GitHub del autor[150] con la etiqueta «v0.5-alpha».

10.3. Implementación *stack* LAMP

10.3.1. Instalación de los requisitos

Los pasos previos, realizados con Ubuntu 21.04, a la creación del proyecto para el desarrollo de la aplicación del lado del servidor de la plantilla LAMP han sido:

1. Actualizar el sistema operativo.

```
$ sudo apt update && sudo apt upgrade -y
```

2. Instalar git, si aún no se tiene instalado.

```
$ sudo apt install git
```

3. Añadir dos repositorios adicionales para instalar PHP 8.

```
$ sudo apt install ca-certificates apt-transport-https software-properties-common
$ sudo add-apt-repository ppa:ondrej/apache2
$ sudo add-apt-repository ppa:ondrej/php
```

4. Instalar Apache 2, el servidor web.

```
$ sudo apt install apache2
```

5. Instalar MySQL, la base de datos.

```
$ sudo apt install mysql-server
```

   Una vez se ha instalado MySQL es recomendable ejecutar el comando siguiente para eliminar ajustes predeterminados poco seguros, especialmente en producción:

```
$ sudo service mysql start
$ sudo mysql_secure_installation
```

6. Instalar PHP 8 y los módulos necesarios para que PHP se comunique con la base de datos MySQL, para que Apache gestione los archivos PHP o porque son dependencias de Laravel.

```
$ sudo apt install php8.0 libapache2-mod-php8.0 libapache2-mod-gnutls
php8.0-common php8.0-cli php8.0-mysql php8.0-mbstring php8.0-xml php8.0-zip
php-bcmath
```

7. Instalar Composer, una herramienta de administración de dependencias para PHP.

```
$ sudo apt install php-cli unzip
$ cd ~
$ curl -sS https://getcomposer.org/installer -o composer-setup.php
```

```
```

Tal y como ya se ha mencionado anteriormente, se debe tener mucho cuidado al descargar y ejecutar *scripts* de Internet. Por este motivo, se puede realizar el siguiente paso adicional para verificar el instalador de Composer.

```
$ HASH=`curl -sS https://composer.github.io/installer.sig`
$ php -r "if (hash_file('SHA384', 'composer-setup.php') === '$HASH') { echo
'Installer verified'; } else { echo 'Installer corrupt';
unlink('composer-setup.php'); } echo PHP_EOL;"
```

Instalar Composer para todo el sistema.

```
$ sudo php composer-setup.php --install-dir=/usr/local/bin --filename=composer
```

8. Instalar Laravel, el *framework* escogido para desarrollar la aplicación web con PHP, como una dependencia global de Composer:

```
$ composer global require laravel/installer
```

9. Añadir la extensión «PHP Sniffer & Beautifier» en Visual Studio Code.

   Para su correcto funcionamiento se necesita instalar también «phpcs», pero como se ha decidido utilizar a nivel de proyecto, no globalmente, la instalación de «phpcs» debe hacerse en la subsección siguiente.

10.3.2. Creación del proyecto

Los pasos para crear el proyecto han sido:

1. Crear el repositorio «cardona-lamp» Git con la interfaz gráfica de Github sin ficheros ya que más adelante se sobreescriben.
2. Crear el proyecto con Composer y Laravel:

```
$ laravel new cardona-lamp --git --branch="main"
```

3. Cambiar al directorio del repositorio.

```
$ cd cardona-lamp
```

4. Añadir la URL del repositorio remoto con:

```
$ git remote add origin git@github.com:SBergillos/cardona-lamp.git
```

5. Añadir al directorio base un fichero con la licencia MIT.
6. Modificar el fichero «README.md» por defecto al provisional del proyecto.
7. Añadir la dependencia de desarrollo «phpcs» para la revisión del código.

```
composer require --dev squizlabs/php_codesniffer
```

8. Revisar el código con «phpcs» y la extensión de Visual Studio Code instalada.
9. Realizar el *commit* y forzarlo en remoto con la comanda:

```
$ git push -u -f origin main
```

10. Para ejecutar la aplicación en el entorno de desarrollo simplemente se tiene que ejecutar el comando siguiente:

```
$ php artisan serve
```

El código completo en este punto del desarrollo puede encontrarse en el GitHub del autor[169] con la etiqueta «v0.0-alpha».

10.3.3. Funcionamiento del *framework*

A diferencia del proyecto creado con Node.js y npm para el *stack* MEVN que solo contenía un esqueleto de la aplicación Node, el proyecto iniciado con Composer y Laravel para el *stack* LAMP ya es un sitio web más completo e implementa algunas de las buenas prácticas que se han comentado en los capítulos anteriores, como el registro o protección frente ataques CSRF. Por este motivo, antes de comentar aquellas buenas prácticas que ya han implementado y añadir aquellas que no, se ha dedicado esta subsección a explicar el ciclo de vida de las peticiones HTTP, la estructura del proyecto y las configuraciones imprescindibles.

La información siguiente puede encontrarse más detallada en la propia documentación de Laravel:

- *Request Lifecycle - Laravel - The PHP Framework For Web Artisans*[170].
- *Configuration - Laravel - The PHP Framework For Web Artisans*[171].
- *Directory Structure - Laravel - The PHP Framework For Web Artisans*[172]

También es recomendable leer la siguiente documentación para aprender tres conceptos muy importantes de Laravel:

- *Service Providers - Laravel - The PHP Framework For Web Artisans*[173]
- *Middleware - Laravel - The PHP Framework For Web Artisans*[174]
- *Views - Laravel - The PHP Framework For Web Artisans*[175]

Ciclo de vida de las peticiones HTTP

El ciclo de vida de las peticiones HTTP en Laravel es:

1. El servidor web, Apache en este caso, redirige todas las peticiones HTTP al fichero «public/index.php» de la aplicación Laravel. Este fichero es el encargado de cargar el resto del *framework*.

2. Después, la petición HTTP es enviada al *kernel* de HTTP: «app/Hhttp/Kernel.php». En este fichero se definen los *bootstrappers*, como el tratamiento de los errores, el registro, etc, que serán ejecutados antes que la petición. Aquí también se definen los *middlewares*, gestión de la sesión, verificación del *token* de CSRF, etc, que todas las peticiones deben pasar antes de ser enviados a la aplicación.

3. La petición es tratada por la ruta correspondiente y retornada al *kernel* de HTTP que devuelve la respuesta.

Los *bootstrappers* más importantes son los proveedores de servicios de la aplicación: son los responsables de cargar los componentes del *framework*, como base de datos, validación, enrutamiento, etc.

Estructura de directorios del proyecto

La estructura del directorio raíz es:

Directorio App
: Contiene el núcleo del código de la aplicación.

Directorio Bootstrap
: Contiene el fichero «app.php» que se encarga de arrancar el *framework*. Además contiene el directorio «cache» para ficheros generados automáticamente por el *framework*.

  Por norma general, no debería de modificarse este directorio.

Directorio Config
: Contiene todos los ficheros de configuración de la aplicación.

Directorio Database
: Contiene las migraciones de la base de datos, constructores de modelos y semillas. También puede utilizarse como directorio para una base de datos SQLite.

Directorio Public
: Contiene el fichero «index.php» que es el punto de entrada por todas las peticiones. También contiene todos los activos como imagenes, JavaScript y CSS.

Directorio Resources
: Contiene las vistas, los activos sin compilar y los archivos de idiomas.

Directorio Routes
: Contiene todas las rutas definidas por la aplicación.

Directorio Storage
: Contiene los registros, sesiones basadas en archivos, archivos de caché,...

directorio Tests
: Contiene las pruebas automatizadas.

La estructura del directorio «app», que contiene la mayoría de la aplicación, es:

Directorio Console
: Contiene los comandos Artisan personalizados.

Directorio Exceptions
Contiene el gestor del tratamiento de los errores.

Directorio Http
Contiene los controladores, *middlewares* y formularios.

Directorio Models
Contiene el modelo de las clases. Permiten interactuar con a base de datos.

Directorio Providers
Contiene todos los proveedores de servicos de la aplicación.

Existen otros directorios opcionales en este nivel para funcionalidades más avanzadas de Laravel como eventos o trabajos en cola que no se consideran en este punto del proyecto.

Configuración del proyecto

Laravel ofrece un gran número de opciones para configurar el proyecto a su medida. Aquellas configuraciones que dependan del entorno en que la aplicación está corriendo están definidas en el fichero «.env» en el directorio raíz del proyecto.

La lista completa de los parámetros configurables de la aplicación puede encontrarse en los distintos ficheros del directorio «config» documentados con comentarios. Si bien, la mayoría de ficheros de configuración son para las funcionalidades avanzadas que no están habilitadas por defecto.

10.3.4. Registro

Tal y como ya se ha comentado, Laravel implementa por defecto un sistema de gestión de registros muy completo y personalizable a través del fichero de configuración «logging.php», las variables de entorno «LOG_CHANNEL» y «LOG_LEVEL», el patrón de diseño Facade[19] y la librería Monolog[20][176].

El registro en Laravel está basado en «canales». Cada canal representa una forma de escribir la información en el registro: ya sea en un simple fichero o con un sistema externo como Slack, que ya se ha comentado brevemente en la subsección 7.2. También permite que las entradas del registro se escriban en múltiples canales.

Por ejemplo, un extracto del fichero de «logging.php» en el que se definen tres canales distintos:

```
'channels' => [
    'single' => [
        'driver' => 'single',
        'path' => storage_path('logs/laravel.log'),
        'level' => env('LOG_LEVEL', 'debug'),
    ],
```

[19]El patrón de diseño Facade simplifica la interfaz de un subsistema más complejo, en este caso, el subsistema de registro de Laravel.

[20]Monolog es una libreria de registro de PHP que implementa la interfaz de registro PSR-3[40], el estándar definido por PHP-FIG.

```
    'daily' => [
        'driver' => 'daily',
        'path' => storage_path('logs/laravel.log'),
        'level' => env('LOG_LEVEL', 'debug'),
        'days' => 14,
    ],

    'slack' => [
        'driver' => 'slack',
        'url' => env('LOG_SLACK_WEBHOOK_URL'),
        'username' => 'Laravel Log',
        'emoji' => ':boom:',
        'level' => env('LOG_LEVEL', 'critical'),
    ],
]
```

El primer canal escribe en un solo fichero localizado en el directorio de Storage y que se llama «logs/laravel.log». A su vez, el segundo canal guarda la entrada en el mismo fichero, pero este se renueva cada catorce días. El último canal definido envía las entradas del registro al *stack* Slack, en la URL definida en el fichero de «.env» por la variable «LOG_SLACK_WEBHOOK_URL».

Además, permite registrar las entradas con ocho niveles de severidad, de menos a más severos: *debug*, *info*, *notice*, *warning*, *error*, *critical*, *alert* y *emergency*.

El uso del registro es muy simple gracias al patrón Facade. Primero, se ha creado una vista nueva con el típico «Hello World!». Cuando el usuario accede a esta, se registra la entrada en el canal por defecto de la forma siguiente:

```
<?php

use Illuminate\Support\Facades\Route;
use Illuminate\Support\Facades\Log;

Route::get('/hello', function () {
    Log::info('Accessing hello world page');
    return view('hello');
});
```

Si en vez de querer utilizar el canal por defecte se quiere especificar otro de distinto también es muy simple. Solo debe de modificarse la sentencia anterior por:

```
Log::channel('single')->info('Accessing hello world page');
```

Sin embargo, este sistema de gestión de registros no permite cambiar el formato de las entradas o añadir información estática, como el nombre y la versión de la aplicación, como sí podía hacerse fácilmente con Pino para el *stack* MEVN, comentado en la subsección 10.2.4.

Para una explicación más detallada del sistema de registro de Laravel se puede consultar *Logging - Laravel - The PHP Framework For Web Artisans*[177], si bien con esta corta explicación y demostración ya se puede observar el enorme potencial y simplicidad de uso del sistema de registro implementado por defecto por Laravel.

El código completo en este punto del desarrollo puede encontrarse en el GitHub del

autor[169] con la etiqueta «v0.1-alpha».

10.3.5. Tratamiento de los errores

El tratamiento de las excepciones y los errores en Laravel ya está implementado también por defecto de un modo muy simple y directo: cuando un método o módulo lanza una excepción, que no es atrapada y tratada posteriormente por un bloque «try/catch» implementado por el programador, esta es manejada por la clase «App/Exceptions/Handler».

El detalle en el que se guardan los errores puede modificarse también con la variable de configuración «debug» en el fichero de configuración «app.php» y la variable de entorno «APP_DEBUG» del fichero «.env». Cuando se está desarrollando interesa ver, por ejemplo, la línia de código en que se ha lanzado la excepción o la pila de llamadas que ha provocado el error, mientras que si está se muestra en producción se corre el riesgo de exponer información de configuración sensible ante un posible atacante.

En el código siguiente pueden verse las tres formas distintas en que pueden manejarse las excepciones en Laravel:

```
<?php

use Illuminate\Support\Facades\Route;

Route::get('/error', function () {
    throw new Exception('Error!');
});

Route::get('/hello_report', function () {
    try {
        throw new Exception('Someone accessed hello world page! Report, but show the
    view!');
    } catch (Throwable $e) {
        report($e);
    }
    return view('hello');
});
```

En la primera ruta, se lanza una excepción que es recogida por la clase por defecto de Laravel. Esta clase se encarga de registrar el error y retornar una vista con un nivel de detalle diferente dependiendo de la configuración.

En la segunda ruta, se lanza otra excepción, pero esta es atrapada y tratada por el propio programador con un bloque «try/catch». Esta excepción despues es enviada por la función *report* a la clase «App/Exceptions/Handler» que la registra en el *log*, pero no finaliza la ejecución del proceso.

Para una explicación más detallada del tratamiento de errores en Laravel se puede consultar *Error Handling - Laravel - The PHP Framework For Web Artisans*[178].

El código completo en este punto del desarrollo puede encontrarse en el GitHub del autor[169] con la etiqueta «v0.2-alpha».

10.3.6. Protocolo HTTPS

Añadir el protocolo HTTPS es un poco más complicado, especialmente en local, ya que debe hacerse a través de un servidor externo, como Apache o Nginx. En este caso se ha utilizado Apache 2.

La configuración del servidor Apache se ha creado utilizando *Mozilla SSL Configuration Generator*[55] con las opciones siguientes:

- *Server Software*: Apache.
- *Mozilla Configuration*: Intermediate.
- *Miscellaneous*: HTTP Strict Transport Security y OCSP Stapling.

Por ejemplo, la configuración de Apache 2 que se ha utilizado para servir la aplicación en Laravel en producción, partir de la configuración anterior y de las buenas prácticas explicadas en la subsección 7.1.4, es:

```
# this configuration requires mod_ssl, mod_socache_shmcb, mod_rewrite, and mod_headers
# redirect from http to https
<VirtualHost *:80>
    ServerName example.com
    Redirect permanent / https://example.com/
</VirtualHost>

<VirtualHost *:443>
    SSLEngine on
    ServerName example.com

    ServerAdmin admin@example.com
    DocumentRoot /var/www/cardona-lamp/public

    <Directory /var/www/cardona-lamp/public>
        DirectoryIndex index.php
        Options Indexes MultiViews
        AllowOverride None
        Require all granted
    </Directory>

    SSLCertificateFile      /path/to/your/cert/fullchain.pem
    SSLCertificateKeyFile   /path/to/your/cert/privkey.pem

    Protocols h2 http/1.1

    Header always set Strict-Transport-Security "max-age=63072000"
    Header always set Content-Security-Policy: "default-src 'none'; base-uri
    'self';frame-ancestors 'self'; script-src 'self'; object-src 'self'; style-src 'self';
    img-src 'self'; media-src 'self'; frame-src 'self'; font-src 'self'; connect-src
    'self'; form-action 'self'; upgrade->    Header always set X-Frame-Options: SAMEORIGIN
    Header always set X-Content-Type-Options: nosniff
</VirtualHost>

SSLProtocol             all -SSLv3 -TLSv1 -TLSv1.1
SSLCipherSuite
    ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-GCM-SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-RSA-AES
SSLHonorCipherOrder     off
SSLSessionTickets       off
```

```
SSLUseStapling On
SSLStaplingCache "shmcb:logs/ssl_stapling(32768)"
```

Para que la configuración de Apache anterior funcione, se deben añadir los módulos siguientes:

```
$ sudo a2enmod ssl
$ sudo a2enmod socache_shmcb
$ sudo a2enmod rewrite
$ sudo a2enmod headers
```

Si se repiten las pruebas de la configuración SSL/TLS y HTTPS, realizadas también en la subsección 10.2.6, se obtienen resultados muy parecidos a los obtenidos ya con Node.js y Express para el *stack* MEVN, tal y como puede verse en las figuras 10.6, 10.7 y 10.8.

Sin embargo, el resultado obtenido con ImmuniWeb, en la figura 10.9, es terrible. Si bien, el problema parece ser externo: el servidor en producción es una máquina virtual dentro de un servidor del grupo BCDS y, por el motivo que sea, parece que la Universidad de Girona ha bloqueado el rango de las IPs que Immuniweb utiliza para realizar la prueba de seguridad. Si se repitiera la misma prueba con la plantilla MEVN, el resultado sería el mismo a pesar de que unos pocos días atrás funcionaba bien.

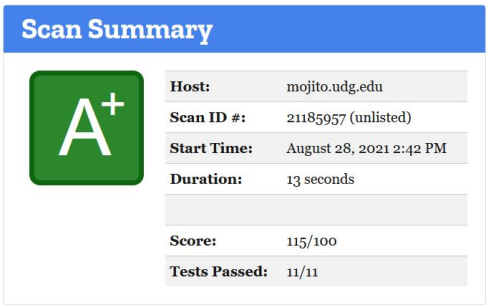

Figura 10.6: Puntuación de la configuración HTTPS con Mozilla Observatory en Laravel.

Figura 10.7: Puntuación de la política de seguridad de CSP con CSP Scanner en Laravel.

El código completo en este punto del desarrollo puede encontrarse todavía en el GitHub del autor[169] con la etiqueta «v0.2-alpha» porque no ha habido ningún cambio en esta subsección.

10.3.7. Asegurar la aplicación web con Apache

La protección de muchos de los ataques informáticos comentados en Laravel, y más que no se han podido comentar, se hace a través del servidor web. En concreto, hay dos módulos de Apache interesantes para defender la aplicación web ya desde el servidor web: ModSecurity y ModEvasive.

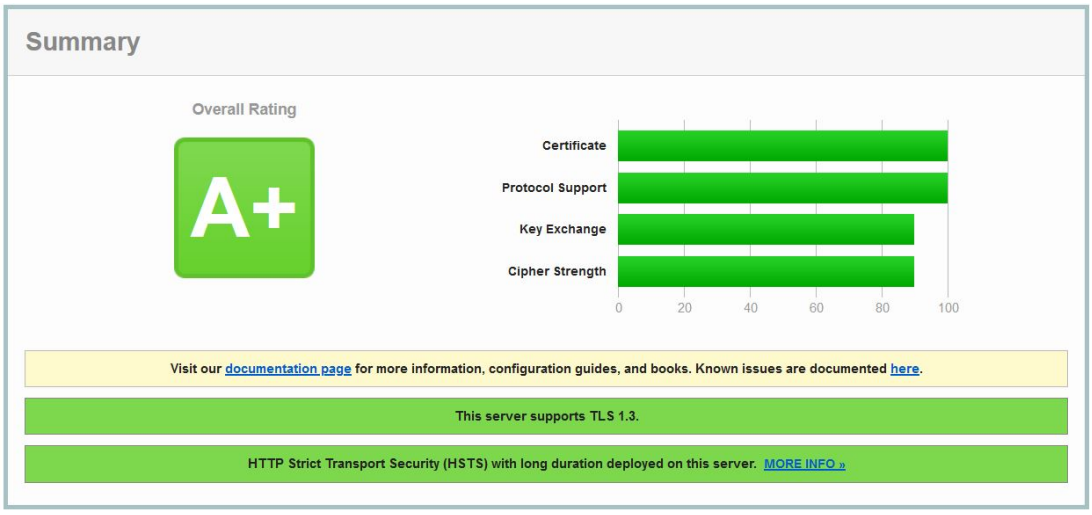

Figura 10.8: Puntuación de la configuración SSL/TLS con SSL Server Test en Laravel.

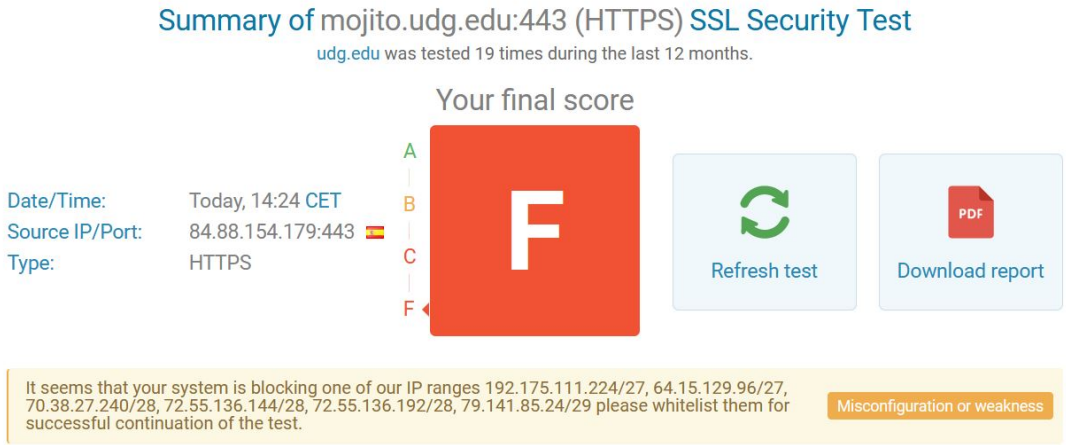

Figura 10.9: Puntuación de la configuración SSL/TLS con ImmuniWeb en Laravel.

ModSecurity

ModSecurity[179] es un módulo que permite la inspección de las solicitudes y respuestas HTTP de acuerdo a unas reglas predefinidas: esto permite proteger la aplicación web de ataques XSS, inyección de código, travesía de ruta,...

Para añadir ModSecurity al servidor Apache se deben seguir los pasos siguientes[180]:

1. Actualizar el sistema operativo.

```
$ sudo apt update && sudo apt upgrade
```

2. Instalar el módulo.

```
$ sudo apt install libapache2-mod-security2
```

3. Crear el fichero con la configuración del módulo a partir de la recomendada.

```
$ sudo mv /etc/modsecurity/modsecurity.conf-recommended
/etc/modsecurity/modsecurity.conf
```

4. Descargar las reglas recomendadas por OWASP[181]:

```
$ cd ~
$ git clone https://github.com/SpiderLabs/owasp-modsecurity-crs.git
```

5. Copiar el fichero de configuración y las normas:

```
$ cd ~/owasp-modsecurity-crs
$ sudo mv crs-setup.conf.example /etc/modsecurity/crs-setup.conf
$ sudo mv rules/ /etc/modsecurity/
```

6. Por defecto, la configuración del Apache debería de detectar el nuevo conjunto de reglas. De no ser así se debe: Abrir el fichero de configuración del módulo:

```
$ sudo nano /etc/apache2/mods-available/security2.conf
```

   Y añadir al final del fichero:

```
<IfModule security2_module>
   # Default Debian dir for modsecurity's persistent data
   SecDataDir /var/cache/modsecurity

   # Include all the *.conf files in /etc/modsecurity.
   # Keeping your local configuration in that directory
   # will allow for an easy upgrade of THIS file and
   # make your life easier
   IncludeOptional /etc/modsecurity/*.conf
   Include /etc/modsecurity/rules/*.conf
</IfModule>
```

7. Por último, reiniciar el servidor Apache:

```
$ sudo systemctl restart apache2
```

   Si bien, este último paso también puede dar problemas. De ser así, se recomienda utilizar el comando siguiente para probar la configuración Apache:

```
$ apachectl configtest
```

   Ya que durante la realización de la guía se han encontrado dos errores distintos: «ModSecurity: Failed to open the audit log file: /var/log/apache2/modsec_audit.log» y: «ModSecurity: Found another rule with the same id».

   El segundo error es más fácil de solucionar: solo debe eliminarse el fichero que se ha creado en el paso anterior: «/etc/apache2/mods-available/security2.conf»

Sin embargo, el primero puede ser un poco más complejo. Una solución, que al autor no le ha funcionado, es cambiar el propietario del fichero «/var/log/apache2/modsec_audit.log» a «www-data» y los permisos a 660[182]. Otra solución, es modificar la ubicación del fichero de registro definida en «/etc/modsecurity/modsecurity.conf» a un directorio que tenga permisos de escritura.

ModEvasive

El segundo módulo que se ha añadido a Apache para asegurar la aplicación en Laravel ha sido ModEvasive el cual permite proteger la aplicación frente ataques de denegación de servicio, distribuido o no, y ataques de fuerza bruta al restringir el número de peticiones que se pueden realizar, tanto por IP como en general.

Para la instalación y la configuración del módulo se han seguido los pasos siguientes[183]:

1. Actualizar el sistema operativo.

```
$ sudo apt update && sudo apt upgrade
```

2. Instalar el módulo.

```
$ sudo apt install libapache2-mod-evasive
```

Que solicita la configuración de un servidor de correo electrónico para enviar las posibles notificaciones. Para realizar la guía simplemente se ha escogido no hacerlo en ese momento ya que no se tenía un servidor de correo electrónico Postfix que añadir. Además, la instalación del módulo ya añade una configuración por defecto en el fichero «/etc/apache2/mods-enabled/evasive.conf» con el contenido siguiente comentado:

```
<IfModule mod_evasive20.c>
    #DOSHashTableSize    3097
    #DOSPageCount        2
    #DOSSiteCount        50
    #DOSPageInterval     1
    #DOSSiteInterval     1
    #DOSBlockingPeriod   10

    #DOSEmailNotify      you@yourdomain.com
    #DOSSystemCommand    "su - someuser -c '/sbin/... %s ...'"
    #DOSLogDir           "/var/log/mod_evasive"
</IfModule>
```

3. Descomentar al menos el primer grupo de directivas, explicadas más abajo.

DOSHashTableSize
: Tamaño de la tabla de *hashs*

DOSPageCount
: Número de peticiones permitidas a la misma URI por segundo e IP. Las direcciones IP que sobrepasen este límite serán bloqueadas.

DOSSiteCount
Número total de peticiones permitidas por la misma dirección IP.

DOSPageInterval
Intervalo en segundos en el que se recuerdan los accesos a una página concreta.

DOSSiteInterval
Intervalo en segundos en el que se recuerdan los accesos al sitio web en general.

DOSBlockingPeriod
Número de segundos en que las direcciones IP son bloqueadas después de superar uno de los dos límites anteriores.

DOSEmailNotify
Dirección de correo electrónico en el que se envían las notificaciones cuando una dirección IP es bloqueada.

DOSSystemCommand
Comando del sistema que es ejecutado cada vez que una dirección IP es bloqueada.

DOSLogDir
Ubicación del directorio del registro de ModEvasive.

El código completo en este punto del desarrollo puede encontrarse todavía en el GitHub del autor[169] con la etiqueta «v0.2-alpha» porque no ha habido ningún cambio en esta subsección.

10.3.8. Validación de las entradas

No se ha podido realizar la guía de la validación de las entradas en LAMP como si se ha hecho en MEVN.

10.4. Implementación *frontend* con Vue.js

Desafortunadamente, no se ha podido realizar la implementación del proyecto de interfaz gráfica de usuario con Vue.js para la entrega del proyecto de final de grado.

Capítulo 11

Conclusiones

La Fundación OWASP define dos listas con los controles y los vulnerabilidades más comunes en las aplicaciones web. Estos son el *OWASP Proactive Controls*[21] y *OWASP Top Ten Web Application Security Risks | OWASP*[20].

De estas veinte cuestiones se han tratado las siguientes doce en el proyecto:

- A1:2017-Injection: se han explicado los diferentes tipos de ataques de inyección de código y las diversas medidas de protección que el desarrollador puede aplicar en la sección 8.5.
- A3:2017-Sensitive Data Exposure: se ha explicado como almacenar los datos sensibles de forma segura y legal en la sección 7.4.
- A6:2017-Security Misconfiguration: se ha explicado la correcta configuración HTTP y del tratamiento de los errores en las secciones 7.1 y 7.3 respectivamente. Dos casos muy habituales de errores de configuración en las aplicaciones.
- A7:2017-Cross-Site Scripting XSS: se ha explicado este ataque, y sus posibles prevenciones y mitigaciones, en la sección 8.4.
- A10:2017-Insufficient Logging & Monitoring: se ha explicado esta debilidad en la sección 7.2.
- C1: Define Security Requirements: este es el objetivo principal del proyecto, ofrecer a los posibles desarrolladores web una serie de buenas prácticas y medidas de protección frente ciberataques para un desarrollo web seguro .
- C2: Leverage Security Frameworks and Libraries: aunque no se ha explicado este tema en ningún momento, sí que los *frameworks* escogidos y bibliotecas utilizados son mantenidos con regularidad y no se les conocen vulnerabilidades.
- C4: Encode and Escape Data: se ha explicado junto con el siguiente control y en diversos ataques como una protección frente ataques de inyección de código.
- C5: Validate All Inputs: se ha explicado la validación de entradas en la sección 7.5.
- C8: Protect Data Everywhere: se ha explicado el almacenamiento de datos sensibles en la sección 7.4.

- C9: Implement Security Logging and Monitoring: se ha explicado el registro de los eventos de la aplicación en la sección 7.2.
- C10: Handle All Errors and Exceptions: se ha explicado el tratamiento de errores en la sección 7.3.

A pesar de que aún faltan por tratar ocho de las veinte cuestiones mencionadas; que la plantilla MEVN aún se encuentra en un estado temprano; y que el desarrollo de la interfaz de usuario con Vue.js está todavía en la fase de diseño, la experiencia obtenida durante la realización de este proyecto ha sido sin duda positiva.

Además, el conocimiento adquirido, aunque todavía limitado, tanto en relación a la planificación y gestión de proyectos, como en el desarrollo web y la ciberseguridad será de gran importancia para el futuro profesional del autor porque a pesar de la mayor concienciación en todos los niveles de la importancia de la seguridad informática, aquellos que quieren aprovecharse de la ignorancia, la desidia o el descuido de los usuarios, de los programadores o administradores de sistemas están también cada vez más organizados y preparados.

Sin embargo, también se han cometido algunos errores durante la planificación y la realización del proyecto que han retrasado y reducido el número de buenas prácticas, ciberataques e implementaciones concretas que se han podido investigar y/o añadir a los dos *stacks* MEVN y LAMP.

El primer error, y seguramente el más grave, ha sido la estimación incorrecta del trabajo en horas de las tareas de investigación e implementación descritas en la tabla 4.1. Este error es debido a que en la estimación inicial no se tuvo en cuenta el tiempo necesario para la escritura de la documentación de las tareas individuales: un aspecto fundamental para este proyecto de final de grado, dado su carácter principalmente educativo y divulgativo. Un error que además, y a pesar de su apariencia inocua, ha causado una gran frustración al autor sobre el desarrollo del trabajo realizado.

Un segundo error, relacionado también con el presente documento, ha sido la falta de un estándar para la escritura del documento: por ejemplo, si utilizar cursiva, comillas españolas (« »), o comillas ingleses (“ ”) para los nombres de los programas, lenguajes o módulos. Un error que a menudo ha provocado retrasos innecesarios y la falta de coherencia en algunos aspectos de la documentación. Un error agravado además por el hecho de haber dedicado la sección 6.2 entera a la elección y definición de los estándares de Git, JavaScript y PHP.

Otro error de planificación cometido es la carencia de una hoja de ruta concreta con el orden de las tareas a realizar: este orden podría haber sido realizar primero los, ya mencionados, controles definidos por la Fundación OWASP o investigar los distintos métodos de prevención y mitigación de los principales peligros de las aplicaciones web.

Los dos últimos problemas que se ha encontrado al desarrollar la plantilla LAMP con Composer y Laravel han sido: primero, la falta de tiempo; y segundo, que Laravel ya es un *framework* más completo de lo esperado. No solo contiene ya por defecto la mayoría de buenas prácticas y protecciones explicadas, sino que añade muchas más. Sin embargo, no era práctico utilizar un *framework* más sencillo para las guías de implementación en LAMP, o simplemente utilizar PHP puro, ya que en realidad el proyecto que ofrece Laravel con su herramienta de instalación ya es, de hecho y salvando las distancias porque Laravel

es un *framework* en desarrollo desde el 2011 y con centenares de contribuidores, el objetivo final de este proyecto de final de grado: ofrecer a los desarrolladores una herramienta para acelerar la implantación de un sitio web sin descuidar la seguridad informática. Quizás hubiera sido mejor utilizar Laravel como inspiración y realizar un trabajo parecido en Node.js y Express.

Capítulo 12

Trabajo futuro

Las múltiples formas en las que se puede ampliar, mejorar o realizar un trabajo futuro sobre el proyecto de final de grado son:

1. Finalizar la recopilación de buenas prácticas y ciberataques, priorizando especialmente aquellos controles proactivos y riesgos de seguridad de las aplicaciones web definidos por la Fundación OWASP que no se han hecho o resuelto.
2. Adaptar las recopilaciones a formato Markdown para añadir una Wiki al proyecto de Github y así permitir la colaboración de terceros.
3. Mejorar el detalle y la calidad de las guías de implementación concretas con el *stack* MEVN para que sean accesibles a programadores novatos, autodidactas o sin experiencia en las diferentes tecnologías usadas.
4. Realizar nuevas guías para otros *stacks* como WISA, Windows, IIS, SQL Server y ASP.NET, o Spring Boot, máquina virtual de Java, Tomcat, Spring Data y Java.
5. Implementar un *framework* inspirado en Laravel con Node.js y Express que incorpore todas las buenas prácticas y protecciones estudiadas y por estudiar con un sitio web de muestra completamente configurable y facilitar su implantación con la utilización de tecnologías como Docker o Kubernetes.
6. Implementar una serie de pruebas para los diferentes *stacks* que los *pull requests* con nuevas funcionalidades deben pasar antes de ser considerados para la revisión.
7. La seguridad informática también es cosa de los usuarios: realizar una recopilación de consejos, guías y advertencias para los usuarios no iniciados en el tema.

Capítulo 13

Guía de contribución

13.1. Informe de errores

Utilice Github Issues para informar de errores específicos: el titulo y la descripción deben ser concisos y claros. Se recomienda utilizar el template que proporciona el propio Github para crear el *issue*. Crear un informe de un error es el primer paso para la solución. Sin embargo, no espere que se resuelva instantáneamente.

13.2. Preguntas de soporte

Utilice Github Discussions para realizar sus preguntas de soporte.

13.3. Petición y discusión de nuevas funcionalidades

Puede proponer nuevas funcionalidades o mejoras de funcionalidades existentes en Github Discussions del proyecto correspondiente.

13.4. Rama destino

Las ramas deben empezar y terminar en *main*, salvo aquellas que solucionen problemas urgentes presentes en la rama de producción, que empiezan en *stable* y terminan en *stable* y en *main*.

En caso de duda, pregunte en Github Discussions del proyecto correspondiente.

13.5. Estilo de código

Cardona sigue los estilos de código siguientes:

- JavaScript Standard Style para el código JavaScript.
- PHP Standard Recommendation para el código PHP.

Antes de marcar el *pull request* como *ready*, asegúrese de que ni «standard» ni «phpcs» encuentran algún problema de estilo con el código.

13.6. Código de conducta

El código de conducta de Cardona se deriva del código de conducta de Ruby. Cualquier violación del código de conducta se puede informar a Sergi Bergillos Pedraza:

- Los participantes serán tolerantes con los puntos de vista opuestos.
- Los participantes deben asegurarse de que su lenguaje y acciones estén libres de ataques personales y comentarios personales despectivos.
- Al interpretar las palabras y acciones de los demás, los participantes siempre deben asumir buenas intenciones.
- No se tolerará ningún comportamiento que pueda considerarse razonablemente como acoso.

Apéndice A

Artículos del Reglamento de Protección de los Datos Personales (RGPD) mencionados

A.1. Artículo 5. Principios relativos al tratamiento

1. Los datos personales serán:

   - tratados de manera lícita, leal y transparente en relación con el interesado («licitud, lealtad y transparencia»);
   - recogidos con fines determinados, explícitos y legítimos, y no serán tratados ulteriormente de manera incompatible con dichos fines; de acuerdo con el artículo 89, apartado 1, el tratamiento ulterior de los datos personales con fines de archivo en interés público, fines de investigación científica e histórica o fines estadísticos no se considerará incompatible con los fines iniciales («limitación de la finalidad»);
   - adecuados, pertinentes y limitados a lo necesario en relación con los fines para los que son tratados («minimización de datos»);
   - exactos y, si fuera necesario, actualizados; se adoptarán todas las medidas razonables para que se supriman o rectifiquen sin dilación los datos personales que sean inexactos con respecto a los fines para los que se tratan («exactitud»);
   - mantenidos de forma que se permita la identificación de los interesados durante no más tiempo del necesario para los fines del tratamiento de los datos personales; los datos personales podrán conservarse durante períodos más largos siempre que se traten exclusivamente con fines de archivo en interés público, fines de investigación científica o histórica o fines estadísticos, de conformidad con el artículo 89, apartado 1, sin perjuicio de la aplicación de las medidas técnicas y organizativas apropiadas que impone el presente Reglamento a fin de proteger los derechos y libertades del interesado («limitación del plazo de conservación»);
   - tratados de tal manera que se garantice una seguridad adecuada de los datos

personales, incluida la protección contra el tratamiento no autorizado o ilícito y contra su pérdida, destrucción o daño accidental, mediante la aplicación de medidas técnicas u organizativas apropiadas («integridad y confidencialidad»).

2. El responsable del tratamiento será responsable del cumplimiento de lo dispuesto en el apartado 1 y capaz de demostrarlo («responsabilidad proactiva»).

A.2. Artículo 9. Tratamiento de categorías especiales de datos personales

1. Quedan prohibidos el tratamiento de datos personales que revelen el origen étnico o racial, las opiniones políticas, las convicciones religiosas o filosóficas, o la afiliación sindical, y el tratamiento de datos genéticos, datos biométricos dirigidos a identificar de manera unívoca a una persona física, datos relativos a la salud o datos relativos a la vida sexual o las orientación sexuales de una persona física.

2. El apartado 1 no será de aplicación cuando concurra una de las circunstancias siguientes:

   - el interesado dio su consentimiento explícito para el tratamiento de dichos datos personales con uno o más de los fines especificados, excepto cuando el Derecho de la Unión o de los Estados miembros establezca que la prohibición mencionada en el apartado 1 no puede ser levantada por el interesado;
   - el tratamiento es necesario para el cumplimiento de obligaciones y el ejercicio de derechos específicos del responsable del tratamiento o del interesado en el ámbito del Derecho laboral y de la seguridad y protección social, en la medida en que así lo autorice el Derecho de la Unión de los Estados miembros o un convenio colectivo con arreglo al Derecho de los Estados miembros que establezca garantías adecuadas del respeto de los derechos fundamentales y de los intereses del interesado;
   - el tratamiento es necesario para proteger intereses vitales del interesado o de otra persona física, en el supuesto de que el interesado no esté capacitado, física o jurídicamente, para dar su consentimiento;
   - el tratamiento es efectuado, en el ámbito de sus actividades legítimas y con las debidas garantías, por una fundación, una asociación o cualquier otro organismo sin ánimo de lucro, cuya finalidad sea política, filosófica, religiosa o sindical, siempre que el tratamiento se refiera exclusivamente a los miembros actuales o antiguos de tales organismos o a personas que mantengan contactos regulares con ellos en relación con sus fines y siempre que los datos personales no se comuniquen fuera de ellos sin el consentimiento de los interesados;
   - el tratamiento se refiere a datos personales que el interesado ha hecho manifiestamente públicos;
   - el tratamiento es necesario para la formulación, el ejercicio o la defensa de reclamaciones o cuando los tribunales actúen en ejercicio de su función judicial;
   - el tratamiento es necesario por razones de un interés público esencial, sobre la base del Derecho de la Unión o de los Estados miembros, que debe ser

proporcional al objetivo perseguido, respetar en lo esencial el derecho a la protección de datos y establecer medidas adecuadas y específicas para proteger los intereses y derechos fundamentales del interesado;

- el tratamiento es necesario para fines de medicina preventiva o laboral, evaluación de la capacidad laboral del trabajador, diagnóstico médico, prestación de asistencia o tratamiento de tipo sanitario o social, o gestión de los sistemas y servicios de asistencia sanitaria y social, sobre la base del Derecho de la Unión o de los Estados miembros o en virtud de un contrato con un profesional sanitario y sin perjuicio de las condiciones y garantías contempladas en el apartado 3;
- el tratamiento es necesario por razones de interés público en el ámbito de la salud pública, como la protección frente a amenazas transfronterizas graves para la salud, o para garantizar elevados niveles de calidad y de seguridad de la asistencia sanitaria y de los medicamentos o productos sanitarios, sobre la base del Derecho de la Unión o de los Estados miembros que establezca medidas adecuadas y específicas para proteger los derechos y libertades del interesado, en particular el secreto profesional,
- el tratamiento es necesario con fines de archivo en interés público, fines de investigación científica o histórica o fines estadísticos, de conformidad con el artículo 89, apartado 1, sobre la base del Derecho de la Unión o de los Estados miembros, que debe ser proporcional al objetivo perseguido, respetar en lo esencial el derecho a la protección de datos y establecer medidas adecuadas y específicas para proteger los intereses y derechos fundamentales del interesado.

3. Los datos personales a que se refiere el apartado 1 podrán tratarse a los fines citados en el apartado 2, letra h), cuando su tratamiento sea realizado por un profesional sujeto a la obligación de secreto profesional, o bajo su responsabilidad, de acuerdo con el Derecho de la Unión o de los Estados miembros o con las normas establecidas por los organismos nacionales competentes, o por cualquier otra persona sujeta también a la obligación de secreto de acuerdo con el Derecho de la Unión o de los Estados miembros o de las normas establecidas por los organismos nacionales competentes.

4. Los Estados miembros podrán mantener o introducir condiciones adicionales, inclusive limitaciones, con respecto al tratamiento de datos genéticos, datos biométricos o datos relativos a la salud.

A.3. Artículo 25. Protección de datos desde el diseño y por defecto

1. Teniendo en cuenta el estado de la técnica, el coste de la aplicación y la naturaleza, ámbito, contexto y fines del tratamiento, así como los riesgos de diversa probabilidad y gravedad que entraña el tratamiento para los derechos y libertades de las personas físicas, el responsable del tratamiento aplicará, tanto en el momento de determinar los medios de tratamiento como en el momento del propio tratamiento, medidas técnicas y organizativas apropiadas, como la seudonimización, concebidas para aplicar de forma efectiva los principios de protección de datos, como la minimización

de datos, e integrar las garantías necesarias en el tratamiento, a fin de cumplir los requisitos del presente Reglamento y proteger los derechos de los interesados.

2. El responsable del tratamiento aplicará las medidas técnicas y organizativas apropiadas con miras a garantizar que, por defecto, solo sean objeto de tratamiento los datos personales que sean necesarios para cada uno de los fines específicos del tratamiento. Esta obligación se aplicará a la cantidad de datos personales recogidos, a la extensión de su tratamiento, a su plazo de conservación y a su accesibilidad. Tales medidas garantizarán en particular que, por defecto, los datos personales no sean accesibles, sin la intervención de la persona, a un número indeterminado de personas físicas.

3. Podrá utilizarse un mecanismo de certificación aprobado con arreglo al artículo 42 como elemento que acredite el cumplimiento de las obligaciones establecidas en los apartados 1 y 2 del presente artículo.

A.4. Artículo 32. Seguridad del tratamiento

1. Teniendo en cuenta el estado de la técnica, los costes de aplicación, y la naturaleza, el alcance, el contexto y los fines del tratamiento, así como riesgos de probabilidad y gravedad variables para los derechos y libertades de las personas físicas, el responsable y el encargado del tratamiento aplicarán medidas técnicas y organizativas apropiadas para garantizar un nivel de seguridad adecuado al riesgo, que en su caso incluya, entre otros:

   - la seudonimización y el cifrado de datos personales;
   - la capacidad de garantizar la confidencialidad, integridad, disponibilidad y resiliencia permanentes de los sistemas y servicios de tratamiento;
   - la capacidad de restaurar la disponibilidad y el acceso a los datos personales de forma rápida en caso de incidente físico o técnico;
   - un proceso de verificación, evaluación y valoración regulares de la eficacia de las medidas técnicas y organizativas para garantizar la seguridad del tratamiento.

2. Al evaluar la adecuación del nivel de seguridad se tendrán particularmente en cuenta los riesgos que presente el tratamiento de datos, en particular como consecuencia de la destrucción, pérdida o alteración accidental o ilícita de datos personales transmitidos, conservados o tratados de otra forma, o la comunicación o acceso no autorizados a dichos datos.

3. La adhesión a un código de conducta aprobado a tenor del artículo 40 o a un mecanismo de certificación aprobado a tenor del artículo 42 podrá servir de elemento para demostrar el cumplimiento de los requisitos establecidos en el apartado 1 del presente artículo.

4. El responsable y el encargado del tratamiento tomarán medidas para garantizar que cualquier persona que actúe bajo la autoridad del responsable o del encargado y tenga acceso a datos personales solo pueda tratar dichos datos siguiendo instrucciones del responsable, salvo que esté obligada a ello en virtud del Derecho de la Unión o de los Estados miembros.

Bibliografía

[1] Unión Europea. *Reglamento (UE) 2016/679 del Parlamento Europeo y del Consejo de 27 de abril de 2016 relativo a la protección de las personas físicas en lo que respecta al tratamiento de datos personales y a la libre circulación de estos datos y por el que se deroga la Directiva 95/46/CE (Reglamento general de protección de datos)*. Disponible en https://www.boe.es/buscar/doc.php?id=DOUE-L-2016-80807. Accedido por última vez el 13 de junio de 2021 (vid. págs. 5, 8, 24, 28, 54, 55).

[2] España. *Ley Orgánica 3/2018, de 5 de diciembre, de Protección de Datos Personales y garantía de los derechos digitales*. Disponible en https://www.boe.es/eli/es/lo/2018/12/05/3/con. Accedido por última vez el 13 de junio de 2021 (vid. págs. 5, 8, 28).

[3] España. *Ley 34/2002, de 11 de julio, de servicios de la sociedad de la información y de comercio electrónico*. Disponible en https://www.boe.es/eli/es/l/2002/07/11/34/con. Accedido por última vez el 13 de junio de 2021 (vid. págs. 5, 8, 24, 29).

[4] España, Ministerio de Educación y Formación Profesional. *El sistema universitario español*. Disponible en https://www.educacionyfp.gob.es/italia/dam/jcr:b53864d2-65a3-4526-abf4-61ef02f5be34/el-sistema-universitario-espa-ol2.pdf. Accedido por última vez el 3 de julio de 2021 (vid. pág. 8).

[5] Talent.com. *Salario para full Stack en España - Salario Medio*. Disponible en https://es.talent.com/salary?job=full+stack. Accedido por última vez el 3 de julio de 2021 (vid. pág. 8).

[6] David J. Anderson. *Agile managment for software enginnering: applying the theory of constraints for business results*. 2003 (vid. pág. 11).

[7] Sergi Bergillos Pedraza. *Tablero Kanban: La seguridad como punto de partida del desarrollo web*. Disponible en https://trello.com/b/YKGkYXTH/tablero-kanban-la-seguridad-como-punto-de-partida-del-desarrollo-web. Accedido por última vez el 1 de septiembre de 2021 (vid. pág. 12).

[8] Ponemon Institute LLC. *The Cost of Cybercrime*. Disponible en https://www.accenture.com/_acnmedia/PDF-96/Accenture-2019-Cost-of-Cybercrime-Study-Final.pdf. Accedido por última vez el 6 de julio de 2021 (vid. pág. 21).

[9] IBM Security. *Informe sobre el coste de una brecha de datos 2020*. IBM, 2020 (vid. págs. 21, 26, 48).

[10] Michel Cukier. *Study: Hackers Attack Every 39 Seconds | A. James Clark School of Engineering, University of Maryland*. Disponible en https://eng.umd.edu/news/story/study-hackers-attack-every-39-seconds. Accedido por última vez el 6 de julio de 2021 (vid. pág. 21).

[11] Steve Morgan. *Cybersecurity Talent Crunch To Create 3.5 Million Unfilled Jobs Globally By 2021*. Disponible en https://cybersecurityventures.com/jobs/. Accedido por última vez el 6 de julio de 2021 (vid. pág. 21).

[12] Erin Winick. *A cyber-skills shortage means students are being recruited to fight off hackers*. Disponible en https://www.technologyreview.com/2018/10/18/139708/a-cyber-skills-shortage-means-students-are-being-recruited-to-fight-off-hackers/. Accedido por última vez el 6 de julio de 2021 (vid. pág. 21).

[13] Liz Bradley. *Cyber Security Engineers Top List of Hottest Tech Jobs for 2019*. Disponible en https://www.globenewswire.com/news-release/2018/12/19/1669553/0/en/Cyber-Security-Engineers-Top-List-of-Hottest-Tech-Jobs-for-2019.html. Accedido por última vez el 6 de julio de 2021 (vid. pág. 21).

[14] Red Hat Inc. *¿Qué es una API?* Disponible en https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 22).

[15] Open Source Initiative. *The Open Source Definition*. Disponible en https://opensource.org/docs/osd. Accedido por última vez el 23 de junio de 2021 (vid. pág. 23).

[16] David Martínez. *Les cookies i tecnologies similars a les pàgines web: Grau de compliment normatiu*. Disponible en https://github.com/MaxiDave/gdpr-web-tracking-regulatory-compliance. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 24).

[17] Techopedia. *What is Web Development? - Definition from Techopedia*. Disponible en https://www.techopedia.com/definition/23889/web-development. Accedido por última vez el 3 de julio de 2021 (vid. pág. 24).

[18] Techopedia. *What is Software Framework? - Definition from Techopedia*. Disponible en https://www.techopedia.com/definition/14384/software-framework. Accedido por última vez el 11 de julio de 2021 (vid. pág. 25).

[19] OWASP Foundation, Inc. *OWASP Foundation | Open Source Foundation for Application Security*. Disponible en https://owasp.org/. Accedido por última vez el 4 de julio de 2021 (vid. pág. 25).

[20] OWASP Foundation, Inc. *OWASP Top Ten Web Application Security Risks | OWASP*. Disponible en https://owasp.org/www-project-top-ten/. Accedido por última vez el 4 de julio de 2021 (vid. págs. 25, 121).

[21] OWASP Contributors. *OWASP Proactive Controls*. Disponible en https://owasp.org/www-project-proactive-controls/. Accedido por última vez el 3 de agosto de 2021 (vid. págs. 26, 121).

[22] Agencia Española de Protección de Datos. *Survey on Device Fingerprinting*. Disponible en https://www.aepd.es/en/documento/estudio-fingerprinting-huella-digital-en.pdf. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 28).

[23] Agencia Española de Protección de Datos. *Guía sobre el uso de las cookies*. Disponible en https://www.aepd.es/sites/default/files/2020-07/guia-cookies.pdf. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 28).

[24] Inria reserchers. *AmIUnique: Learn how identifiable you are on the Internet*. Disponible en https://amiunique.org/about. Accedido por última vez el 13 de agosto de 2021 (vid. pág. 28).

[25] Wikipedia Contributors. *Modelo OSI - Wikipedia, la enciclopedia libre*. Disponible en https://es.wikipedia.org/wiki/Modelo_OSI. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 29).

[26] Cloudfare. *What is HTTP? | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/ddos/glossary/hypertext-transfer-protocol-http/. Accedido por última vez el 21 de julio de 2021 (vid. pág. 30).
[27] Mozilla Contributors. *HTTP | MDN*. Disponible en https://developer.mozilla.org/es/docs/Web/HTTP. Accedido por última vez el 21 de julio de 2021 (vid. pág. 30).
[28] Mozilla Contributors. *Métodos de petición HTTP - HTTP | MDN*. Disponible en https://developer.mozilla.org/es/docs/Web/HTTP/Methods. Accedido por última vez el 21 de julio de 2021 (vid. pág. 30).
[29] Mozilla Contributors. *HTTP headers - HTTP | MDN*. Disponible en https://developer.mozilla.org/es/docs/Web/HTTP/Headers. Accedido por última vez el 21 de julio de 2021 (vid. págs. 30, 72).
[30] Mozilla Contributors. *HTTP response status codes - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Status. Accedido por última vez el 21 de julio de 2021 (vid. pág. 30).
[31] Agencia Española de Protección de Datos. *Introducción al hash como técnica de seudonimización de datos personales*. Disponible en https://www.aepd.es/sites/default/files/2020-05/estudio-hash-anonimidad.pdf. Accedido por última vez el 4 de agosto de 2021 (vid. págs. 31, 58).
[32] Github Inc. *MIT License | Choose a License*. Disponible en https://choosealicense.com/licenses/mit/. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 33).
[33] Jeremy Helms. *Branching*. Disponible en https://gist.github.com/digitaljhelms/4287848. Accedido por última vez el 7 de julio de 2021 (vid. págs. 34, 35).
[34] Chris Beams. *How to Write a Git Commit Message*. Disponible en https://chris.beams.io/posts/git-commit/. Accedido por última vez el 8 de julio de 2021 (vid. pág. 36).
[35] Feross Aboukhadijeh. *JavaScript Standard Style*. Disponible en https://standardjs.com/. Accedido por última vez el 8 de julio de 2021 (vid. págs. 37, 38).
[36] capaj. *JavaScript standardjs styled snippets - Visual Studio Marketplace*. Disponible en https://marketplace.visualstudio.com/items?itemName=capaj.vscode-standardjs-snippets. Accedido por última vez el 8 de julio de 2021 (vid. pág. 37).
[37] capaj. *JavaScript standardjs styled snippets - Visual Studio Marketplace*. Disponible en https://marketplace.visualstudio.com/items?itemName=capaj.vscode-standardjs-snippets. Accedido por última vez el 8 de julio de 2021 (vid. pág. 37).
[38] Standard. *Standard JS - JavaScript Standard Style - Visual Studio Marketplace*. Disponible en https://marketplace.visualstudio.com/items?itemName=standard.vscode-standard. Accedido por última vez el 8 de julio de 2021 (vid. pág. 38).
[39] PHP Framework Interop Group. *PHP.FIG — PHP framework Interop Group - PHP-FIG*. Disponible en https://www.php-fig.org/. Accedido por última vez el 10 de julio de 2021 (vid. pág. 38).
[40] PHP Framework Interop Group. *PHP Standards Recommendations - PHP-FIG*. Disponible en https://www.php-fig.org/psr/. Accedido por última vez el 10 de julio de 2021 (vid. págs. 38, 111).

[41] Squiz Labs. *GitHub - squizlabs/PHP_ CodeSniffer*. Disponible en https://github.com/squizlabs/PHP_CodeSniffer. Accedido por última vez el 10 de julio de 2021 (vid. pág. 39).
[42] Samuel Hilson. *PHP Sniffer & Beautifier - Visual Studio Marketplace*. Disponible en https://marketplace.visualstudio.com/items?itemName=ValeryanM.vscode-phpsab. Accedido por última vez el 10 de julio de 2021 (vid. pág. 39).
[43] Nils Adermann y Jordi Boggiano. *Composer*. Disponible en https://getcomposer.org/. Accedido por última vez el 10 de julio de 2021 (vid. pág. 39).
[44] Brian Jackson. *Is PHP Dead? No! At Least Not According to PHP Usage Statistics*. Disponible en https://kinsta.com/blog/is-php-dead/. Accedido por última vez el 11 de julio de 2021 (vid. pág. 39).
[45] W3Techs. *Usage Statistics and Market Share of Server-side Programming Languages for Websites, July 2021*. Disponible en https://w3techs.com/technologies/overview/programming_language. Accedido por última vez el 11 de julio de 2021 (vid. págs. 39, 40).
[46] W3Techs. *Usage Statistics and Market Share of Web Servers, July 2021*. Disponible en https://w3techs.com/technologies/overview/web_server. Accedido por última vez el 11 de julio de 2021 (vid. pág. 40).
[47] Pattem Digital. *6 Benefits of Using LAMP As A Web Development Platform*. Disponible en https://pattemdigital.com/insight/benefits-of-lamp-stack. Accedido por última vez el 11 de julio de 2021 (vid. pág. 40).
[48] W3Techs. *Usage Statistics of Client-side Programming Languages for Websites, July 2021*. Disponible en https://w3techs.com/technologies/overview/client_side_language. Accedido por última vez el 12 de julio de 2021 (vid. pág. 40).
[49] www.json.org. *Introducción a JSON*. Disponible en https://www.json.org/json-es.html. Accedido por última vez el 20 de julio de 2021 (vid. pág. 40).
[50] IONOS. *Protege tu proyecto web del sslstrip - IONOS*. Disponible en https://www.ionos.es/digitalguide/hosting/cuestiones-tecnicas/protege-tu-proyecto-web-del-sslstrip/. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 43).
[51] Michal Pecánek. *What is HTTPS? Everything You Need to Know*. Disponible en https://ahrefs.com/blog/what-is-https/. Accedido por última vez el 13 de julio de 2021 (vid. pág. 43).
[52] The Free Software Foundation. *Guía de "Geneu Privacy Guard": Sistemas de cifrado simétrico*. Disponible en https://www.gnupg.org/gph/es/manual.html#AEN199. Accedido por última vez el 14 de julio de 2021 (vid. pág. 45).
[53] Qualys, Inc. *SSL and TLS Deployment Best Practices*. Disponible en https://github.com/ssllabs/research/wiki/SSL-and-TLS-Deployment-Best-Practices. Accedido por última vez el 13 de julio de 2021 (vid. págs. 45, 47).
[54] Mozilla Contributors. *Security/Server Side TLS - MozillaWiki*. Disponible en https://wiki.mozilla.org/Security/Server_Side_TLS. Accedido por última vez el 14 de julio de 2021 (vid. págs. 45, 47).
[55] Mozilla Foundation. *Mozilla SSL Configuration Generator*. Disponible en https://ssl-config.mozilla.org/. Accedido por última vez el 14 de julio de 2021 (vid. págs. 45, 114).
[56] Mozilla Contributors. *Web Security*. Disponible en https://infosec.mozilla.org/guidelines/web_security. Accedido por última vez el 14 de julio de 2021 (vid. págs. 46, 47).

[57] Mozilla Contributors. *<iframe>: el elemento Inline Frame - HTML: Lenguaje de etiquetas de hipertexto | MDN*. Disponible en https://developer.mozilla.org/es/docs/Web/HTML/Element/iframe. Accedido por última vez el 15 de julio de 2021 (vid. pág. 46).

[58] Mozilla Contributors. *MIME types (IANA media types) - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types. Accedido por última vez el 15 de julio de 2021 (vid. pág. 46).

[59] OWASP Foundation, Inc. *Transport Layer Protection - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html. Accedido por última vez el 14 de julio de 2021 (vid. pág. 47).

[60] OWASP Foundation, Inc. *HTTP Strict Transport Security - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/HTTP_Strict_Transport_Security_Cheat_Sheet.html. Accedido por última vez el 14 de julio de 2021 (vid. págs. 47, 98).

[61] Mozilla Contributors. *Content Security Poliy (CSP) - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/CSP. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[62] Mozilla Contributors. *Content-Security-Policy - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[63] Mozilla Contributors. *Content-Security-Policy-Report-Only - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Content-Security-Policy-Report-Only. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[64] OWASP Foundation, Inc. *Content Security Policy - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Content_Security_Policy_Cheat_Sheet.html. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[65] Qualys, Inc. *SSL Server Test (Powered by Qualys SSL Labs)*. Disponible en https://www.ssllabs.com/ssltest/. Accedido por última vez el 13 de julio de 2021 (vid. pág. 47).

[66] OWASP Foundation, Inc. *Transport Layer Protection: Test the server configuration - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Transport_Layer_Protection_Cheat_Sheet.html#test-the-server-configuration. Accedido por última vez el 14 de julio de 2021 (vid. pág. 47).

[67] Qualys, Inc. *SSL Server Rating Guide*. Disponible en https://github.com/ssllabs/research/wiki/SSL-Server-Rating-Guide. Accedido por última vez el 13 de julio de 2021 (vid. pág. 47).

[68] Mozilla Foundation. *Mozilla Observatory*. Disponible en https://observatory.mozilla.org/. Accedido por última vez el 14 de julio de 2021 (vid. pág. 47).

[69] Google Inc. *CSP Evaluator*. Disponible en https://csp-evaluator.withgoogle.com/. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[70] RapidSec. *CSP Scanner*. Disponible en https://cspscanner.com/. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 47).

[71] Karen Kent y Murugiah Souppaya. *Guide to Computer Security Log Management. Recomendations of the National Institue of Standards and Technology. Special Pu-*

blication 800-92. National Institute of Standards y Technology, Technology Administration, U.S. Department of Commerce, 2020 (vid. pág. 48).

[72] OWASP Foundation, Inc. *Logging - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Logging_Cheat_Sheet.html. Accedido por última vez el 21 de julio de 2021 (vid. págs. 48, 49, 89).

[73] Trent Mick. *Service logging in JSON with Bunyan | Node.js*. Disponible en https://nodejs.org/es/blog/module/service-logging-in-json-with-bunyan/. Accedido por última vez el 21 de julio de 2021 (vid. pág. 49).

[74] Graylog, Inc. *Industry Leading Log Management | Graylog*. Disponible en https://www.graylog.org/. Accedido por última vez el 20 de julio de 2021 (vid. pág. 50).

[75] Elasticsearch B.V. *El Elastic Stack: Elasticsearch, Kibana, Beats y Logstash | Elastic*. Disponible en https://www.elastic.co/es/elastic-stack/. Accedido por última vez el 20 de julio de 2021 (vid. pág. 50).

[76] Techopedia. *What is Error Handling? - Definition from Techopedia*. Disponible en https://www.techopedia.com/definition/16626/error-handling. Accedido por última vez el 29 de julio de 2021 (vid. pág. 50).

[77] Yan Babitski. *Getting error handling right*. Disponible en https://medium.com/swlh/getting-error-handling-right-9a1d39da0fa3. Accedido por última vez el 29 de julio de 2021 (vid. págs. 50-53).

[78] OWASP Foundation, Inc. *Error Handling - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Error_Handling_Cheat_Sheet.html. Accedido por última vez el 30 de julio de 2021 (vid. págs. 50, 54).

[79] Ramesh Dadatare. *How the Exception Handling Works in Java*. Disponible en https://www.javaguides.net/2018/08/how-exception-handling-works-in-java.html. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 52).

[80] Nottingham, M. and E. Wilde. *"Problem Details for HTTP APIs", RFC 7807, DOI 10.17487/RFC7807, March 2016*. Disponible en https://www.rfc-editor.org/info/rfc7807. Accedido por última vez el 29 de julio de 2021 (vid. pág. 53).

[81] PCI Security Standards Council, LLC. *Sitio oficial del Consejo sobre Normas de Seguridad de la PCI*. Disponible en https://es.pcisecuritystandards.org/minisite/env2/. Accedido por última vez el 2 de agosto de 2021 (vid. págs. 54, 101).

[82] OWASP Contributors. *C8: Protect Data Everywhere | OWASP*. Disponible en https://owasp.org/www-project-proactive-controls/v3/en/c8-protect-data-everywhere. Accedido por última vez el 3 de agosto de 2021 (vid. págs. 54-56).

[83] Agencia Española de Protección de Datos. *Guía de Privacidad desde el Diseño*. Disponible en https://www.aepd.es/es/documento/guia-privacidad-desde-diseno.pdf. Accedido por última vez el 4 de agosto de 2021 (vid. pág. 58).

[84] Agencia Española de Protección de Datos. *Guía de Protección de Datos por Defecto*. Disponible en https://www.aepd.es/sites/default/files/2020-10/guia-proteccion-datos-por-defecto.pdf. Accedido por última vez el 4 de agosto de 2021 (vid. pág. 58).

[85] European Union Agency For Cybersecurity. *Pseudonymisation techniques and best practices*. Disponible en https://www.enisa.europa.eu/publications/pseudonymisation-techniques-and-best-practices. Accedido por última vez el 18 de agosto de 2021 (vid. pág. 58).

[86] Agencia Española de Protección de Datos. *La K-anonimidad como medida de la privacidad*. Disponible en https://www.aepd.es/sites/default/files/2019-0

9/nota-tecnica-kanonimidad.pdf. Accedido por última vez el 18 de agosto de 2021 (vid. pág. 59).

[87] Agencia Española de Protección de Datos. *10 malentendidos relacionados con la anonimización*. Disponible en https://www.aepd.es/es/documento/10-malentendidos-anonimizacion.pdf. Accedido por última vez el 18 de agosto de 2021 (vid. pág. 59).

[88] National Institute of Standards and Technology. *Block Cipher Techniques | CSRC*. Disponible en https://csrc.nist.gov/projects/block-cipher-techniques. Accedido por última vez el 18 de agosto de 2021 (vid. pág. 59).

[89] National Institute of Standards and Technology. *Key Management | CSRC*. Disponible en https://csrc.nist.gov/projects/key-management/key-management-guidelines. Accedido por última vez el 19 de agosto de 2021 (vid. pág. 59).

[90] Justin Clarke. «Chapter 8 - Code-Level Defenses». En: *SQL Injection Attacks and Defense*. Ed. por Justin Clarke. Boston: Syngress, 2009, págs. 341-376. ISBN: 978-1-59749-424-3. DOI: https://doi.org/10.1016/B978-1-59749-424-3.00008-6. URL: https://www.sciencedirect.com/science/article/pii/B9781597494243000086 (vid. pág. 60).

[91] OWASP Contributors. *Input Validation | OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Input_Validation_Cheat_Sheet.html. Accedido por última vez el 8 de agosto de 2021 (vid. págs. 60, 61).

[92] OWASP Contributors. *C5: Validate All Inputs | OWASP*. Disponible en https://owasp.org/www-project-proactive-controls/v3/en/c5-validate-inputs. Accedido por última vez el 8 de agosto de 2021 (vid. págs. 60, 61).

[93] OWASP Contributors. *Regular expression Denial of Service - ReDoS | OWASP*. Disponible en https://owasp.org/www-community/attacks/Regular_expression_Denial_of_Service_-_ReDoS. Accedido por última vez el 8 de agosto de 2021 (vid. págs. 61, 64, 103).

[94] json-schema-org. *JSON Schema | The home of JSON Schema*. Disponible en https://json-schema.org/. Accedido por última vez el 8 de agosto de 2021 (vid. págs. 61, 103).

[95] IETF Trust. *Regular Expressions for Email Addresses*. Disponible en https://tools.ietf.org/id/draft-seantek-mail-regexen-03.html#rfc.section.3. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 61).

[96] Cisco Inc. *Cyber Attack - What Are Common Cyberthreats?* Disponible en https://www.cisco.com/c/en/us/products/security/common-cyberattacks.html#~types-of-cyber-attacks. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 63).

[97] Amazon Web Services. *Qué es un ataque DDOS y cómo proteger su sitio contra uno*. Disponible en https://aws.amazon.com/es/shield/ddos-attack-protection. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 63).

[98] Panda Security. *Ataques de Red: DoS y DDoS - Panda Security*. Disponible en https://www.pandasecurity.com/es/security-info/network-attacks/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 64).

[99] AWS Shield. *Thread Landscape Report - Q1 2020*. Disponible en https://aws-shield-tlr.s3.amazonaws.com/2020-Q1_AWS_Shield_TLR.pdf/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 64).

[100] Cloudfare Inc. *What is a Distributed Denial-of-Service (DDoS) Attack? | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/ddos/what-is-a-ddos-attack/. Accedido por última vez el 5 de agosto de 2021 (vid. págs. 64, 65).

[101] Cloudfare Inc. *What is DDoS blackhole routing? | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/ddos/glossary/ddos-blackhole-routing/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 64).

[102] Cloudfare Inc. *What is reverse proxy? | Proxy servers explained | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/cdn/glossary/reverse-proxy/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 64).

[103] F5, Inc. *What is Load Balancing? How Load Balancers Work*. Disponible en https://www.nginx.com/resources/glossary/load-balancing/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 65).

[104] AWS Shield. *Qué es un ataque DDOS y cómo proteger su sitio contra uno*. Disponible en https://aws.amazon.com/es/shield/ddos-attack-protection/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 65).

[105] Cloudfare Inc. *¿Qué es un WAF? | Explicación de Web Application Firewall | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/ddos/glossary/web-application-firewall-waf/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 65).

[106] Cloudfare Inc. *What is a CDN? | How do CDNs work? | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/cdn/what-is-a-cdn/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 65).

[107] Cloudfare Inc. *What is Anycast? | How does Anycast work? | Cloudfare*. Disponible en https://www.cloudflare.com/es-es/learning/cdn/glossary/anycast-network/. Accedido por última vez el 5 de agosto de 2021 (vid. pág. 66).

[108] Awake Security. *Credential Theft Definition & Examples | Awake Security*. Disponible en https://awakesecurity.com/glossary/credential-theft/. Accedido por última vez el 12 de agosto de 2021 (vid. pág. 66).

[109] Alex Weinert. *Your Pa$$word doesn't matter - Microsoft Tech Community*. Disponible en https://techcommunity.microsoft.com/t5/azure-active-directory-identity/your-pa-word-doesn-t-matter/ba-p/731984. Accedido por última vez el 12 de agosto de 2021 (vid. págs. 66, 67).

[110] Microsoft. *Introducción a Multi-Factor Authentication de Azure AD | Microsoft Docs*. Disponible en https://docs.microsoft.com/es-es/azure/active-directory/authentication/concept-mfa-howitworks. Accedido por última vez el 12 de agosto de 2021 (vid. pág. 67).

[111] Alex Weinert. *All your credes are belong to us! - Microsoft Tech Community*. Disponible en https://techcommunity.microsoft.com/t5/azure-active-directory-identity/all-your-creds-are-belong-to-us/ba-p/855124. Accedido por última vez el 13 de agosto de 2021 (vid. pág. 67).

[112] AbuseIPDB. *AbuseIPDB - IP addresses abuse reports - Making the Internet safer, one IP at a time*. Disponible en https://www.abuseipdb.com/. Accedido por última vez el 13 de agosto de 2021 (vid. pág. 68).

[113] Troy Hunt. *Have I Been Pwned: Pwned Passwords*. Disponible en https://haveibeenpwned.com/. Accedido por última vez el 13 de agosto de 2021 (vid. pág. 68).

[114] Bernard Meyer. *Most common passwords: latest 2021 statistics*. Disponible en https://cybernews.com/best-password-managers/most-common-passwords/. Accedido por última vez el 13 de agosto de 2021 (vid. pág. 68).

[115] OWASP Contributors. *Credential Stuffing Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Credential_Stuffing_Prevention_Cheat_Sheet.html. Accedido por última vez el 12 de agosto de 2021 (vid. pág. 68).

[116] OWASP Contributors. *Cross Site Request Forgery (CSRF) | OWASP Foundation*. Disponible en https://owasp.org/www-community/attacks/csrf. Accedido por última vez el 17 de agosto de 2021 (vid. págs. 69, 70).

[117] Microsoft. *Prevent Cross-Site Request Forgery (XSRF/CSRF) attacks in ASP.NET Core | Microsoft Docs*. Disponible en https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery. Accedido por última vez el 17 de agosto de 2021 (vid. págs. 69, 70).

[118] Laravel. *CSRF Protection - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/csrf. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 70).

[119] Django. *Cross Site Request Forgery protection | Django documentation | Django*. Disponible en https://docs.djangoproject.com/en/3.2/ref/csrf/. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 70).

[120] Daniel Jimenez Garcia. *ASP.NET Core CSRF defence with Antiforgery*. Disponible en https://www.dotnetcurry.com/aspnet/1343/aspnet-core-csrf-antiforgery-token. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 70).

[121] CSURF Contributors. *CSRF token middleware*. Disponible en https://github.com/expressjs/csurf. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 70).

[122] Port Swigger. *CSRF tokens | Web Security Academy*. Disponible en https://portswigger.net/web-security/csrf/tokens. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 70).

[123] David Johansson. *Bypassing CSRF Protections. A Double Defeat of the Double-Submit Cookie Pattern*. Disponible en https://owasp.org/www-pdf-archive/David_Johansson-Double_Defeat_of_Double-Submit_Cookie.pdf. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 71).

[124] Wikipedia Contributors. *HMAC - Wikipedia, la enciclopedia libre*. Disponible en https://es.wikipedia.org/wiki/HMAC. Accedido por última vez el 17 de agosto de 2021 (vid. pág. 71).

[125] Mozilla Contributors. *Set-Cookie - HTTP | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/Set-Cookie. Accedido por última vez el 17 de agosto de 2021 (vid. págs. 71, 73).

[126] OWASP Contributors. *Cross-Site Scripting (XSS) Software Attack | OWASP Foundation*. Disponible en https://owasp.org/www-community/attacks/xss/. Accedido por última vez el 19 de agosto de 2021 (vid. págs. 72, 73).

[127] OWASP Contributors. *A7:2007-Cross-Site Scripting (XSS) | OWASP*. Disponible en https://owasp.org/www-project-top-ten/2017/A7_2017-Cross-Site_Scripting_(XSS). Accedido por última vez el 19 de agosto de 2021 (vid. pág. 72).

[128] OWASP Contributors. *XSS Filter Evasion Cheat Sheet - OWASP*. Disponible en https://owasp.org/www-community/xss-filter-evasion-cheatsheet. Accedido por última vez el 20 de agosto de 2021 (vid. pág. 73).

[129] OWASP Contributors. *Cross Site Scripting Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Cross

_Site_Scripting_Prevention_Cheat_Sheet.html. Accedido por última vez el 19 de agosto de 2021 (vid. pág. 73).
[130] OWASP Contributors. *Code Injection Software Attack | OWASP Foundation*. Disponible en https://owasp.org/www-community/attacks/Code_Injection. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 74).
[131] OWASP Contributors. *A1:2017-Injection | OWASP*. Disponible en https://owasp.org/www-project-top-ten/2017/A1_2017-Injection. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 74).
[132] OWASP Contributors. *Injection Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_Cheat_Sheet.html. Accedido por última vez el 31 de agosto de 2021 (vid. págs. 74, 75).
[133] The PHP Group. *PHP: Prepared Statements - Manual*. Disponible en https://www.php.net/manual/en/mysqli.quickstart.prepared-statements.php. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[134] Node.js Contributors. *Child process | Node.js v16.8.0 Documentation*. Disponible en https://nodejs.org/api/child_process.html. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[135] OWASP Contributors. *SQL Injection Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/SQL_Injection_Prevention_Cheat_Sheet.html. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[136] OWASP Contributors. *LDAP Injection Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/LDAP_Injection_Prevention_Cheat_Sheet.html. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[137] OWASP Contributors. *OS Command Injection Prevention - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/OS_Command_Injection_Defense_Cheat_Sheet.html. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[138] OWASP Contributors. *Injection Prevention in Java - OWASP Cheat Sheet Series*. Disponible en https://cheatsheetseries.owasp.org/cheatsheets/Injection_Prevention_in_Java_Cheat_Sheet.html. Accedido por última vez el 31 de agosto de 2021 (vid. pág. 75).
[139] Chris Richardson. *What are microservices?* Disponible en https://microservices.io/. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 78).
[140] GraphQL Foundation. *GraphQL | A query language for your API*. Disponible en https://graphql.org/. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 79).
[141] Sergi Bergillos Pedraza. *Cardona API · Apiary*. Disponible en https://cardona1.docs.apiary.io/. Accedido por última vez el 1 de setiembre de 2021 (vid. pág. 79).
[142] Electronic Frontier Foundation. *Certbot*. Disponible en https://certbot.eff.org/. Accedido por última vez el 1 de agosto de 2021 (vid. pág. 83).
[143] Maud Nalpas. *When to use HTTPS for local development*. Disponible en https://web.dev/when-to-use-local-https/. Accedido por última vez el 1 de agosto de 2021 (vid. pág. 84).

[144] Maud Nalpas. *How to use HTTPS for local development*. Disponible en https://web.dev/how-to-use-local-https/. Accedido por última vez el 1 de agosto de 2021 (vid. pág. 84).
[145] nvm contributors. *Node Version Manager*. Disponible en https://github.com/nvm-sh/nvm. Accedido por última vez el 22 de julio de 2021 (vid. pág. 85).
[146] OpenJS Foundation. *Releases | Node.js*. Disponible en https://nodejs.org/en/about/releases/. Accedido por última vez el 22 de julio de 2021 (vid. pág. 85).
[147] Sam Quinn. *Bulletproof node.js project architecture*. Disponible en https://softwareontheroad.com/ideal-nodejs-project-structure/?utm_source=github&utm_medium=readme. Accedido por última vez el 22 de julio de 2021 (vid. pág. 86).
[148] OpenJS Foundation. *Performance Best Practices Using Express in Production*. Disponible en https://expressjs.com/en/advanced/best-practice-performance.html. Accedido por última vez el 22 de julio de 2021 (vid. pág. 86).
[149] Node Best Practices Contributors. *Separate Express app and server*. Disponible en https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/projectstructre/separateexpress.md. Accedido por última vez el 27 de julio de 2021 (vid. pág. 87).
[150] Sergi Bergillos Pedraza. *cardona-node is the server-side application of the Cardona MEVN stack*. Disponible en https://github.com/SBergillos/cardona-node. Accedido por última vez el 22 de julio de 2021 (vid. págs. 88, 92, 97, 102, 103, 106).
[151] Node Best Practices Contributors. *Delegate anything possible (e.g. static content, gzip) to a reverse proxy*. Disponible en https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/production/delegatetoproxy.md. Accedido por última vez el 22 de agosto de 2021 (vid. pág. 88).
[152] Express Contributors. *Writing middleware for use in Express apps*. Disponible en https://expressjs.com/en/guide/writing-middleware.html. Accedido por última vez el 22 de agosto de 2021 (vid. pág. 88).
[153] Express Contributors. *Using Express middleware*. Disponible en https://expressjs.com/en/guide/using-middleware.html. Accedido por última vez el 22 de agosto de 2021 (vid. pág. 88).
[154] Node.js Contributors. *Async hooks | Node.js v14.17.3 Documentation*. Disponible en https://nodejs.org/docs/latest-v14.x/api/async_hooks.html. Accedido por última vez el 27 de julio de 2021 (vid. pág. 90).
[155] Node Best Practices Contributors. *Assign TransactionId to each log statement*. Disponible en https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/production/assigntransactionid.md. Accedido por última vez el 27 de julio de 2021 (vid. pág. 90).
[156] Pino Contributors. *API | pino*. Disponible en https://getpino.io/#/docs/api?id=base-object. Accedido por última vez el 27 de julio de 2021 (vid. págs. 90, 91).
[157] Node Best Practices Contributors. *Your application code should not handle log routing*. Disponible en https://github.com/goldbergyoni/nodebestpractices/blob/master/sections/production/logrouting.md. Accedido por última vez el 27 de julio de 2021 (vid. pág. 92).
[158] Adam Wiggins. *The Twelve-Factor App. XI Logs*. Disponible en https://12factor.net/logs. Accedido por última vez el 27 de julio de 2021 (vid. pág. 92).

[159] Joyent Inc. *Error Handling in Node.js*. Disponible en https://www.joyent.com/node-js/production/design/errors. Accedido por última vez el 31 de julio de 2021 (vid. págs. 92, 97).
[160] Express Contributors. *Express error handling*. Disponible en https://expressjs.com/en/guide/error-handling.html. Accedido por última vez el 30 de julio de 2021 (vid. pág. 93).
[161] Mozilla Contributors. *Promise - JavaScript | MDN*. Disponible en https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise. Accedido por última vez el 30 de julio de 2021 (vid. pág. 93).
[162] Helmet Contributors. *Helmet. Express.js security with HTTP headers.* Disponible en https://helmetjs.github.io/. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 98).
[163] Node.js Contributors. *TLS (SSL) | Node.js v14.17.4 Documentation*. Disponible en https://nodejs.org/docs/latest-v14.x/api/tls.html. Accedido por última vez el 2 de agosto de 2021 (vid. págs. 99, 101, 103).
[164] Juliana De Groot. *What is HIPAA Compliance? | Digital Guardian*. Disponible en https://digitalguardian.com/blog/what-hipaa-compliance. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 101).
[165] Nate Lord. *What is NIST Compliance? | Digital Guardian*. Disponible en https://digitalguardian.com/blog/what-nist-compliance. Accedido por última vez el 2 de agosto de 2021 (vid. pág. 101).
[166] Nathan Friedly. *express-rate-limiter-flexible: Basic rate-limiting middleware for Express*. Disponible en https://github.com/nfriedly/express-rate-limit. Accedido por última vez el 7 de agosto de 2021 (vid. pág. 102).
[167] animir. *node-rate-limiter-flexible: Node.js rate limit requests by key with atomic increments in single process or distributed environment*. Disponible en https://github.com/animir/node-rate-limiter-flexible. Accedido por última vez el 7 de agosto de 2021 (vid. pág. 102).
[168] Sideway Inc. *The most powerful data validation library for JS*. Disponible en https://github.com/sideway/joi. Accedido por última vez el 8 de agosto de 2021 (vid. pág. 103).
[169] Sergi Bergillos Pedraza. *cardona-node is the server-side application of the Cardona MEVN stack*. Disponible en https://github.com/SBergillos/cardona-lamp. Accedido por última vez el 21 de agosto de 2021 (vid. págs. 109, 113, 115, 119).
[170] Laravel Contributors. *Request Lifecycle - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/lifecycle. Accedido por última vez el 21 de agosto de 2021 (vid. pág. 109).
[171] Laravel Contributors. *Configuration - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/configuration. Accedido por última vez el 21 de agosto de 2021 (vid. pág. 109).
[172] Laravel Contributors. *Directory Structure - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/structure. Accedido por última vez el 21 de agosto de 2021 (vid. pág. 109).
[173] Laravel Contributors. *Service Providers - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/providers. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 109).

[174] Laravel Contributors. *Middleware - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/middleware. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 109).
[175] Laravel Contributors. *Views - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/configuration. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 109).
[176] Jordi Boggiano. *Monolog: Sends your logs to files, sockets, inboxes, databases and various web services*. Disponible en https://github.com/Seldaek/monolog. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 111).
[177] Laravel Contributors. *Logging - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/logging. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 112).
[178] Laravel Contributors. *Error Handling - Laravel - The PHP Framework For Web Artisans*. Disponible en https://laravel.com/docs/8.x/errors. Accedido por última vez el 26 de agosto de 2021 (vid. pág. 113).
[179] SpiderLabs. *SpiderLabs/ModSecurity*. Disponible en https://github.com/SpiderLabs/ModSecurity. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 116).
[180] idroot. *How To Install ModSecurity Apache on Ubuntu 20.04 LTS*. Disponible en https://idroot.us/install-modsecurity-apache-ubuntu-20-04/. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 116).
[181] OWASP Contributors. *OWASP ModSecurity Core Rule Set*. Disponible en https://owasp.org/www-project-modsecurity-core-rule-set/. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 117).
[182] kagulik. *ModSecurity fails in Ubuntu 16.04*. Disponible en https://github.com/SpiderLabs/ModSecurity/issues/1679. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 118).
[183] Hitesh Jethva. *How to Install and Configure ModEvasive with Apache on Ubuntu 18.04*. Disponible en https://www.atlantic.net/vps-hosting/how-to-install-and-configure-modevasive-with-apache-on-ubuntu-18-04/. Accedido por última vez el 30 de agosto de 2021 (vid. pág. 118).