UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica

Universitat Politècnica de València

# Desarrollo de un portal web para administrar una red de distribución de contenidos de Stackpath

Trabajo Fin de Grado

**Grado en Ingeniería Informática**

**Autor**: Aarón Martín Oviedo

**Tutor**: Julio Pons Terol

Curso 2021-2022

# Resumen

Este Trabajo de Fin de Grado muestra la construcción de una aplicación o herramienta integrada con la plataforma de servicios en la nube Stackpath, la cual pretende resolver la dificultad de asignar uno o varios dominios administrados en la cuenta a clientes de la empresa, siendo capaces de poder gestionar ciertas opciones de la configuración del dominio u observar estadísticas de este. A través de lenguajes y tecnologías web utilizadas hoy en día, se obtiene el resultado del ejercicio propuesto, adquiriendo nuevos conocimientos y experiencias.

**Palabras clave:** aplicación, sistema, herramienta, Stackpath.

# Resum

Este Treball de Fi de Grau mostra la construcció d'una aplicació o ferramenta integrada amb la plataforma de servicis en la núvol Stackpath, la qual pretén resoldre la dificultat d'assignar un o més dominis administrats en el compte a clients de l'empresa, sent capaços de poder gestionar certes opcions de la configuració del domini o observar estadístiques d'aquest. A través de llenguatges i tecnologies web utilitzades hui en dia, s'obté el resultat de l'exercici proposat, adquirint nous coneixements i experiències.

**Palabras clave:** aplicació, sistema, ferramenta, Stackpath.

# Abstract

This Final Degree Project shows the construction of an application or tool integrated with the cloud services platform Stackpath, which aims to solve the difficulty of assigning one or more domains managed in the account to customers of the company, being able to manage certain options of the domain configuration or observe statistics of this. Through languages and web technologies used today, the result of the proposed exercise is obtained, acquiring new knowledge and experience.

**Keywords:** application, system, tool, Stackpath.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE ILUSTRACIONES

# ÍNDICE DE TABLAS

# 1. INTRODUCCIÓN

En la actualidad, existen numerosas plataformas encargadas de ofrecer gran cantidad de servicios en la nube a aquellos usuarios que no deseen administrar el proceso de construcción o despliegue de una aplicación web. De esta forma se evita que el usuario tenga que centrarse en la implementación personalizada de tareas que pertenecen globalmente a un proceso común.

Asimismo, con la utilización de estas plataformas, se puede conseguir que los costes que se generen sean menores gracias al Cloud Computing, dejando atrás las dificultades del mantenimiento, la resolución de incidencias, problemas de seguridad o costes externos, como pueden ser la electricidad o los lugares geográficos donde realizar la instalación.

No obstante, cabe destacar que utilizar estos servicios no solo aportan ventajas. Es importante tener en cuenta que cada servicio ofrece un conjunto de herramientas diferente como, por ejemplo, la administración de accesos a usuarios que una empresa quiera configurar para los recursos de sus clientes, por lo que es necesario hacer un análisis previo de las necesidades que se tienen y qué servicio las cubren.

En el caso de estudio de este Trabajo de Fin de Grado se trabajará con la plataforma de computación en la nube Stackpath, una plataforma estadounidense que ofrece servicios a gran escala como redes de distribución de contenido (CDN), firewall de aplicaciones web (WAF), ejecución de código sin servidor (Serverless Scripting), entre otros… Gracias a estos servicios, existen empresas encargadas de la gestión de los recursos de terceras, a través de un usuario en dicha plataforma. Sin embargo, estas terceras empresas carecen de acceso a los recursos sobre la plataforma, y es por ello que nace la necesidad de proporcionarles un portal desde el que poder configurar sus recursos.

## 1.1 Motivación

A lo largo del grado he logrado darme cuenta de qué es lo que verdaderamente me llena de satisfacción de este mundo. Mi pasión tanto por el desarrollo de aplicaciones (o portales web) como por la seguridad y mantenimiento de estas ha aumentado a lo largo del último año, por lo que a falta de un trabajo de fin de grado público que llenara estas necesidades, decidí aceptar la propuesta de mi tutor.

## 1.2 Objetivos

El principal objetivo de este Trabajo de Fin de Grado es desarrollar una aplicación que a través de la API de Stackpath permita a un cliente acceder solamente a los ajustes de su dominio. Dicho objetivo se puede subdividir a su vez en dos subobjetivos:

- La creación de un portal para la administración de usuarios locales, los cuales se almacenen en una base de datos y se relacionen directamente con el dominio asociado.

- La creación de un portal para la visualización de métricas y administración de la caché del CDN por parte de los usuarios locales.

Como objetivos adicionales, se busca que el portal sea escalable en el tiempo pudiendo añadir nuevas funcionalidades de una forma sencilla, aprender nuevas tecnologías de desarrollo web y aplicarlas sobre la aplicación.

## 1.3 Metodología

En cuanto al método de trabajo, se ha optado por una primera etapa de análisis para identificar los elementos arquitectónicos a través de la especificación de requisitos funcionales y no funcionales que se utilizan en la siguiente fase, la del diseño de la infraestructura, en la cual se desarrollan los esquemas y el flujo que se debe seguir la fase de desarrollo. Posteriormente, se realizarán un conjunto de pruebas para determinar el estado final de la herramienta a través de pruebas de caja negra. Por último, se presentarán una serie de conclusiones obtenidas del resultado del ejercicio.

## 1.4 Estructura

Para descomponer lo que se va a tratar en esta memoria de forma que el lector pueda conocer brevemente qué aspecto se va a tratar en cada sección se desglosa en los siguientes puntos un breve resumen por capítulo:

- En el capítulo **2**. **Estado del arte** se define el contexto en el que se desenvuelve la herramienta tratada en este trabajo y una referencia a un producto que ofrece funcionalidades parecidas a la finalidad de este trabajo, llevado a cabo por la empresa *Amazon Web Services* (AWS).
- En el capítulo **3**. **Análisis** se estudia la seguridad, los requerimientos de la aplicación y el problema en cuestión que conlleva la realización de este trabajo.
- En el capítulo **4**. **Diseño de la solución** se documenta el proceso que se va a seguir para el desarrollo de los diferentes componentes de la aplicación. Se explican también las tecnologías que se utilizarán el proceso de desarrollo.
- En el capítulo **5**. **Desarrollo de la solución** se comenta el proceso que se ha llevado a cabo para la realización de la aplicación, tanto la parte con la que el usuario final puede interactuar como el mecanismo interno que hace posible el intercambio de datos.
- **6**. **Implantación y pruebas** es el capítulo donde se pone a prueba la aplicación y se habla de los resultados obtenidos tras hacer múltiples operaciones.
- En el capítulo de **7**. **Conclusiones** se resume la información general que se ha obtenido como resultado del trabajo realizado.
- Finalmente se expone una **8**. **Bibliografía** de donde se ha extraído información y se ha llevado una investigación sobre cada tema tratado.

En esta memoria pueden aparecer abreviaturas o términos desconocidos para el lector, por lo que se recomienda, en primera instancia, revisar el glosario adjunto al final de este documento.

# 2. ESTADO DEL ARTE

## 2.1 Historia

El auge de las telecomunicaciones a finales de los años 50 y principios de los 60 provocó un cambio drástico en la forma en la que los seres humanos nos comunicamos hoy en día. A raíz de una investigación llevada a cabo por una agencia del ministerio de defensa de los Estados Unidos, surge la primera red de computadores y el principio de una nueva estructura globalizada de intercambio de información con el nombre de ARPANET.

Dos decadas más tarde se define el modelo OSI (*Open Systems Interconnection*), un conjunto de capas que permiten el intercambio de información sin dificultades entre diferentes sistemas. La primera de ellas, la capa física, define los canales físicos que conectan varios sistemas, como los cables o el aire, y la forma en la que se transmite por estos canales. La segunda capa o nivel de enlace de datos, se encarga principalmente de las direcciones físicas, el control de acceso a los canales y la detección de errores a través de tramas (segmentos de los datos transferidos por medio de paquetes) . En la t ercera capa o nivel de red se identifican las rutas existentes en una o varias redes con el fin de encaminar los paquetes de datos por la red. La cuarta capa es la de transporte, la cual tiene como finalidad transportar los segmetos del origen al destino. Finalmente, la última capa corresponde al nivel de aplicación, la cual ofrece acceso a todas las capas anteriores a las aplicaciones que quieran hacer uso de ellas para el intercambio de datos.

A partir de la definición de este modelo y de los diferentes protocolos de que operan en cada capa, surge a principios de los años 90 la denominada World Wide Web (WWW) o Web, un sistema de nodos interconectados a través de la red en el cual se comparten datos que. Este sistema se ha ido desarrollando a lo largo de los años hasta la actualidad con el fin de obtener un sistema cada vez más globalizado, con una mayor cantidad de nodos y funcionalidades. El método empleado para servir el contenido a los usuarios era a través de los navegadores, los cuales obtenían los recursos que se habían solicitado y los procesaban. Al principio, estos recursos eran almacenados como aplicaciones o portales web en servidores locales por empresas o desarrolladores de contenido, lo que proporcionaba ventajas como la posibilidad del mantenimiento de la infraestructura donde se desplegaba ; sin embargo, las desventajas que este tipo de almacenamiento tenía eran significantes: elevados costes, dificultad para acceder a los recursos, falta de replicación de los datos en caso de catástrofe en el servidor… A causa de estos problemas, se diseñó un nuevo tipo de almacenamiento que solucionaría las desventajas que presentaba el almacenamiento tradicional y aportara nuevas funcionalidades a los clientes, naciendo lo que conocemos hoy en día como servicios en la nube.

*Ilustración 1 - Logo WWW*

Los servicios en la nube (o en inglés, *Cloud Computing*), son hoy en día un pilar fundamental para el desarrollo web, pues gracias a ellos se pueden almacenar y desplegar aplicaciones y páginas web, siendo accesibles para todo el mundo. Además, la cantidad de servicios que estos pueden llegar a ofrecer pueden ir desde funcionalidades simples como el almacenaje y despliegue de los recursos hasta la posibilidad de montar una infraestructura de aplicación propia y gestionarla.

El caso que nos trae en este Trabajo de Fin de Grado está relacionado directamente con estos servicios. Hoy en día existen empresas dedicadas exclusivamente a proporcionar a terceras un mecanismo de despliegue sencillo y económico, basado en la utilización de los servicios Cloud. Con esto, las empresas desarrolladoras de contenido web pueden desprenderse de las tareas de administración de sus recursos, focalizando el tiempo en la implementación de las aplicaciones.

## 2.2 Servicios en la nube

Los servicios en la nube (Cloud Computing) son funcionalidades, exclusivas o generales de uno o varios proveedores, que se ofrecen a través de Internet como alternativa a la tradicional forma de infraestructura local. Entre sus características, destacan:

- **Flexibilidad**. Muchos proveedores mantienen una política de servicio bajo demanda, pagando únicamente por los servicios que se consuman, o bajo consumo, pagando por los recursos a los que se accedan. Otros proveedores prefieren establecer para todos o algunos de sus servicios cuotas fijas por cantidades bajas de dinero.
- **Accesibilidad**. Los recursos deben estar disponibles para todo el mundo o bien pueden estar restringidos a un cierto sector de la población según las políticas de acceso.
- **Seguridad**. Los proveedores tienen la obligación de guardar los recursos de forma segura, sin comprometer los datos de los clientes.
- **Replicación de datos**. En caso de catástrofes naturales, humanas o tecnológicas, los datos deben poder ser respaldados desde un almacenamiento secundario.
- **Actualización y sincronización**. Cada vez que se realicen cambios en los recursos, estos deben sincronizarse y actualizarse en todos los dispositivos.
- **Bajo coste por servicio**. A diferencia del coste que suponía el mantenimiento de los servidores tradicionales, los servicios en la nube apuestan por un bajo coste para el usuario, pues la infraestructura que estos utilizan es compartida entre varios, de forma que se reducen los costes de mantenimiento.

Aunque todos los proveedores de Cloud Computing tienen en común el mismo principio de funcionamiento, existen tres tipos básicos de computación en la nube.

- **SaaS**: ofrece software (aplicaciones) basado en la nube a través de Internet. Algunas de las ventajas que ofrece este tipo son el pago único por uso, acceso

gratuito a determinadas aplicaciones o el acceso a los datos de las aplicaciones desde cualquier lugar. Ejemplos de proveedores SaaS: GitHub, Gmail, Microsoft Office, Slack, Zoom…

- **PaaS**: ofrece un conjunto de recursos que facilitan el desarrollo de aplicaciones y su despliegue como herramientas de monitorización, de base de datos o de desarrollo. Algunas de las ventajas que presenta este tipo de computación en la nube son: reducir el tiempo que se dedica al desarrollo, desarrollo multiplataforma o un bajo coste de las herramientas. Ejemplos de proveedores PaaS: Heroku, Microsoft Azure App Service, AWS Lambda, Salesforce Lightning, Google App Engine…
- **IaaS**: proporciona la infraestructura necesaria para la realización de actividades de desarrollo, despliegue, almacenamiento o gestión. De este modo, se encarga de ofrecer servidores virtuales, redes, unidades de almacenamiento, entre otros, con los que un cliente puede trabaja0,r reduciendo los costes y el esfuerzo que generaría mantener toda esta estructura. Ejemplos de proveedores IaaS: Amazon Web Services, Gogle Cloud Engine, Microsoft Azure.


*Ilustración 2 - Servicios en la nube*

Actualmente, el conjunto de servicios que se ofrecen en la red es muy grande, de manera que es posible categorizarlos:

- **Almacenamiento de ficheros**. Guardar de forma segura la información y la proporcionan cuando se solicita. Ejemplos: Google Drive, Dropbox, Onedrive, iCloud, AWS S3.
- **Distribución de contenido (CDN)**. Replican la información indicada en varios servidores repartidos estratégicamente por todo el planeta. Su función principal es reducir los tiempos de acceso a los recursos. Ejemplos: AWS Cloudfront, Cloudflare, Stackpath Edge Delivery, Microsoft Azure.
- **Seguridad web (WAF)**. Protege las aplicaciones o sitios web filtrando el tráfico a partir de reglas predeterminadas o definidas por el usuario. Ejemplos:

- **Mensajería**. Permiten el intercambio de información a través de correo electrónico o aplicaciones. Ejemplos: Firebase Cloud Messaging, AWS Pinpoint, Gmail, Hotmail.
- **Máquinas virtuales**. Ofrecen los componentes físicos (hardware) en los que desplegar aplicaciones. En ellos es posible configurar, a partir de parámetros: la plataforma, la memoria asignara al proceso o el número de núcleos, entre otros. Ejemplos: Amazon EC2, Google Compute Engine, Azure Virtual Machine.

## 2.3 Alternativas

Aunque no se han encontrado referencias a trabajos relacionados, existen otras vías por las cuales se podrían haber alcanzado los objetivos propuestos en este trabajo, como por ejemplo, el cambio de proveedor.

Es cierto que existen gran cantidad de proveedores de servicios en la nube, pero no todos ofrecen la posibilidad de administrar grupos de usuarios por recursos en una cuenta. Uno de los grandes competidores en el mercado actual que ofrece esta característica es AWS, desde su servicio IAM. Para cada cuenta, es posible crear usuarios, grupos de usuarios y roles a los que se aplican políticas de acceso por recursos. Pongamos un ejemplo sobre AWS:

Un cliente inicia sesión en la cuenta principal de la empresa, donde previamente se le ha proporcionado acceso. Este cuenta con ficheros almacenados en un S3, una distribución de Cloudfront (CDN) y una base de datos DynamoDB. Su usuario debería tener definidas en IAM, por parte del administrador de la cuenta, las políticas necesarias para proporcionarle acceso a estos recursos y denegarle la lectura y escritura en el resto. De esta forma, se gestionan los recursos de los clientes de una forma sencilla y eficiente.

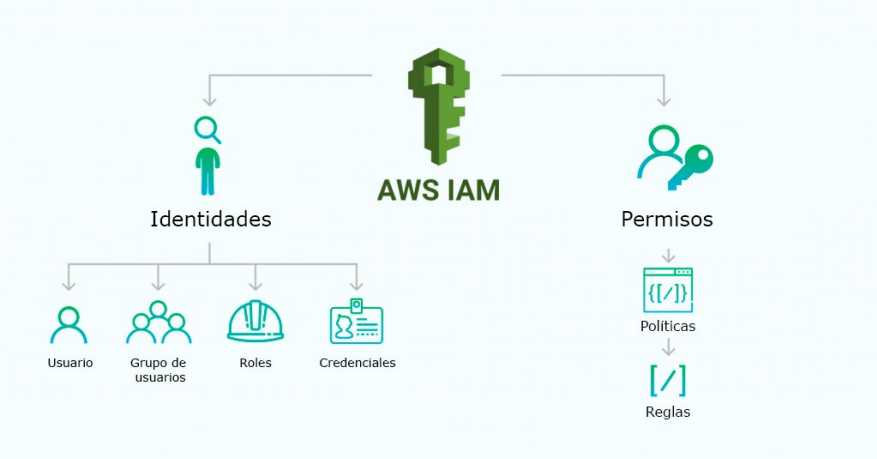


*Ilustración 3 - Objetos del servicio IAM (AWS)*

# 3. ANÁLISIS

## 3.1 Análisis de requisitos

Tomando en cuenta que el desarrollo de este trabajo parte de la base de ampliar las funcionalidades que un negocio quiere ofrecer a sus clientes mediante la integración de un nuevo portal con el proveedor de servicios que actualmente se está utilizando, podemos observar las siguientes necesidades:

1. **Necesidades de los clientes**: tras finalizar el desarrollo de una aplicación web en entornos de integración, llega la necesidad de poder ofrecer el producto a usuarios finales a través de Internet en un entorno de producción. Para ello, la mejor opción es desplegar esta aplicación web sobre un proveedor de servicios en la nube que permita exponerla de forma global a través de todo el mundo, mediante un CDN, para mejor de esta forma el tiempo de acceso a la aplicación para usuarios que se encuentren en puntos geográficos indefinidos. Como parte del proceso de mantenimiento de una aplicación desplegada en un entorno de producción es la actualización y corrección de errores detectados, existe la necesidad de tener que invalidar los ficheros que se sirven desde caché por parte de los CDN, por lo que los clientes deben poder indicar que los recursos originales han sido modificados para que la caché retire los viejos y proceda a servir los nuevos. Por último, los clientes deben poder visualizar datos estadísticos a partir de los accesos, almacenaje y configuración de su plataforma web.
2. **Necesidades del negocio**: el negocio tiene que ser capaz de poder administrar y relacionar cada uno de sus clientes con su portal o portales por separado. Además, tiene que poder ofrecerles información sobre el uso que los usuarios finales están haciendo sobre su aplicación web.
3. **Necesidades de la aplicación**: debe ofrecer una interfaz gráfica la cual permita a los clientes acceder a sus recursos desplegados en el proveedor de servicios en la nube e interaccionar con ellos. A su vez, debe implementar una estructura que permita el intercambio de información entre la aplicación y el proveedor.

A partir de estas necesidades es necesario definir un conjunto de requisitos funcionales y no funcionales que el sistema debe ofrecer.

### Requisitos funcionales

Los requisitos funcionales son un conjunto de operaciones del sistema completo o de algunos de sus componentes que, a partir del procesado de ciertos datos de entrada, generan una salida. A continuación, se describen los requisitos funcionales de la aplicación de acuerdo con el estándar IEEE 830-1998:

| ***RF.1*** | ***Operaciones con usuarios*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | - |

| | |
|---|---|
| Descripción | El sistema debe ser capaz de *recuperar*, *buscar*, *crear*, *modificar* y *eliminar* usuarios sobre la base de datos. |
| Precondición | Tener el rol de administrador y existir la base de datos con las tablas necesarias. |
| Postcondición | - |
| Importancia | Alta |
| Urgencia | Alta |

*Tabla 1 - Requisito funcional 1*

| ***RF.2*** | ***Asignar dominio a usuarios*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | ***RF.1*** |
| Descripción | El sistema debe permitir asignar a un usuario los dominios que se deseen. |
| Precondición | Tener el rol de administrador y existir la base de datos con las tablas necesarias. |
| Postcondición | - |
| Importancia | Alta |
| Urgencia | Alta |

*Tabla 2 - Requisito funcional 2*

| ***RF.3*** | ***Recuperar usuario*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | - |
| Descripción | Obtener la información sobre el usuario que se identifica ante el sistema. |
| Precondición | El usuario debe existir y aportar las credenciales válidas. |
| Postcondición | El usuario debe seleccionar el dominio que desea administrar. |
| Importancia | Media |
| Urgencia | Media |

*Tabla 3 - Requisito funcional 3*

| ***RF.4*** | ***Actualizar dominios*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | ***RF.3*** |
| Descripción | El sistema actualizará la lista de dominios de los clientes almacenados el proveedor de servicios en la nube y los guardará en la base de datos. |
| Precondición | Estar autenticado en el sistema como administrador |
| Postcondición | - |
| Importancia | Media |
| Urgencia | Alta |

*Tabla 4 - Requisito funcional 4*

| ***RF.5*** | ***Recuperar dominios*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | ***RF.3*** |
| Descripción | El sistema devolverá los dominios existentes en la base de datos para el rol de administrador y en el caso de que el rol sea de usuario, devolverá los dominios al que este tenga acceso. |
| Precondición | Estar autenticado en el sistema. |
| Postcondición | - |
| Importancia | Media |
| Urgencia | Media |

*Tabla 5 - Requisito funcional 5*

| ***RF.6*** | ***Invalidar la caché*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | ***RF.3*** |
| Descripción | El usuario podrá a través del sistema invalidar la caché del CDN del proveedor de servicio en la nube para sus recursos. |
| Precondición | Estar autenticado en el sistema. |
| Postcondición | - |
| Importancia | Media |
| Urgencia | Baja |

*Tabla 6 - Requisito funcional 6*

| ***RF.7*** | ***Obtención de métricas*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | ***RF.3*** |
| Descripción | El usuario podrá a través del sistema obtener métricas tanto del CDN como del WAF del proveedor de servicio en la nube para sus recursos. |
| Precondición | Estar autenticado en el sistema. |
| Postcondición | - |
| Importancia | Media |
| Urgencia | Baja |

*Tabla 7 - Requisito funcional 7*

## Requisitos no funcionales

Los requisitos no funcionales son aquellos que no implementan una funcionalidad específica del sistema, sino que detallan las características que debe tener su funcionamiento y pueden considerarse los siguientes:

| ***RNF.1*** | ***Escalabilidad*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | - |
| Descripción | El sistema debe estar preparado para poder ampliar las funcionalidades en un futuro de manera sencilla |
| Importancia | Alta |
| Urgencia | Media |
| Estado | Actualmente el sistema es escalable de una forma sencilla. |

*Tabla 8 – Requisito no funcional 1*

| ***RNF.2*** | ***Disponibilidad*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | - |
| Descripción | El sistema debe estar operativo la mayor parte del tiempo para los usuarios. |
| Importancia | Alta |
| Urgencia | Baja |
| Estado | Actualmente el sistema no está disponible a través de Internet (herramienta local). |

*Tabla 9 – Requisito no funcional 2*

| ***RNF.3*** | ***Mantenibilidad*** |
|---|---|
| Autor | Aarón Martín Oviedo |
| Requisitos asociados | - |
| Descripción | El sistema debe poder mantenerse sin excesiva complicación. |
| Importancia | Alta |
| Urgencia | Baja |
| Estado | Integrando el sistema con un software de gestor de procesos Node.js el cual administre la mantenibilidad. |

*Tabla 10 – Requisito no funcional 3*

## 3.2 Análisis de seguridad

Una cuestión importante que debe ser analizada en cualquier aplicación expuesta a amenazas es la seguridad. Para ello, se debe tener en cuenta qué elementos arquitectónicos se van a utilizar y qué problema presenta cada uno de ellos.

En primer lugar, la base de datos puede ser el principal foco de un atacante: información sensible de los usuarios, del negocio, del proveedor de servicios en la nube, sesiones de usuario... Los atacantes pueden llegar hasta esta información si no es configurada correctamente, por lo que hay que intentar prevenir la mayoría de los ataques

sobre esta infraestructura. Uno de los más utilizados por los atacantes ha sido la inyección SQL, el cual consiste en la introducción en campos de texto de sentencias SQL específicas con el fin de cambiar la solicitud que se realiza finalmente a la base de datos. Estas sentencias forman parte de un lenguaje de programación orientado a las bases de datos con las cuales se pueden realizar operaciones sobre estas, con el fin de manipular su configuración, tablas, vistas o usuarios.

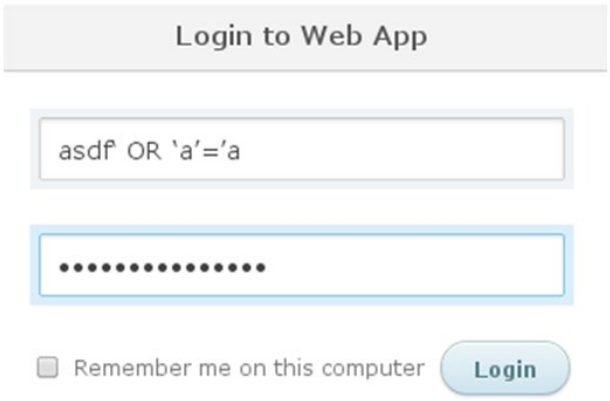


*Ilustración 4 - Ejemplo ataque por inyección SQL*

Para prevenir este tipo de ataques, las últimas versiones de los módulos que se utilizan comúnmente para realizar operaciones sobre las bases de datos ya cuentan con un sistema capaz de detectar este tipo de intrusiones y desviarlas o descartarlas, por lo que lo único que sería necesario realizar es una investigación del módulo y versión que se usa para saber si soporta este sistema. En caso contrario, se debe tener en cuenta esta amenaza e implementar una solución.

Otro problema de las bases de datos viene condicionado por el factor humano. A la hora de realizar la instalación y configuración, es importante definir los usuarios que podrán tener acceso a ellas, así como los permisos que tendrán sobre las tablas de acuerdo con las operaciones que estos puedan realizar sobre el sistema. Un error común es utilizar un usuario para todo tipo de operaciones sobre ella, y se acentúa su gravedad cuando este usuario tiene permisos sobre todo tipo de operaciones.

En segundo lugar, otro tipo de amenaza proviene del método que se utiliza en el autenticado al sistema. Es importante conocer de la existencia de métodos de autenticación que cifren los datos sensibles de identificación ante un servicio, pues sería un grave error enviar en texto plano el identificador y la contraseña, permitiendo al atacante utilizar solamente un sencillo ataque de interposición entre los extremos de la comunicación, más conocido como *Man-in-the-middle*.

La web funciona a través de un protocolo denominado HTTP (o HTTPS si se realiza una conexión segura), surgido a finales de los años 80 y principios de los 90 destinado al intercambio de recursos web. A lo largo de su desarrollo se han ido incluyendo cabeceras nuevas, es decir, metadatos que circulan encapsulados en el protocolo que ofrecen información del mensaje que se envía. Uno de estos metadatos es la cabecera *Authentication*, encargada de ofrecer un nivel extra de seguridad a la hora de iniciar sesión

en un sistema. Esta cabecera es capaz de operar con diversos mecanismos, como pueden ser *Basic*, *Bearer*, *OAuth2.0*, entre otros. A continuación, se explican brevemente algunos de estos mecanismos:

- **Basic**: consiste en la concatenación de la palabra *Basic* seguida de la codificación en Base64 de la cadena identificador:contraseña, separados por dos puntos (:).
- **Bearer**: consiste en la concatenación de la palabra *Bearer* seguida de una cadena, generalmente generada por el servidor y llamada token, que dota de acceso a los servicios a quien la posea.
- **OAuth2.0**: es la versión mejorada de OAuth1.0, la cual permitía a las aplicaciones acceder a los datos que proporcionaba APIs externas . Esta versión mejorada implica que en un primer paso se obtenga un token para autenticar próximas solicitudes y mejorar tiempos de respuesta a la vez que se mejora el nivel de seguridad.

## 3.3 Análisis del marco legal

Como hemos nombrado anteriormente, la aplicación se constituye de usuarios, los cuales se autentican en el sistema. Esto implica el uso de credenciales de acceso como un identificador y una contraseña. Actualmente, el Reglamento General de Protección de Datos (RGPD) establece que, en el caso de ser gestionadas, las contraseñas deben ser tratadas con un algoritmo de encriptación. El sistema implementado administra las contraseñas de los usuarios con un método de encriptación seguro y especificado en los estándares.

## 3.4 Solución propuesta

Cabe destacar que las necesidades del negocio quedan cubiertas en cierta parte por la selección del proveedor en la nube, en este caso, Stackpath. El negocio posee sobre este proveedor un acceso unipersonal, sobre el cual se gestionan todos los recursos de los clientes. Quedaría de esta forma cubiertas las necesidades de proveer a los clientes un alojamiento para el almacenaje y despliegue de sus aplicaciones web, así como la configuración. No obstante, el cliente no es capaz de acceder a sus recursos y, por tanto, la solución propuesta tiene como objetivo dotar de este acceso.

Esta solución consiste en crear una herramienta administrativa para la empresa, a través de un portal web, denominado Frontal o front-end, donde se mostrarán dos tipos de información:

1. **Panel de usuario**, donde acceden los clientes. En este panel se muestran las herramientas que los clientes pueden usar para modificar o visualizar sus recursos desplegados sobre Stackpath. Entre ellas están:
    a. **Invalidar la caché** mediante la introducción de rutas relativas sobre el dominio seleccionado, pudiendo ser posible realizar esta invalidación de forma completa con un mecanismo más sencillo.
    b. **Deshabilitar y habilitar la monitorización** de los recursos del WAF.

c. **Deshabilitar y habilitar** el servicio WAF sobre el dominio.
d. **Observar las métricas** registradas para el dominio seleccionado en un rango de fechas concreto. Estos datos analizan el ancho de banda de la aplicación como es el número de peticiones, la tasa de aciertos a la caché, la tasa de datos por segundo transferidos o el uso del ancho de banda.

2. **Panel de administración**, donde acceden los empleados de la empresa. En este panel se gestionan los usuarios clientes con acceso a la aplicación y sobre qué recursos. De este modo un administrador puede:
    a. **Añadir, modificar y eliminar** usuarios.
    b. **Asignar los dominios** sobre los que un usuario puede tener acceso.

En un plano paralelo, se implementa la parte de backend, el pilar con el que se sustenta el frontal y se nutre de los datos necesarios para mostrar al usuario. Este backend implementará tanto un sistema de autenticación por usuario como un sistema de autenticación por roles. Es importante entender que un rol es una función que se asigna a un usuario que se autentica ante el sistema, para determinar los recursos que acaban mostrándose ante él y las operaciones que este puede realizar. De este modo, un simple usuario, que identificaremos como al cliente de la empresa, solo podrá acceder a la configuración de su dominio, mientras que un usuario administrador podrá acceder a la configuración de la herramienta. Adicionalmente, el back-end tendrá una base de datos que almacenará datos de sesiones, información de usuarios, roles y políticas y dominios que administra la empresa. Por último, se hará una integración del sistema con Stackpath, de modo que exista permanentemente una sesión entre ambos extremos y se conceda el intercambio de información a través de las APIs.

Para comprobar el funcionamiento del conjunto, se llevará a cabo una sección de pruebas de funcionamiento, rendimiento y seguridad, la cual se describe en el capítulo 6 de esta memoria.

# 4. DISEÑO DE LA SOLUCIÓN

En este apartado se explica el proceso de diseño gráfico seguido, se introducen las tecnologías utilizadas indicando qué elementos de estas se utilizan para la implementación y se describen las arquitecturas de las diferentes capas de la herramienta: capa de presentación y de acceso a los datos.

## 4.1 Arquitectura del sistema

En lo referido a la infraestructura que se implementa en la herramienta cabe destacar que tanto front-end como back-end, se implementan sin base previa de código, aunque sí se cuenta con la ayuda de diferentes frameworks que han permitido reducir el tiempo de desarrollo.

El modelo de infraestructura elegido es Server Side Rendering (SSR), con algún matiz de la arquitectura Client Side Rendering (SSR). Veamos cada una de estas arquitecturas en detalle.

**Server Side Rendering**

Este modelo de infraestructura consiste en incrustar, desde el lado del servidor, los datos solicitados, filtrados o sin filtrar, en la plantilla HTML. Una vez formada esta plantilla, se envía como respuesta a la petición del cliente, que visualiza la página con los datos cargados. Se suelen utilizan los elementos tradicionales de desarrollo de aplicaciones web: HTML, JavaScript y CSS, entre otros.

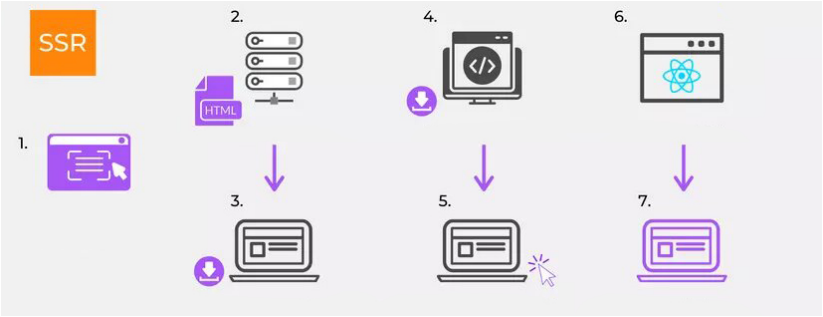


*Ilustración 5 - Estructura del Server Side Rendering*

Fijándonos en la Ilustración 2 podemos describir el proceso que sigue esta arquitectura. En primer lugar (1), un cliente solicita una página de nuestra aplicación web, en nuestro caso, asumiremos que constará de un objeto HTML y un fichero JavaScript con funciones interactivas. Esta solicitud se envía al servidor, el cual la procesa según el recurso que haya solicitado y los parámetros de entrada, con lo que construye el objeto HTML y lo envía al cliente (2). El navegador del cliente recibe el objeto y lo renderiza por pantalla (3), con lo que el usuario puede visualizar contenido estático, pero todavía no es capaz de interactuar con ciertos elementos, por lo que el navegador tiene que realizar la segunda petición del fichero JavaScript (4). El servidor devuelve este objeto que tiene almacenado y, por tanto, ahora el navegador es capaz de registrar las interacciones (5) y

de ejecutar el framework de JavaScript necesario (6) para mostrar y guardar las interacciones que el usuario realice (7).

Las ventajas que ofrece esta arquitectura son escasas. Tanto es así, que prácticamente se utiliza solo en las siguientes circunstancias:

- Necesidad de optimizar el motor de búsqueda (SEO).
- Necesidad de realidad alguna integración específica en el lado del servidor.
- Se pretende facilitar el desarrollo de la aplicación.

En cuanto a las desventajas que esta arquitectura presenta, entre otras más complejas, son:

- Interrupciones entre cambio de secciones dentro de la aplicación (pantalla en blanco que se muestra en el tiempo de carga de la página).
- Al desplegar toda la estructura sobre una única superficie, los ataques se vuelven potencialmente más críticos, ya que a un atacante le sería suficiente con acceder al servidor para acceder a todas las funcionalidades del sistema.
- Todas las peticiones del cliente desembocan en el servidor, por lo que este debe lidiar con todas ellas al tiempo que realiza otro tipo de operaciones como pueden ser tareas programadas de mantenimiento o monitorización. Esto puede conllevar que el servidor desborde y se produzcan ataques de denegación de servicios, dejando la aplicación inaccesible.
- Si el sistema es completamente personalizado, puede conllevar la necesidad de implementar cachés más complejas que puedan integrarse.

Los motivos que han llevado a escoger esta arquitectura han sido: disminuir la dificultad del desarrollo del sistema y la necesidad de realizar una integración en la parte del servidor, como un middleware por el que necesita pasar cada una de las peticiones.

**Client Side Rendering**

Esta arquitectura se basa en la carga de un único componente que funcionará como enrutador de todas las posibles vistas de la aplicación. De este modo, se obtiene la estructura, pero no los datos que conforman las vistas por lo que estos, generalmente, se obtienen mediante la resolución de peticiones XHR a una API implementada para el caso de uso. Se suelen emplear frameworks desarrollados exclusivamente para este tipo de arquitectura, como son: Angular, React o Vue.

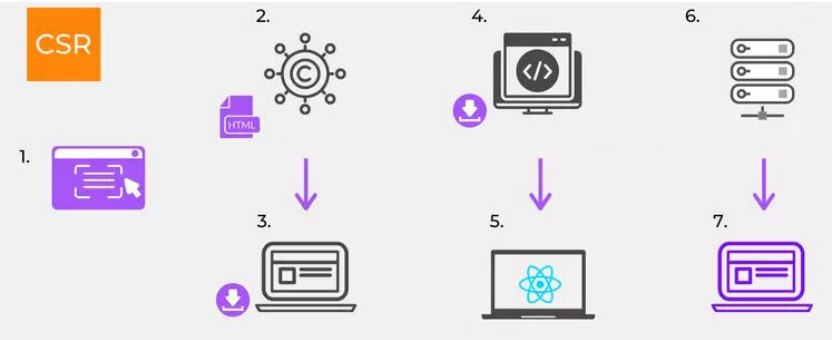


*Ilustración 6 - Estructura del Client Side Rendering*

Si se observa la Ilustración 3, se pueden apreciar notables diferencias con respecto a la arquitectura anterior. Para las dos arquitecturas, en un primer instante se necesita que haya por parte del cliente una petición (1). Es a partir de este momento en el que ambas arquitecturas distan. Mientras que en la primera la petición llega directamente al servidor (o a una caché implementada por el servidor), aquí nos encontramos una caché independiente en la infraestructura, la cual proporcionará en un menor tiempo el fichero principal de la aplicación con las referencias a los ficheros dinámicos o estáticos que se requieran (2). Los navegadores descargan estos ficheros y posteriormente las referencias que contienen en su interior (3 y 4), como ficheros JavaScript, estilos, imágenes... Es importante conocer que, en el proceso de descarga de todos estos ficheros, el cliente no podrá visualizar las vistas de la aplicación hasta obtener el último fichero referenciado en al archivo principal. A diferencia de la arquitectura anterior, la navegación por las páginas se hace de forma fluida, puesto que ya están cargados todos los componentes, no existiendo los cortes entre vistas. Una vez formada la estructura de la aplicación en el cliente, se realizan las peticiones XHR oportunas (5) a través de los ficheros JavaScript hacia el API, el cual pregunta al servidor por los datos (6) y una vez obtenidos los reenvía al cliente y se forma una página interactiva (7). Las ventajas que esta arquitectura ofrece frente a Server Side Rendering son las siguientes:

- Navegación fluida por las vistas.
- Utilización de rutas en lugar de solicitar objetos, ofreciendo una mayor comodidad, a la vez que simplificamos el esquema de vistas.
- Mantiene el servidor libre de gran parte de la carga.
- Separa la capa de presentación de la de negocio y datos en la arquitectura de programación por capas [11].

A pesar de que esta arquitectura es una de las más usadas en estos momentos, también presenta ciertas desventajas:

- Dificultad de inclusión de la aplicación en los motores de búsqueda.
- Altos tiempos de carga cuando se accede a la aplicación sin caché, o en su defecto, cuando se accede por primera vez.

El motivo por el que se ha escogido parte de esta arquitectura es separar la capa de presentación y la de negocio. De este modo, se libera al servidor de gran parte de la carga que llevaría construir cada una de las páginas con sus datos por cada una de las peticiones.

En la Ilustración 4 se muestra la estructura que sigue la herramienta desarrollada en este trabajo. Como se puede apreciar, se asemeja al tipo de arquitectura vista SSR, pero con la peculiaridad de que los datos de las vistas de la aplicación se rellenan a través de peticiones a la API, las cuales pasan previamente por un breve middleware de autenticado. Veamos con un ejemplo el flujo de la aplicación:

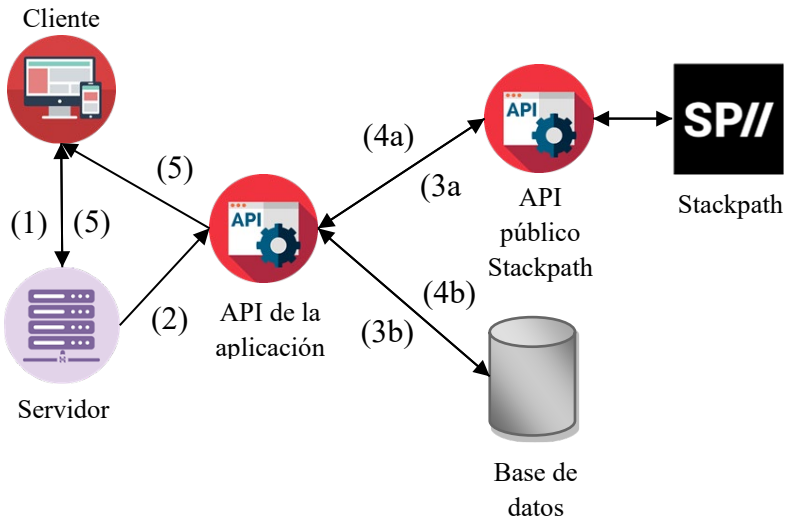


*Ilustración 7 - Estructura de la aplicación*

1. El cliente solicita, a partir de una URL, la ruta a la que desea acceder.
2. La petición pasa por un middleware de autenticación en el lado del servidor y es enviada posteriormente a la API.
3. Según el tipo de petición que se haya realizado, pueden darse los siguiente casos:
    a. Se soliciten los datos dinámicos que completen la vista. En este caso, se redirige la solicitud a la API pública de Stackpath.
    b. Se soliciten datos de aplicación, con lo que se consultan o actualizan los datos almacenados en la base de datos.
4. Como en el punto anterior, dependerá del camino que haya seguido la petición. Tendremos entonces dos respuestas diferentes encapsuladas en un objeto JSON:
    a. En el caso de la respuesta de Stackpath, los datos asociados al usuario del dominio.
    b. En el caso de la respuesta de la base de datos, la información solicitada por la aplicación.
5. Finalmente, los datos se envían desde la API de la aplicación al cliente, en diferentes formatos. Si el usuario ha solicitado una ruta de aplicación, se le devolverá el objeto HTML resultado de una breve renderización. Si la aplicación ha solicitado los datos del dominio, se devolverá un JSON con estos.

## 4.2 Diseño

En este apartado se desglosan los elementos de la arquitectura en sus esquemas más básicos, pues entorno a ellos se guía la implementación.

En un primer plano tenemos el esquema de la base de datos, pilar de la estructura que sostiene la aplicación. Esta será la encargada de almacenar los datos de la aplicación mediante el modelado de los recursos que operan en el sistema.

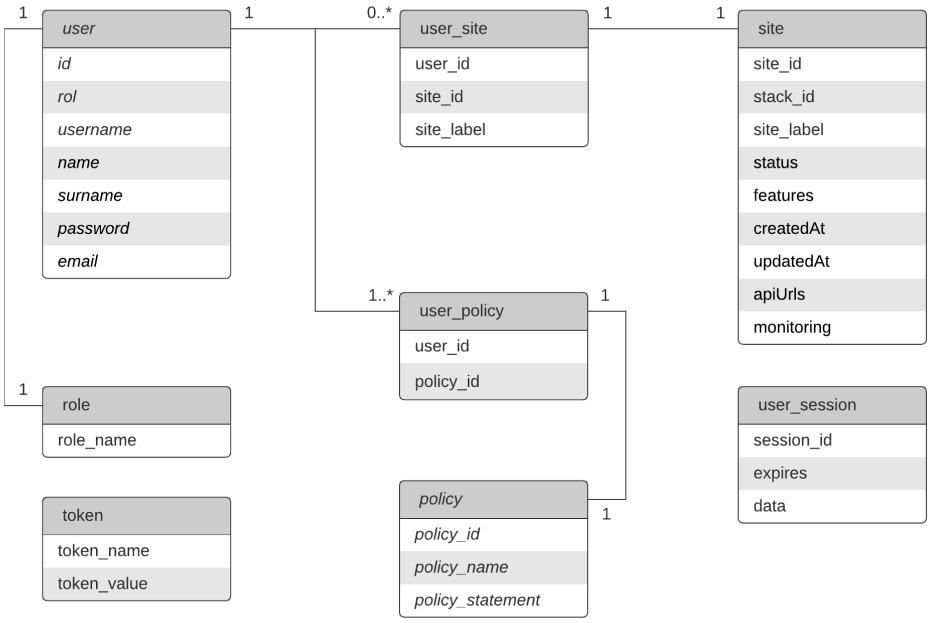


*Ilustración 8 - Diagrama UML base de datos*

Como se puede observar en la Ilustración 5, se representa mediante el estándar de esquemas UML las tablas que resultan del análisis de los modelos necesitados. Se obtienen de esta forma las 7 tablas siguientes:

- **User**: contiene la información del usuario de la aplicación. Un usuario puede tener únicamente un solo rol y dependiendo de este se le asigna una o más políticas para el uso de la API. Si es el caso de ser un cliente, puede tener asignados desde ningún dominio (*sites*) hasta el máximo de ellos.
- **Role**: contiene el nombre de los roles que se permiten en la aplicación. Actualmente se disponen solo de dos, aunque se espera que en un futuro pueda escalar la herramienta e incluir más.
- **Token** y **user_session**: mantienen información sobre las sesiones de la aplicación y los usuarios.
- **Policy**: almacena las diferentes políticas que permiten gestionar los permisos que tienen los usuarios sobre las llamadas a la API.
- **Site**: contiene los dominios que se administran desde el panel de control de Stackpath y su información.
- **User_site**: relaciona un usuario con los dominios a los que tiene acceso.

Algunas de las tablas que conforman la base de datos necesitan relacionarse con otras. Para ello, se utiliza en las bases de datos relacionales el término clave ajena. Una clave ajena, o clave foránea, es una restricción que debe existir entre dos tablas, donde se enlazan cada una de las columnas que se requieran de una tabla con las columnas de otra tabla distinta. En nuestro diseño, podemos encontrar estás claves en la Ilustración 6 representadas mediante líneas discontinuas.

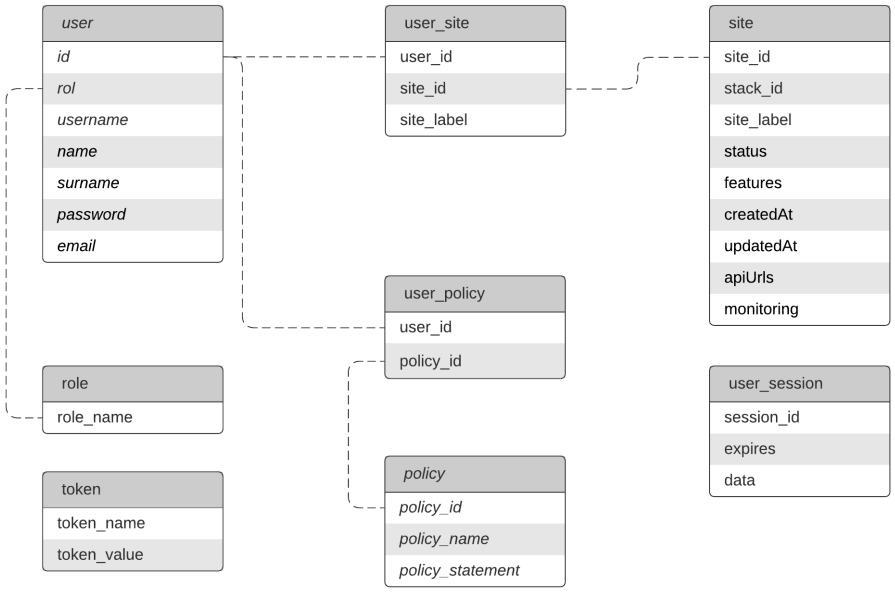


*Ilustración 9 - Claves ajenas de la base de datos*

Es común realizar los desarrollos en máquinas locales para realizar pruebas de concepto. Una vez completadas las pruebas se necesita llevar el trabajo realizado a entornos de producción, por lo que los gestores de las bases de datos implementan en su interfaz funciones para exportar estos esquemas e importarlos de una manera simple. El lenguaje utilizado por la base de datos de nuestra aplicación es SQL y, por ende, se deben crear mediante la siguiente secuencias de ordenes:

```
1.  CREATE TABLE `policy` (
2.    `policy_id` int NOT NULL AUTO_INCREMENT,
3.    `policy_name` varchar(45) NOT NULL,
4.    `policy_statement` json DEFAULT NULL,
5.    PRIMARY KEY (`policy_id`),
6.    UNIQUE KEY `name_UNIQUE` (`policy_name`),
7.    UNIQUE KEY `id_UNIQUE` (`policy_id`)
8.  ) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=utf8mb3;
9.
10. CREATE TABLE `role` (
11.   `role` varchar(15) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
12.   PRIMARY KEY (`role`),
13.   UNIQUE KEY `role_UNIQUE` (`role`)
14. ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_spanish2_ci;
15. CREATE TABLE `site` (
16.   `site_id` varchar(50) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
17.   `stack_id` varchar(50) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
18.   `site_label` varchar(40) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
19.   `status` varchar(15) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
20.   `features` json DEFAULT NULL,
21.   `createdAt` bigint NOT NULL,
22.   `updatedAt` bigint NOT NULL,
23.   `apiUrls` json DEFAULT NULL,
24.   `monitoring` tinyint NOT NULL DEFAULT '1',
25.   PRIMARY KEY (`site_id`),
26.   UNIQUE KEY `domain_UNIQUE` (`site_id`),
```

```
  UNIQUE KEY `label_UNIQUE` (`site_label`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_spanish2_ci;

CREATE TABLE `token` (
  `token_name` varchar(20) NOT NULL,
  `token_value` json DEFAULT NULL,
  PRIMARY KEY (`token_name`),
  UNIQUE KEY `token_name_UNIQUE` (`token_name`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

CREATE TABLE `user` (
  `id` int NOT NULL AUTO_INCREMENT,
  `role` varchar(15) COLLATE utf8_spanish2_ci NOT NULL,
  `username` varchar(45) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
  `name` varchar(20) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
  `surname` varchar(45) CHARACTER SET utf8 COLLATE utf8_spanish2_ci DEFAULT
NULL,
  `password` varchar(120) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
  `email` varchar(50) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `username_UNIQUE` (`username`),
  KEY `role_fk` (`role`),
  CONSTRAINT `role_fk` FOREIGN KEY (`role`) REFERENCES `role` (`role`)
) ENGINE=InnoDB AUTO_INCREMENT=163 DEFAULT CHARSET=utf8mb3
COLLATE=utf8_spanish2_ci COMMENT='Users table';

CREATE TABLE `user_policy` (
  `user_id` int NOT NULL,
  `policy_id` int NOT NULL,
  PRIMARY KEY (`user_id`,`policy_id`),
  KEY `policy_id_fk` (`policy_id`),
  KEY `user_id_fk` (`user_id`),
  CONSTRAINT `policy_id_fk` FOREIGN KEY (`policy_id`) REFERENCES `policy`
(`policy_id`) ON DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `user_id_fk` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

CREATE TABLE `user_session` (
  `session_id` varchar(128) CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL,
  `expires` int unsigned NOT NULL,
  `data` mediumtext CHARACTER SET utf8mb4 COLLATE utf8mb4_bin,
  PRIMARY KEY (`session_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3;

CREATE TABLE `user_site` (
  `user_id` int NOT NULL,
  `site_id` varchar(50) CHARACTER SET utf8 COLLATE utf8_spanish2_ci NOT NULL,
  `site_label` varchar(70) COLLATE utf8_spanish2_ci NOT NULL,
  PRIMARY KEY (`user_id`,`site_id`),
  UNIQUE KEY `user_site_UNIQUE` (`user_id`,`site_id`),
  KEY `site_fk` (`site_id`),
  CONSTRAINT `site_fk` FOREIGN KEY (`site_id`) REFERENCES `site` (`site_id`) ON
DELETE CASCADE ON UPDATE CASCADE,
  CONSTRAINT `user_fk` FOREIGN KEY (`user_id`) REFERENCES `user` (`id`) ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb3 COLLATE=utf8_spanish2_ci
COMMENT='Policies table';
```

En el siguiente nivel nos encontramos con la API de la aplicación. Su diseño está basado en la arquitectura API REST [13], la cual no está regida por un estándar en concreto, sino que recurre al cumplimiento de determinados criterios definidos a continuación:

- **Arquitectura cliente-servidor**. Esto implica desacoplar el servidor del cliente y facilitar el intercambio de datos.
- **Ausencia de estado**. El servidor no guarda el estado de cada una de las peticiones de los clientes, sino que son tratadas individualmente y cada una de ellas debe contener los datos necesarios para su correcto procesamiento.
- **Interfaz uniforme**. Se debe tratar el intercambio de mensajes a través de una misma estructura de datos. Para este caso, se utiliza el formato JSON, aunque otras opciones son XML o HTML. También debe proporcionar la información necesaria de forma clara y concisa.
- **Sistema por capas**. El cliente no debe conocer la estructura de la aplicación y como se desenvuelve la resolución de la petición. De esta manera, el cliente solo conocerá el servicio con el que se comunica directamente.

En la Tabla 11 se muestran los recursos del API REST a los que un usuario cliente tiene acceso junto a una breve descripción de cada uno.

| **Recurso** | **Método** | **Descripción** |
|---|---|---|
| /api/v1/sites | GET | Devuelve la información del dominio en el que se ha iniciado sesión. |
| /api/v1/sites/metrics | GET | Devuelve un conjunto de estadísticas del sitio a través de unos parámetros de entrada. |
| /api/v1/cdn/purge | POST | Invalida la caché del dominio seleccionado a partir de un objeto que contiene las rutas a invalidar y si debe ser recursivo. |
| /api/v1/cdn/purgeAll | DELETE | Invalida todas las rutas del dominio seleccionado. |
| /api/v1/cdn/purge/{purgeId} | GET | Obtiene el estado de la invalidación con ID {purgeId}. |
| /api/v1/cdn/metrics | GET | Obtiene estadísticas del servicio CDN asociado al dominio a través de unos parámetros de entrada. |
| /api/v1/waf | POST | Activa el servicio WAF. |
| /api/v1/waf | DELETE | Desactiva el servicio WAF. |
| /api/v1/waf/monitoring | POST | Activa la monitorización. |
| /api/v1/waf/monitoring | DELETE | Desactiva la monitorización. |
| /api/v1/waf/requests | GET | Obtiene información sobre las peticiones que han pasado por los registros del WAF. |
| /api/v1/waf/requests_table | GET | Obtiene el conjunto de peticiones que han pasado por el WAF, optimizado para el plugin DataTables de JS en modo servidor. |

| | | |
|---|---|---|
| /api/v1/waf/request_stats | GET | Obtiene el total de peticiones bloqueadas para el dominio. |

*Tabla 11 - API de la aplicación*

Al contrario que la API, las rutas que se utilizan para acceder a cada una de las vistas de la aplicación no devuelven un objeto JSON con datos, sino que sirven para que el servidor identifique, a través de un router lógico previamente configurado en Express, qué vista desea cargar el usuario, para renderizarla y enviársela como respuesta a la petición. Se definen las vistas en la siguiente tabla (Tabla 12):

| **Recurso** | **Método** | **Descripción** |
|---|---|---|
| / | GET | Renderiza el objeto principal de la aplicación, una simple página de información sobre la herramienta y la envía al cliente. |
| /login | GET | Renderiza la página de autenticación a la herramienta y la envía al cliente. |
| | POST | Verifica el objeto del cuerpo de la petición, que debe contener los datos de inicio de sesión del usuario, ante el sistema y devuelve el resultado de la petición. |
| /logout | GET | Cierra la sesión establecida previamente por el usuario frente al sistema y redirige a la página de autenticación. |
| /sites | GET | Renderiza la página de selección de dominio entre las listas de dominios asignados al usuario. |
| | POST | Obtiene el dominio seleccionado por el usuario, lo verifica ante el sistema y devuelve el resultado de la comprobación. |
| /switch-site | GET | Retira el dominio asignado a la sesión de usuario para redirigir a la vista de selección de dominio. |
| /dashboard | GET | Renderiza la página principal del panel de usuario, con información breve de la herramienta y sus características. |
| /dashboard/cache | GET | Renderiza la página donde el usuario podrá invalidar la caché a nivel completo o selectivo. |
| /dashboard/metrics/cdn | GET | Renderiza la página de estadísticas CDN del dominio. |
| /dashboard/metrics/waf | GET | Renderiza la página de estadísticas WAF del dominio. Además, se pueden activar y desactivar las funciones de monitorización y WAF. |
| /administrador* | GET | Renderiza la página principal del administrador, donde se muestran las herramientas para este rol. |
| /administrador/usuarios* | GET | Renderiza la vista con los usuarios dentro del sistema. |

| | | |
|---|---|---|
| /administrador/usuarios/crear* | GET | Renderiza la vista donde se puede crear un usuario. |
| /administrador/usuarios /{usuario}* | GET | Renderiza la vista con información del usuario {usuario}, donde se pueden realizar ajustes sobre este. |
| *Por seguridad, se oculta el verdadero nombre de la ruta con el que el usuario administrador accede. | | |

*Tabla 12 - Router de las vistas de aplicación*

Por último, tenemos la capa de presentación, aquella donde el usuario interactúa con el sistema. Esta sección no entra en el desarrollo de la herramienta, sino que en su lugar se ha optado por seleccionar plantillas web y rellenarlas con los datos obtenidos por la capa de negocio, realizando pequeños ajustes de estilos. Se muestra a continuación las vistas obtenidas a partir de las plantillas:

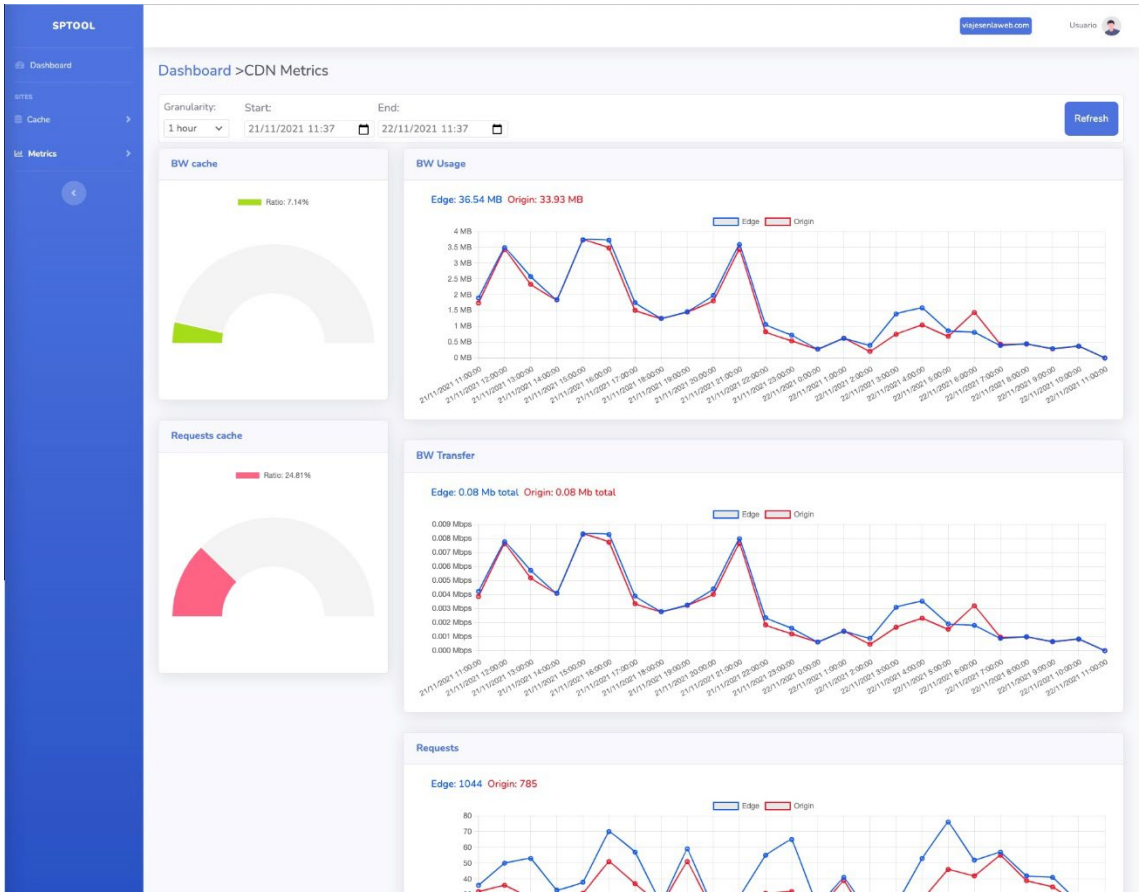


*Ilustración 10 - Vista CDN*

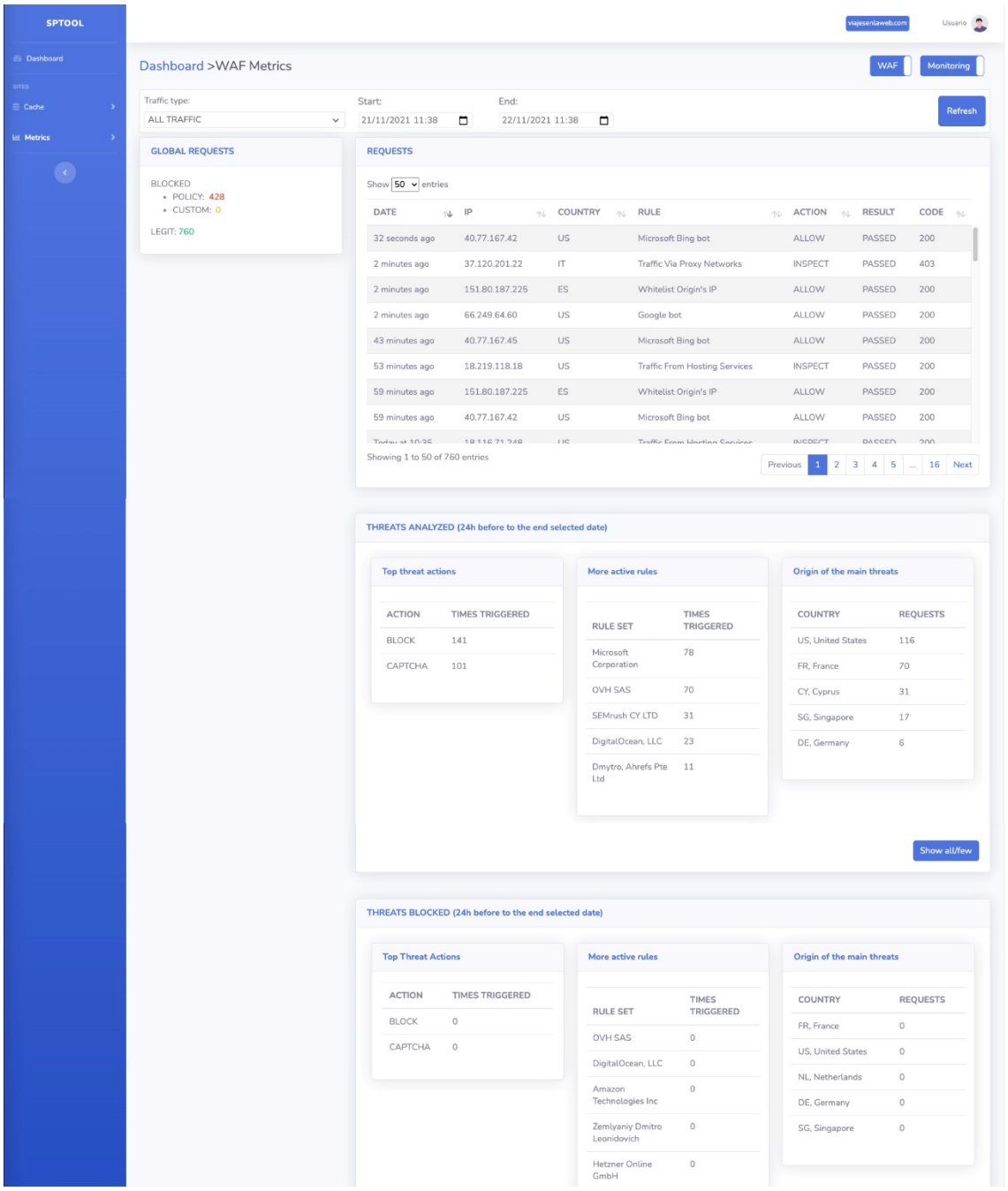


*Ilustración 11 - Vista WAF*

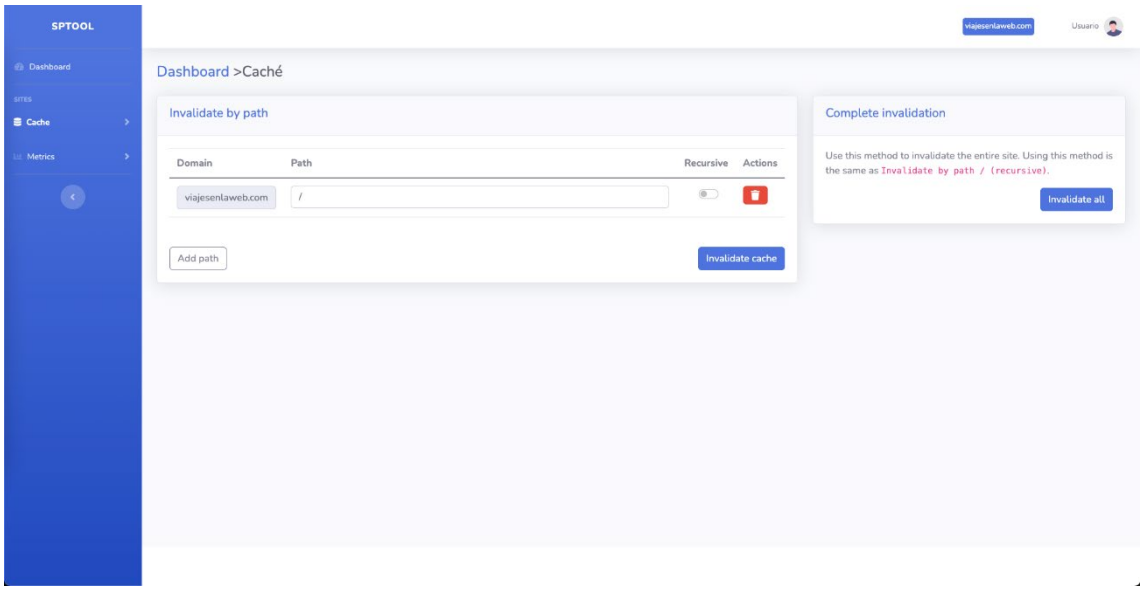


*Ilustración 12 - Vista invalidación caché*

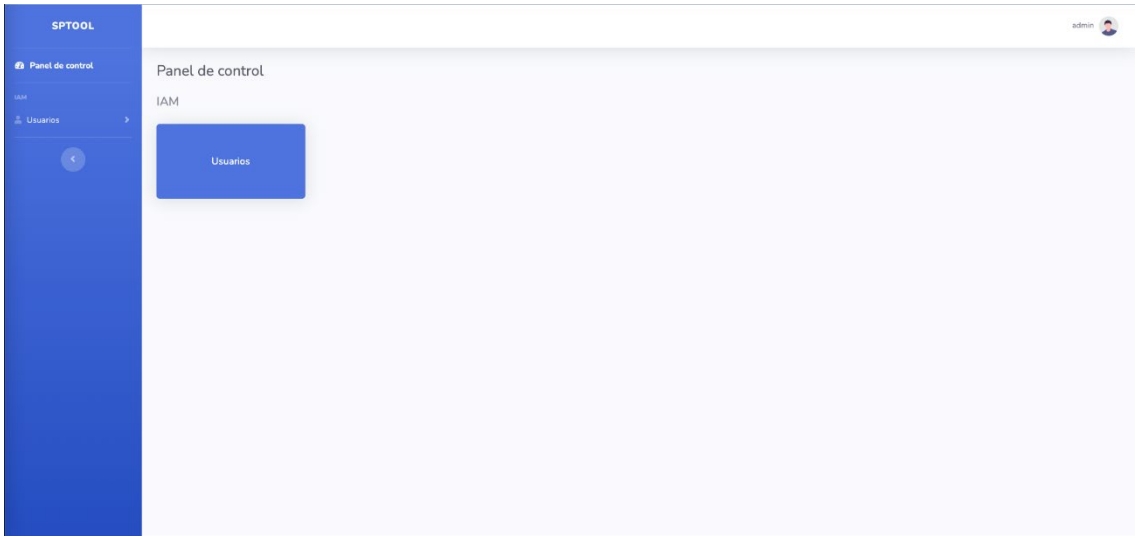


*Ilustración 13 - Vista herramientas administrador*

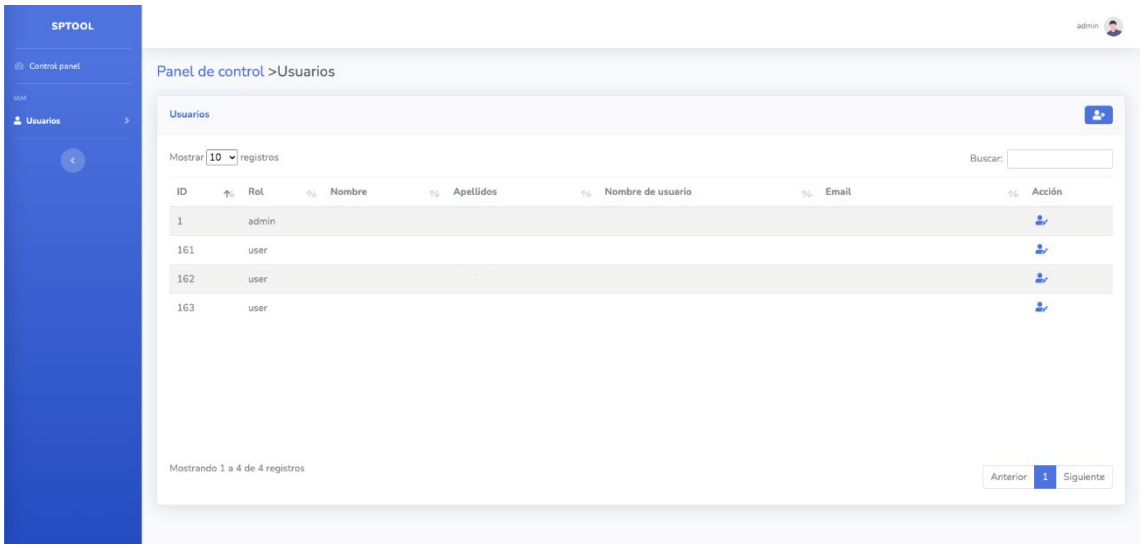


*Ilustración 14 - Vista usuarios de la aplicación administrador*

## 4.3 Tecnologías utilizadas

### 4.3.1. Lenguajes

**Node.js**

Entre los diferentes *runtimes* que se pueden elegir para envolver las aplicaciones web en el lado del servidor se encuentra Node.js (o simplemente Node) [14], basado en el lenguaje de programación JavaScript (JS).

*Ilustración 15 - Logo Node.js*

Las características que llevan a este entorno de ejecución a ser unos de los más usados en la industria de la programación web son:

- **Facilita el desarrollo**. Este entorno es muy simple de utilizar si se tienen conocimientos previos de JS, por lo que no sería necesario llevar a cabo un aprendizaje exhaustivo.
- **Escalable**. Aunque el motor principal del entorno sea un único hilo, este entorno está diseñado para permitir que las aplicaciones puedan crecer considerablemente. Las entradas y salidas asíncronas pueden ser ejecutadas concurrentemente, manteniendo un alto rendimiento.
- **Asincronía**. Un suceso de un evento puede no tener correspondencia temporal con otro suceso del mismo evento.
- **Bajos tiempos de respuesta**. La asincronía y la ejecución mono hilo del código hacen que la velocidad de ejecución sea mayor, puesto que no se generan bloqueos.
- **Amplia variedad de módulos**. Existe una gran cantidad de librerías de código prediseñadas por empresas o usuarios que permiten reducir el tiempo de desarrollo para nuestra aplicación. Por ejemplo, podemos acceder a ellas a través del gestor de paquetes NPM.
- **Funcionamiento multiplataforma**. Al formar parte de un lenguaje inclusivo, esta tecnología funciona independientemente de la plataforma que tengamos instalada en el servidor. Además, podremos intercambiar información entre servidores bajo plataformas diferentes siempre y cuando este intercambio se produzca a nivel de entorno.

Todas estas propiedades hacen que Node.js sea perfecto para empezar a desarrollarte en el universo de la programación web, ya que facilita muchas tareas que, en otros entornos, se hubieran tenido que resolver. Este es el motivo por el que se ha seleccionado Node.js como entorno de ejecución.

Se muestra a continuación un ejemplo de una simple implementación de un servidor web HTTP basado en el entorno de ejecución Node.js:

```
1.  const http = require('http');
2.
3.  const hostname = '127.0.0.1';
4.  const port = 1337;
5.
6.  http.createServer((req, res) => {
7.    res.writeHead(200, { 'Content-Type': 'text/plain' });
8.    res.end('Hello World\n');
9.  }).listen(port, hostname, () => {
10.   console.log(`Server running at http://${hostname}:${port}/`);
11. });
```

Con el fin de facilitar la creación de este servidor, se utiliza el módulo prediseñado llamado Express, explicado en el siguiente apartado.

### Express

Partiendo de la necesidad de facilitar las labores de implementación del sistema, se ha seleccionado Express como infraestructura para el desarrollo del servidor.


*Ilustración 16 - Logo Express*

Express es un módulo, accesible desde gestores de paquetes como el comentado anteriormente NPM, el cual permite inicializar un servidor a partir de un puerto y nombre de dominio.

A partir de este momento, se encuentra disponible un servidor escuchando peticiones de clientes por el puerto asignado, pero sin ninguna funcionalidad. Es por ello por lo que dentro de este módulo se define una clase denominada *Router*, a partir de la cual podemos asignar rutas a nuestra aplicación mediante el uso de los diferentes métodos que ofrece el protocolo HTTP. Además, una característica interesante que ofrece es la posibilidad de crear y asignar middlewares sobre la aplicación en su conjunto o sobre rutas específicas.

En la Ilustración 14 se muestra un ejemplo de servidor Express. Primero se carga el módulo Express con todos sus componentes. Seguidamente se inicializa el componente principal de la aplicación al cual se le asignan, entre otros parámetros: los directorios donde se encuentran las rutas y las vistas, el motor de renderizado de plantillas, o el uso de módulos adicionales como es el caso del middleware JSON que incluye el módulo Express por defecto.

```
const express = require('express');

try {
    const app = express();
    app.use(express.json());
    app.use(require('./routes/main.routes'));
    app.set('views', './src/www/views');
    app.set('view engine', 'ejs')
    app.listen(80);
} catch (err) {
    process.exit(-1);
}
```

*Ilustración 17 - Ejemplo de servidor Express*

Para gestionar las rutas de la aplicación se asigna sobre el Router un recurso con su método HTTP que se invoca cuando el servidor recibe una petición del cliente con el método establecido. Este recurso es una función la cual recibe como parámetros de entrada el path que desembocará su llamada y una sucesión de funciones que se invocan consecutivamente. Para cada una de estas funciones consecutivas se dispone de la petición original del cliente y de la respuesta que se debe enviar al cliente.

### MySQL

MySQL [15] es un gestor de base de datos relacionales que opera bajo el lenguaje SQL. Las bases de datos relacionales funcionan bajo el concepto de relaciones entre los datos que estas almacenan, a través del enlace de atributos de diferentes tablas tras el concepto de clave ajena. Las principales características que definen el gestor MySQL son:

- **Disponibilidad en múltiples plataformas**. este gestor se puede encontrar para numerosas plataformas como pueden ser: Windows, Linux, Mac OS, Solaris…
- **Transacciones**. conjunto de operaciones (sentencias SQL) que deben ser ejecutadas consecutivamente sin alterar el estado del sistema hasta haber finalizado la última de ellas.
- **Conexión segura**.
- **Búsqueda e indexación de campos de texto**.
- **Replicación de los datos**. Consiste en copiar y actualizar los datos almacenados en varias bases de datos con el fin de no perder la información ante cualquier incidente.
- **Opción de escoger el motor de almacenamiento**. Permite trabajar con diferentes motores de almacenamiento que se añaden en tiempo de ejecución. Algunos ejemplos son: InnoDB, Falcon, Merge, MyISAM.

Entre las características que han hecho escoger este gestor ante otros han sido la mayor variedad de librerías que ofrece NPM y su fácil integración con las clases con las que trabaja la aplicación.

## HTML y EJS

Desde los inicios de Internet, HTML [16] ha sido uno de los principales lenguajes de programación web. Este lenguaje basa la construcción de páginas web en un conjunto de etiquetas que acaban componiendo una estructura en forma de árbol, llamado DOM, como se puede observar en la Ilustración 17. Las etiquetas son un conjunto de propiedades que definen el objeto que se va a construir. La mayoría de los objetos que se construyen deben tener una etiqueta de apertura y otra de cierre para indicar su extensión. Su estructura se define como:

- Carácter "menor que" (<). Las etiquetas de cierre llevarán, adicionalmente, el carácter "barra oblicua" (/).
- Elemento (p.ej: *div*, *p*, *label*, *link*, …)
- Atributos (p.ej: *style*, *value*, *href*, …)
- Carácter "mayor que" (>)

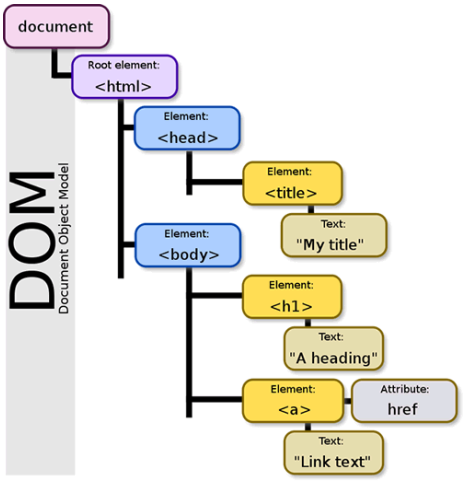


*Ilustración 18 - Árbol DOM (HTML)*

Actualmente existen 4 versiones de este lenguaje, empezando en la versión 2.0 del estándar (deprecada) y llegando hasta la versión 5.2 del mismo. Con cada nueva versión se han ido incorporando nuevos cambios que han permitido a los desarrolladores incluir nuevas funcionalidades a las páginas, como nuevas etiquetas o también la posibilidad de representar los objetos en los navegadores más recientes.

La estructura que sigue este estándar comienza por definir con la etiqueta reservada <html>. En su interior podemos encontrar la etiqueta <head>, contenedora de metadatos en forma de etiquetas (p. ej: el título de la página). También encontramos las etiquetas <body> y <footer>. Ambas contienen un conjunto de etiquetas que definen el contenido que el usuario final observa, con la diferencia que la primera tiene prioridad en el DOM, por lo que suele preceder a la segunda.

HTML ofrece la posibilidad de crear objetos de texto estáticos y ser enviados como respuesta a una petición, pero no ser modificados en tiempo real. En ocasiones se requiere

poder incrustar en él ciertos datos por parte del servidor y como solución a ello surgen los motores de plantillas. Su función básica consiste en separar el código comercial del lógico, mejorando la eficiencia del desarrollo y pudiendo reutilizar código. Entre los diversos motores que existen en la actualidad se ha seleccionado para esta aplicación EJS.

Cabe destacar que no existe mayor diferencia entre los distintos motores más que la sintaxis empleada para la inclusión de código, por lo que cualquiera de ellos podría servirnos para llevar a cabo nuestra implementación.

**CSS y Bootstrap**

Para que el usuario tenga una buena experiencia dentro de nuestra aplicación es importante dotar a nuestras vistas de unos estilos visuales adecuados. En esto consiste el trabajo de CSS, un lenguaje creado para modelar los elementos creados en el HTML y ser interpretado por los navegadores.

Para poder aplicar estilos con CSS es importante marcar las etiquetas en el HTML con una clase o un identificador, el cual se usa para indicar qué elementos van a ser modificados visualmente mediante propiedades. Estos estilos se pueden aplicar sobre el propio HTML o enlazándolo a un fichero CSS. Ejemplos de propiedades que pueden ser modificadas son: el estilo y fuente de la letra, posicionamiento de elementos, color de fondo, estilo de bordes, etc.

El proceso de modelado puede suponer tener que crear reglas para cada uno de los elementos, por lo que esta aplicación se ayuda del framework Bootstrap, encargado de gestionar las clases y sus estilos por nosotros mediante librerías de estilos previamente creadas. De esta forma, no hay que preocuparse de definir reglas CSS, sino simplemente de aplicar la clase que mejor se adecue al formato que deba tener cada elemento.

**JavaScript y JQuery**

El usuario tiene que poder interactuar con los elementos de la página, respondiendo a eventos o creándolos. Esta funcionalidad no la ofrece por defecto HTML y CSS, por lo que a estos dos hay que añadirle un tercero para completar la programación web en el cliente. Este es el caso de JavaScript.

JavaScript es un lenguaje dedicado a la programación orientada a objetos, basado en prototipos e interpretado o compilado en tiempo de ejecución. Su funcionamiento se basa en la ejecución mono hilo y asíncrona de eventos, de modo que estos no tienen correspondencia temporal con otros sucesos.

Desde el lado del cliente, es el navegador quien se encarga de interpretar el código JavaScript enviado por el servidor y mostrar los resultados de sus eventos, aunque no es el único ámbito donde este opera. Podemos encontrarnos con él dentro de entornos de ejecución (como el mencionado anteriormente Node.js), en aplicaciones (como Adobe Acrobat) o en scripts (secuencias de comandos ejecutados por un intérprete).

Al ser un lenguaje en el que el coste de aprendizaje es relativamente bajo y ser referencia de estudio en el grado, es perfecto para ser empleado en el desarrollo de código web dinámico de nuestra aplicación. Concretamente se hace uso de él para el intercambio de mensajes entre el cliente - API y para la interacción con elementos de la vista. En el segundo caso de uso, se utiliza a través del framework JQuery, una biblioteca multiplataforma de JavaScript encargada de proporcionar funciones comunes para reducir tiempo y coste de desarrollo.

# 5. DESARROLLO DE LA SOLUCIÓN

Este capítulo tiene como finalidad explicar el proceso llevado a cabo en la implementación de la aplicación, tanto el frontend como el backend mediante la expliación y ejemplificación del código utilizado.

## 5.1 Backend

En el backend se implementan todos los componentes que hacen posible las interacciones en el sistema. Es la capa que está por detrás de lo que el usuario percibe visualmente y es encargada de proporcionar la mayor parte de las funcionalidades descritas en el análisis de requisitos. Veamos los procedimientos llevados a cabo para su desarrollo.

En primer lugar se construye la base de datos, pilar fundamental que almacena tanto los datos de la aplicación como los datos de usuario. Para ello, se instala y configura en la máquina el software MySQL Server y posteriormente se inicializa. Con el servidor en funcionamiento, se crea la base de datos y sus tablas a través del software gráfico de base de datos MySQL Workbench.

Con la base de datos y las tablas creadas procedemos a definir los atributos y sus tipos tal y como se ha especificado en la fase de diseño. Se definen también las claves ajenas como propiedades de tablas y de esta manera, quedan relacionadas entre sí. Como último paso de configuración, se crean los usuarios que tendrán acceso a través de la aplicación.

Seguidamente, se prepara el entorno para comenzar a trabajar con el servidor y la API. Todo el código se almacena en un repositorio privado de GitHub, por lo que en un primer lugar se crea el repositorio mediante la herramienta Git con el comando “git init”. A partir de unas preguntas que nos muestra este comando, se creará un directorio local con la configuración de Git, donde almacenaremos el código y posteriormente iremos subiendo a medida que se desarrolle al repositorio online. Con el repositorio listo para subir código, comienza el desarrollo del servidor.

El servidor web se crea a partir de la inicialización del componente Express, al que llamaremos aplicación. En él se definen los diferentes middlewares que se van a manejar en la aplicación, así como las rutas y los ficheros estáticos (ver Ilustración 17). Para esta aplicación se utilizan middlewares predefinidos (gestión de JSON en los objetos de peticiones y respuestas) como middlewares nuevos (gestión de roles y autenticación de usuarios). Los segundos son creados con el fin de ofrecer una herramienta que mantenga sesiones de usuarios con diferentes permisos dentro de esta, ya sea solo acceder a cierta información o administrar la herramienta.

La estructura sobre la que se elabora el backend cuenta con: la API, la librería, logs con información del sistema, el middleware utilizado por la aplicación, los modelos que

se utilizan para la base de datos, las rutas de la aplicación y utilidades. Esta estructura de ficheros se puede observar en la Ilustración 19.

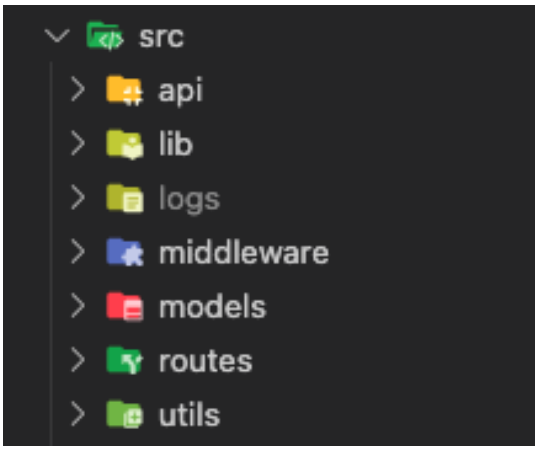


*Ilustración 19 - Estructura de archivos de la aplicación*

La API se desarrolla a partir del componente Router de Express. Este componente permite definir rutas a las cuales se pueden asignar tantos métodos HTTP diferentes se quiera. Cada uno de los métodos definidos requiere mínimo de una función que se ejecutará cuando la ruta sea activada, teniendo como argumentos el objeto de petición y el objeto de respuesta los cuales pueden ser alterados dependiendo de los criterios de funcionalidad de la ruta. De esta forma se construye la estructura de llamadas API definida en el apartado de diseño.

En el apartado de librería (*lib*) se desarrollan dos componentes: un conector para intercambiar datos con la base de datos y la integración con el servicio de Stackpath. El primero de ellos recibe sentencias SQL que envía a la base de datos y devuelve la respuesta de forma estructurada; el segundo ejecuta peticiones HTTP hacia la API de Stackpath con los parámetros que se le pasan de entrada, devolviendo la salida generada por el proveedor.

En cuanto a los middlewares, tanto el de sesión como el de roles, son encargados de gestionar el control a la aplicación, denegando aquellas peticiones no autorizadas y redirigiendo las autenticadas.

Para una interacción eficiente entre la base de datos, Stackpath y la aplicación, se crean los modelos Site, contenedor de atributos de los dominios y User, referente a un usuario cliente de la aplicación.

Al igual que la API, las rutas de la aplicación se generan a partir del componente Router de Express, pero en este caso, su función es proporcionar al frontal la información necesaria sobre qué vista renderizar al usuario.

Finalmente, se cuenta con utilidades utilidades del sistema. En este caso tenemos un sistema de logs que describen el funcionamiento del sistema, tanto errores como información relevante y la deja accesible en ficheros de texto.

## 5.2 Frontend

En el frontend nos encontramos con las vistas y elementos con los que el usuario final interactua a través del navegador, así como los estilos que dotan a la interfaz de un aspecto más elegante.

En primer lugar se define la estructura que tendrá el frontal, mostrado en la Ilustración 20 y encapsulado como *www*. Podemos distinguir dos secciones: los ficheros públicos y las vistas que serán renderizadas.

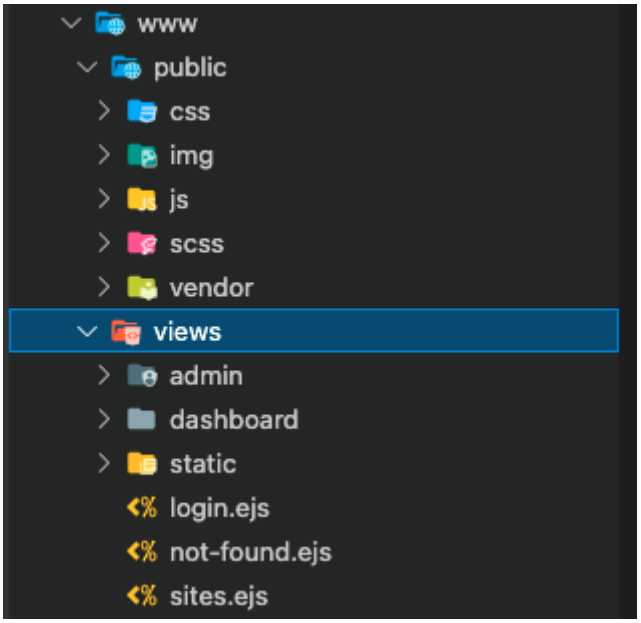


*Ilustración 20 - Estructura de ficheros del frontal*

Comenzando por los ficheros públicos encontramos el apartado de hojas de estilos (*css*), donde a partir del lenguaje CSS3, se definen los estilos visuales que tendrán los componentes de la aplicación como por ejemplo, el color de fondo, el tamaño de la letra, el alto y ancho de los elementos, etc. A continuación existe una carpeta contenedora de imágenes, por lo que se almacenarán aquí todas aquellas que se usen en la aplicación.

Seguidamente tenemos el apartado más extenso del frontal y que ofrece las interacciones con el sistema. Esta es la carpeta *js*, donde se encuentran los ficheros JavaScript que otorgan la funcionalidad del frontal. En ellos se encuentran las llamadas a la API de la aplicación, recogiendo y mostrando los datos obtenidos. Además, ofrece funcionalidades como la carga de componentes de terceros como las tablas del plugin DataTables o interacciones con elementos de la vista (botones, toogles, datapickers...). Las carpetas *scss* y *vendor* pertenecen a plugins de terceros, por lo que obviaremos su contenido.

La segunda sección del frontal tiene que ver con las vistas de la aplicación. Se cuenta con las dos subsecciones como objetivos de la aplicación: la parte del usuario (*dashboard*) y la parte del administrador (*admin*). Destacar previamente que ambas cuentan con su vista de autenticación (*login*), teniendo el usuario una vista adicional para seleccionar el

dominio al que acceder. Las vistas están creadas a raíz de una plantilla base, modificando los elementos que se deben mostrar en cada una de ellas, quedando por lo tanto:

1. **Dashboard**:
    a. **index**: muestra una breve información de la aplicación.
    b. **cache**: componentes para poder invalidar la caché de Stackpath a partir de inputs de texto o una invalidación completa.
    c. **cdn**: muestra estadísticas del CDN de Stackpath para un dominio.
    d. **waf**: muestra información sobre las reglas del servicio WAF de Stackpath y permite, mediante toogles, habilitar y deshabilitar tanto el servicio WAF como la monitorización.
2. **Admin:**
    a. **usuarios**: a través de una tabla se muestran los usuarios que tiene la aplicación, tanto administradores como usuarios nominales.
    b. **detalles de usuario**: muestra los detalles de un usuario y permite modificar sus propiedades. También permite asignar los dominios correspondientes.
    c. **crear un usuario**: presenta un formulario que se debe rellenar con valores de texto para crear un nuevo usuario en el sistema.

Finalmente, existe una vista HTML que se muestra cuando una ruta solicitada no es encontrada en la aplicación (*not-found*).

# 6. IMPLANTACIÓN Y PRUEBAS

Una vez analizada, diseñada e implementada la aplicación, se ha de desplegar en un entorno productivo. En este caso, se trata de una herramienta local, lo que significa que no está disponible a través de Internet. El proceso de despliegue se lleva a cabo mediante la instalación del software requerido desde las fuentes oficiales de sus desarrolladores y para las siguientes versiones:

- Node.js: versión 14.17.0
- NPM: 6.14.13
- Express: versión 4.17.1
- MySQL Server: versión 8.0.26

En primer lugar, se instala el servidor de la base de datos, creando y configurando la base de datos con sus tablas utilizando el modelo ilustrado en el capítulo 4. Seguidamente se descargan las librerías y dependencias requeridas para el proyecto a través de NPM. Una vez descargadas las dependencias, se inicia la aplicación a través de un script insertado en el fichero de configuración del proyecto. Esto provoca que se levante un proceso Node, el cual ejecuta el archivo principal de la aplicación, y expone el servidor en el puerto y dominio indicados en las variables de entorno. En este punto, nuestra aplicación está lista para ser utilizada.

Para comprobar el funcionamiento de la aplicación, se ha creado un usuario cliente de prueba al cual se ha asignado un dominio administrado por la empresa en el portal Stackpath. Esta comprobación se ha realizado mediante pruebas de caja negra, técnica de testing que consiste en proporcionar un conjunto de valores de entrada, para posteriormente observar y verificar la salida que genera el sistema, sin importar cuales sean las operaciones internas que este realice. Este tipo de prueba nos facilita el trabajo a la hora de verificar el funcionamiento global de la aplicación, a la vez que estas pueden ser realizadas por usuarios nominales; sin embargo, presentan algunos inconvenientes como es el uso de casos de prueba generales, que pueden no incluir casos de uso específicos.

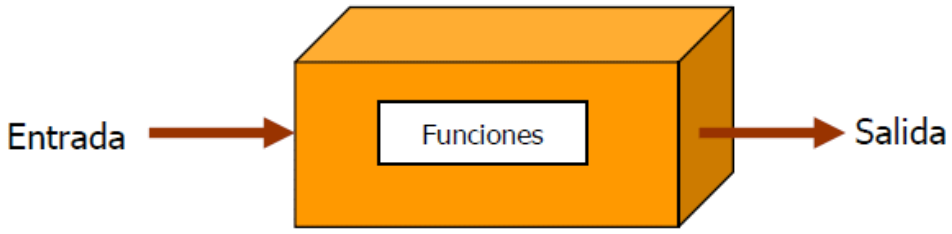


*Ilustración 21 - Flujo prueba de caja negra*

Veamos a continuación los resultados de las pruebas realizadas mediante una tabla que recoge los valores observados:

| Prueba | Entrada | Salida | Valor esperado |
|---|---|---|---|
| Invalidar caché | Domain: viajesenlaweb.com<br>Path: /<br>Recursive: true | Notificación: Invalidación completada | Notificación: Invalidación completada |
| | Domain: viajesenlaweb.com<br>Path: /<br>Recursive: false | Notificación: Invalidación completada | Notificación: Invalidación completada |
| | Domain: viajesenlaweb.com<br>Path: /<br>Recursive: false | Notificación: Invalidación completada | Notificación: Invalidación completada |
| | Domain: viajesenlaweb.com<br>Path: /<br>Recursive: false | Notificación: Invalidación completada | Notificación: Invalidación completada |
| Visualización métricas CDN | Granularity: 1 hour<br>Start: 22/11/2021 18:50<br>End: 23/11/2021 18:50 | Representación de las métricas CDN mediante gráficas. | Representación de las métricas CDN mediante gráficas. |
| | Granularity: 1 hour<br>Start: 24/11/2021 20:30<br>End: 23/11/2021 20:30 | Notificación error: error al obtener las métricas. | Notificación error: error al obtener las métricas. |
| | Granularity: 1 day<br>Start: 21/11/2021 10:20<br>End: 23/11/2021 10:20 | Representación de las métricas CDN mediante gráficas. | Representación de las métricas CDN mediante gráficas. |
| | Granularity: 1 day<br>Start: 21/11/2021 00:10<br>End: 23/11/2021 00:10 | Representación de las métricas CDN mediante gráficas. | Representación de las métricas CDN mediante gráficas. |
| | Granularity: 1 month<br>Start: 21/11/2021 18:50<br>End: 23/11/2021 18:50 | Representación de las métricas CDN mediante gráficas. | Representación de las métricas CDN mediante gráficas. |
| | Granularity: 5 min<br>Start: 22/11/2020 00:00<br>End: 23/11/2020 00:00 | Notificación: no hay métricas disponibles. | Notificación: no hay métricas disponibles. |

| | | | |
|---|---|---|---|
| Visualización métricas WAF | Traffic type: ALL TRAFFIC<br>Start: 22/11/2020 18:00<br>End: 23/11/2020 18:00 | Representación de las métricas y reglas WAF mediante tablas. | Representación de las métricas y reglas WAF mediante tablas. |
| | Traffic type: LEGIT<br>Start: 22/11/2020 18:00<br>End: 23/11/2020 18:00 | Representación de las métricas y reglas WAF aceptadas mediante tablas. | Representación de las métricas y reglas WAF aceptadas mediante tablas |
| | Traffic type: BLOCKED (CUSTOM)<br>Start: 22/11/2020 18:00<br>End: 23/11/2020 18:00 | Representación de las métricas y reglas WAF bloqueadas mediante tablas. | Representación de las métricas y reglas WAF bloqueadas mediante tablas. |
| | Traffic type: LEGIT<br>Start: 25/11/2020 18:00<br>End: 23/11/2020 18:00 | Notificación error: error al obtener las métricas. | Notificación error: error al obtener las métricas. |
| Deshabilitar WAF | Checkbox: false | Recarga de la página, deshabilitando las opciones de toda la sección. | Recarga de la página, deshabilitando las opciones de toda la sección. |
| Habilitar WAF | Checkbox: true | Recarga de la página, habilitando las opciones de toda la sección. | Recarga de la página, habilitando las opciones de toda la sección. |
| Deshabilitar monitoring | Checkbox: false | Recarga de la página, deshabilitando opción de monitorización en Stackpath. | Recarga de la página, deshabilitando opción de monitorización en Stackpath. |
| Habilitar monitoring | Checkbox: true | Recarga de la página, habilitando opción de monitorización en Stackpath. | Recarga de la página, habilitando opción de monitorización en Stackpath. |

*Tabla 13 - Pruebas de caja negra*

# 7. CONCLUSIONES

De acuerdo con lo expuesto a lo largo de esta memoria podemos afirmar que se han alcanzado los objetivos planteados inicialmente. Este proyecto deja abierta una amplia gama de implementaciones para la expansión de la herramienta, como la inclusión de otras funcionalidades de Stackpath para que los clientes puedan usar. Al mismo tiempo, queda pendiente para el futuro el despliegue en un entorno privado de la herramienta, accesible solo para determinados usuarios a través de Internet.

El desarrollo diario ha implicado el conocimiento de nuevas tecnologías y sus funcionalidades, pero también han surgido imprevistos. Todo nuevo conocimiento requiere de una previa investigación y para este proyecto, se ha contado para todos los casos con documentación, oficial y extraoficial, para todas las tecnologías utilizadas, consiguiendo resolver todos los problemas. Además, durante el desarrollo de este también se han ido presentado diferentes errores. Entre ellos, cabe destacar el tardío entendimiento de las sesiones en Stackpath, el cual desembocó en la reestructuración del proyecto cuando este había comenzado su etapa de desarrollo.

A pesar de los problemas presentados, finalmente se ha obtenido el objetivo principal, y es que los clientes ahora pueden acceder a sus recursos desplegados en Stackpath a través de este portal.

Este proyecto ha supuesto un gran reto a resolver, pues si se tienen en cuenta las materias cursadas durante el grado, se abarca un gran abanico de temas estudiados: bases de datos, estudiadas en asignaturas como BDA y TBD; programación y diseño web, vistas en asignaturas como DEW; diseño de interfaces, estudiadas en IPC; estudio de redes, visto en REDES y RCO e incluso leyes sobre protección de datos estudiadas en DYP. A pesar de que toda esta materia ha servido como base para este trabajo, no ha sido suficiente para llevarlo a cabo. El desconocimiento de entornos de ejecución como Node o de motores de plantillas como EJS han dificultado las tareas de desarrollo, las cuales se han conseguido completar a través del esfuerzo y estudio de estas tecnologías.

En definitiva, este proyecto ha conseguido ampliar mis conocimientos y experiencia y me han abierto las puertas al mundo de la programación web con nuevas tecnologías que no había utilizado previamente o desconocía por completo, pudiendo hacer uso de ellas en nuevos proyectos.

# 8. BIBLIOGRAFÍA


[1] Arcodatos., «¿Cómo establecer contraseñas para proteger los datos de la empresa?,» [En línea]. Available: https://www.protecciondedatos.com.es/rgpd/empresas/proteccion-de-datos-lopd-parla/. [Último acceso: 12 11 2021].

[2] Loginradius, «Authorization header,» [En línea]. Available: https://www.loginradius.com/blog/async/everything-you-want-to-know-about-authorization-headers/. [Último acceso: 13 11 2021].

[3] Ciberseguridad.com, «Ciberseguridad de las bases de datos,» [En línea]. Available: https://ciberseguridad.com/guias/bases-datos/. [Último acceso: 12 11 2021].

[4] wikis.fdi.ucm.es, «Especificación de Requisitos Software según el estándar IEEE 830,» [En línea]. Available: https://wikis.fdi.ucm.es/ELP/Especificaci%C3%B3n_de_Requisitos_Software_seg%C3%BAn_el_est%C3%A1ndar_IEEE_830. [Último acceso: 12 11 2021].

[5] Wikipedia, la enciclopedia libre, «Front end y back end,» [En línea]. Available: https://es.wikipedia.org/wiki/Front_end_y_back_end. [Último acceso: 15 11 2021].

[6] Wikipedia, la enciclopedia libre, «GitHub,» [En línea]. Available: https://es.wikipedia.org/wiki/GitHub. [Último acceso: 12 11 2021].

[7] MDN Web Docs, «HTML5,» [En línea]. Available: https://developer.mozilla.org/en-US/docs/Glossary/HTML5. [Último acceso: 14 11 2021].

[8] Wikipedia, la enciclopedia libre, «Node.js,» [En línea]. Available: https://es.wikipedia.org/wiki/Node.js. [Último acceso: 13 11 2021].

[9] Wikipedia, la enciclopedia libre., «Requisito funcional,» [En línea]. Available: https://es.wikipedia.org/wiki/Requisito_funcional. [Último acceso: 12 11 2021].

[10] Cibernos, «Ventajas económicas de los servicios en la nube,» [En línea]. Available: https://www.grupocibernos.com/blog/ventajas-economicas-de-los-servicios-en-la-nube. [Último acceso: 11 11 2021].

[11] Wikipedia, la enciclopedia libre, «Programación por capas,» [En línea]. Available: https://es.wikipedia.org/wiki/Programaci%C3%B3n_por_capas. [Último acceso: 18 11 2021].

[12] Wikipedia, la enciclopedia libre, «Clave foránea,» [En línea]. Available: https://es.wikipedia.org/wiki/Clave_for%C3%A1nea. [Último acceso: 20 11 2021].

[13] Redhat, «¿Qué es una API de REST?,» [En línea]. Available: https://www.redhat.com/es/topics/api/what-is-a-rest-api#rest. [Último acceso: 20 11 2021].

[14] Wikipedia, la enciclopedia libre, «Node.js,» [En línea]. Available: https://es.wikipedia.org/wiki/Node.js. [Último acceso: 21 11 2021].

[15] Wikipedia, la enciclopedia libre, «MySQL,» [En línea]. Available: https://es.wikipedia.org/wiki/MySQL. [Último acceso: 21 11 2021].

[16] Wikipedia, la enciclopedia libre, «HTML,» [En línea]. Available: https://es.wikipedia.org/wiki/HTML. [Último acceso: 21 11 2021].

[17] MDN Web Docs, «JavaScript,» [En línea]. Available: https://developer.mozilla.org/es/docs/Web/JavaScript. [Último acceso: 22 11 2021].

[18] Wikipedia, la enciclopedia libre, «Historia de Internet,» [En línea]. Available: https://es.wikipedia.org/wiki/Historia_de_Internet#Cronolog%C3%ADa. [Último acceso: 25 11 2021].

[19] computerhoy.com, «30 años de la informática doméstica en España,» [En línea]. Available: https://computerhoy.com/noticias/hardware/30-anos-informatica-domestica-espana-21257. [Último acceso: 25 11 2021].

[20] Wikipedia, la enciclopedia libre, «ARPANET,» [En línea]. Available: https://es.wikipedia.org/wiki/ARPANET. [Último acceso: 25 11 2021].

[21] Wikipedia, la enciclopedia libre, «ModeloOSI,» [En línea]. Available: https://es.wikipedia.org/wiki/Modelo_OSI#Capa_de_aplicaci%C3%B3n_-_Capa_7. [Último acceso: 25 11 2021].

[22] Wikipedia, la enciclopedia libre, «World Wide Web,» [En línea]. Available: https://es.wikipedia.org/wiki/World_Wide_Web. [Último acceso: 25 11 2021].

# GLOSARIO

## TÉRMINOLOGÍA

**Cloud Computing**: conjunto de servicios que se proporcionan a través de Internet como alternativas a la tradicional forma de infraestructura. Sinónimo de: servicios en la nube.

**SEO**: conjunto de técnicas que ayudan a los motores de búsqueda a posicionar las páginas web en los buscadores.

**XHR**: interfaz empleada para realizar peticiones HTTP y HTTPS a servidores web.

**Man-in-the-middle**: ataque informático a través de la red que consiste en posicionarse entre dos nodos capturando los datos intercambiados.

**URL**: cadena de texto que identifica un recurso en la red el cual puede variar a lo largo del tiempo.

**JSON**: estructura de datos utilizada principalmente para el intercambio de datos. Está compuesto de pares "clave: valor" separados por comas.

**Variables de entorno**: valor definido en memoria que afecta al comportamiento de los procesos informáticos.

## ABREVIATURAS

**API**: Application Programming Interface (Interfaz de Programación de Aplicaciones)

**CDN**: Content Delivery Network (Red de Distribución de Contenidos)

**WAF**: Web Application Firewall (Firewall de Aplicaciones Web)

**IEEE**: Institute of Electrical and Electronics Engineers (Instituto de Ingenieros Eléctricos y Electrónicos)

**SQL**: Structured Query Language (Lenguaje de Consulta Estructurada)

**HTTP**: HyperText Transfer Protocol (Protocolo de Transferencia de HiperTexto)

**HTTPS**: HyperText Transfer Protocol Secure (Protocolo Seguro de Transferencia de HiperTexto)

**HTML**: HyperText Markup Language (Lenguaje de Marcado de HiperTexto)

**CSS**: Cascading Style Sheets (hojas de estilo en cascada)

**SaaS**: Software as a Service (Software como servicio)

**IaaS**: Infrastructure as a Service (Infraestructura como servicio)

**PaaS**: Platform as a Service (Plataforma como servicio)