



Universidad Politécnica
de Madrid

**Escuela Técnica Superior de
Ingenieros Informáticos**



Grado en Ingeniería Informática

Trabajo Fin de Grado

Infraestructura de registro de mensajes de WiFi

Autor: Carlos Bonilla Gómez
Tutor: Miguel Jimenez Gañan

Madrid, FEBRERO-2024

Este Trabajo Fin de Grado se ha depositado en la ETSI Informáticos de la Universidad Politécnica de Madrid para su defensa.

Trabajo Fin de Grado

Grado en Ingeniería Informática

Título: Infraestructura de registro de mensajes de WiFi

FEBRERO-2024

Autor: Carlos Bonilla Gómez

Tutor: Miguel Jimenez Gañan

Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software

Escuela Técnica Superior de Ingenieros Informáticos

Universidad Politécnica de Madrid

Índice general

1. Introducción	2
1.1. Motivación del proyecto	2
1.2. Contexto del proyecto	2
1.3. Objetivos	3
1.4. Estructura del Documento	3
2. Trabajo relacionado y Estado del Arte	5
2.1. Estado del arte	5
2.2. Herramientas utilizadas	6
3. Desarrollo	10
3.1. Puesta en marcha de las sondas	11
3.1.1. Esquema de conexión	11
3.1.2. Configuración / Especificación de las sondas	12
3.1.3. Complicaciones con las sondas	13
3.1.4. Uso de airodump-ng	14
3.1.4.1. Modo de utilización de airodump-ng	14
3.1.4.2. Ejemplos de Uso	15
3.1.4.3. El archivo generado	15
3.2. Especificación de requisitos	17
3.2.1. Requisitos iniciales	17
3.2.2. Requisitos no funcionales	17
3.2.3. Roles de la aplicación	17
3.3. Arquitectura del proyecto	19
3.3.1. Diseño de Sistema	19
3.3.2. Patrones de Diseño	20
3.4. Desarrollo de la API	23
3.4.1. Descripción general de la API	23
3.4.2. Endpoints de la API	24
3.4.2.1. Probes	24
3.4.2.2. Probes Groups	26
3.4.3. Componentes	30
3.5. Base de Datos	33
3.5.1. Estructura de la base de datos	33
3.5.2. Tablas y Columnas	34
3.6. Servidor	36
3.6.1. Implementación de métodos	37
3.6.1.1. Models	39

3.6.1.2. Controllers	42
3.7. Desarrollo del front-end	48
3.7.1. Componentes	48
4. Pruebas de ejecución	57
4.1. Definición de pruebas	58
4.2. Realización de pruebas	59
4.2.1. Conclusiones de las pruebas	74
5. Impacto del trabajo	75
5.1. Impacto general	75
5.1.1. Impacto personal	75
5.1.2. Impacto empresarial	76
5.1.3. Impacto social	76
5.1.4. Impacto económico	77
5.1.5. Impacto medioambiental	77
5.1.6. Impacto cultural	77
5.2. Objetivos de Desarrollo Sostenible	77
6. Resultados y conclusiones	79
6.1. Resultados obtenidos y dificultades encontradas	79
6.2. Trabajo futuro	80
6.2.1. Implementación de un sistema de login	80
6.2.2. Mejora del diseño del front-end	80
6.2.3. Mejora del componente de visualización de sondas	80
6.2.4. Mejora del escaneo	81
6.2.5. Mejora de los endpoints de la API	81
6.2.6. Aumento de funcionalidades de la API	81
6.2.7. Adición de soporte para nuevos programas	81
Bibliografía	82

Resumen

Desde que el estándar 802.11 se implantó ha habido grandes fallos de seguridad, entre ellos en el que se enfoca este trabajo, los “greedy attacks” en el protocolo CSMA/CA ocurren cuando un nodo no respeta el número aleatorio del backoff, ocupando el canal de manera indebida y dificultando la conexión de otros dispositivos en la red. Para poder detectar los tiempos de backoff, hemos decidido analizar los paquetes que se envían por los dispositivos finales ya que, mediante la detección de portadora, se puede identificar el tiempo que se ha esperado para enviar la trama a la siguiente dirección en una sección conocida como NAV. Para ello, vamos a necesitar una serie de sondas que recogan estos paquetes y las cuales van a ser controladas por una infraestructura de registro de mensajes WiFi la cual nos ayude a mandar señales, recibir información, archivar datos y procesar las diferentes tramas que consigamos. Esto va a ser el punto central de este Trabajo de Fin de Grado, donde nos centraremos en como se ha conseguido implementar un servicio web que haga de centro de control con las sondas, pudiendo operar con ellas al mismo tiempo que guarda información sobre las capturas que realicen las sondas, todo esto controlado por una API. En el final se comentará cuales han sido las dificultades que se han encontrado a la hora de realizar este trabajo, además de cuales son las virtudes y desventajas que se han detectado en su realización y se darán indicaciones para futuros trabajos que necesiten basarse en una estructura similar o que puedan hacer uso de esta misma. Este proyecto pertenece a uno más grande el cual trata todo el proceso desde la gestión de sondas, la captura de información, el análisis y la visualización de los resultados. Mi trabajo recogería principalmente la parte de gestión de sondas y una parte en la captura de la información. **Palabras Clave:** CSMA/CA, API, Probes, Packet Sniffing, Database.

Capítulo 1

Introducción

En este capítulo, se tiene el objetivo de introducir el por qué de este trabajo, dar contexto a la realización del mismo, los objetivos que se van a desarrollar y la estructura que se pretende seguir en el trabajo.

1.1. Motivación del proyecto

Este proyecto surge por la necesidad de analizar el protocolo de comunicaciones via WIFI llamado CSMA/CA. Este protocolo es un mecanismo de contienda que permite a múltiples estaciones repartirse un medio compartido, es decir, el espacio radioeléctrico de un canal. El CSMA/CA es el responsable de comprobar que un canal está vacío antes de mandar un mensaje a través de él. Si no es así, el protocolo establece un número aleatorio de tiempo llamado backoff para esperar y volver a comprobar que el canal este vacío para mandar el mensaje o volver a reiniciar el backoff. Esto se hace con el fin de no sobrecargar el canal y para mantener la integridad de los mensajes. Este protocolo acarrea un problema, el backoff puede ser aprovechado a través de la técnica de backoff jamming, la cual provoca una denegación de servicio y en consecuencia la nula utilización del medio. Otro ataque conocido se centra en la alteración del NAV, en el cual una entidad maliciosa manda tramas con vectores NAV extremadamente grandes, negando a otros dispositivos la utilización del canal durante periodos prolongados. Por ello se ha decidido crear un sistema que, a través de sondas capturen las señales de WIFI, se compruebe los diferentes paquetes en busca de posibles violaciones del protocolo CSMA/CA.

1.2. Contexto del proyecto

Fue en 2004 [1] cuando la Universidad de Queensland descubrió por primera vez estos ataques y su potencial amenaza para las redes, especialmente las públicas. Desde entonces, varios estudios han propuesto soluciones, incluyendo técnicas basadas en inteligencia artificial para detectar comportamientos "greedy". Nosotros adoptamos un enfoque diferente: detectar estos ataques maliciosos externamente al router o punto de acceso, sin depender de soluciones no estandarizadas. Para implementar nuestra solución, se requiere una infraestructura de registro de mensajes WIFI que permita no solo obtener la información registrada por las sondas, sino también mane-

Introducción

jarlas, crear grupos y mantener un control superior para registrar todos los paquetes importantes para su tratamiento futuro.

1.3. Objetivos

El objetivo principal de este trabajo es la creación de una infraestructura de registro de tramas WIFI la cual actúe como centro de control. Para ello primero se deberá realizar una primera investigación sobre los mecanismos de comunicación y la puesta en marcha de las sondas. Después se necesitará comunicar las diferentes sondas que se instalen con el usuario final que necesite operarlas, dándole control total sobre las diferentes operaciones que designemos. Para conseguir este objetivo será necesario realizar los siguientes objetivos:

- Investigación sobre las diferentes herramientas, técnicas y protocolos que se tienen que utilizar a lo largo del Trabajo de Fin de Grado.
- Puesta en marcha de las sondas, realizando la configuración inicial para su conexión.
- Creación de una API REST al completo para la correcta utilización de las sondas disponibles, detallando todo tipo de métodos, parámetros y respuestas.
- Creación del backend del sistema que permita ejecutar las operaciones de la API
- Creación de una interfaz web para poder interactuar visualmente con la API, facilitando el uso de esta y ampliando las posibilidades de desarrollo.

1.4. Estructura del Documento

Este documento se compondrá de varios capítulos, los cuales tienen la misión de hacer entender al lector la gravedad del problema, así como también la solución expuesta, métodos utilizados y programas desarrollados. Para alcanzar este objetivo, el documento dispondrá de las siguientes secciones:

- Introducción: Parte inicial del documento donde se explican los motivos y se contextualiza el trabajo, además de explicarse la relevancia del tema y se crean unos objetivos que se quieren alcanzar para su finalización.
- Trabajo relacionado y Estado del Arte: Segundo capítulo del Trabajo de Fin de Grado donde se exponen las literatura relacionada con el tema donde se identifica la teoría y los conceptos relevantes. Además se desarrolla en las herramientas utilizadas durante la realización, explicando el porque de cada una.
- Desarrollo: Tercer capítulo, dedicado a la descripción del trabajo realizado. Este está compuesto por diferentes apartados y subapartados los cuales explican en profundidad los pasos y decisiones que se han ido tomando a lo largo del Trabajo de Fin de Grado.
- Impacto del trabajo: Cuarto capítulo en el que se encuentra un análisis del potencial impacto que puede acarrear este trabajo, centrandose en que es lo que se ha logrado conseguir/que problema se ha solucionado. También se incluye un apartado dando a conocer los objetivos cumplidos respecto a los Objetivos de Desarrollo Sostenible (ODS).

- Resultado y conclusiones: Quinto capítulo en el que se realiza una discusión a fondo sobre los resultados obtenidos, comparandolo con otros trabajos del campo e indicando si se han cumplido todos los objetivos establecidos. También se centra en las limitaciones que puede presentar este trabajo, ayudando y guiando a futuros investigadores para su resolución futura.
- Referencias: Séptimo capítulo donde se exponen los trabajos consultados para la realización de este documento en notación Bibtex para su consulta.
- Apéndice: Octavo capítulo el cual expone los diferentes materiales usados en el estudio como imágenes, gráficas o líneas de código.

Capítulo 2

Trabajo relacionado y Estado del Arte

Para este capítulo, primero hablaremos del estado del arte en los diferentes aspectos del Trabajo de Fin de Grado, junto con referencias para poder navegar hasta los artículos. En segundo lugar tendremos un recopilatorio de las herramientas utilizadas para la realización del Trabajo de Fin de Grado, con una explicación de por qué se han elegido y como se han utilizado.

2.1. Estado del arte

Como se ha expuesto en el capítulo anterior, el backoff jamming es un problema potencialmente significativo en las redes, ya que puede interrumpir las conexiones de otros dispositivos con el punto de acceso asignado, inhabilitando el uso de la red y causando pérdidas de paquetes. Esta forma de interferencia deliberada se basa en explotar el mecanismo de retroceso (backoff) del protocolo CSMA/CA, que es fundamental para evitar colisiones en redes inalámbricas.

El impacto del backoff jamming es particularmente peligroso en el contexto de las redes IoT (Internet de las Cosas), donde una gran cantidad de dispositivos depende de una conexión estable a Internet. Estos dispositivos van desde asistentes de voz y termostatos inteligentes hasta infraestructura más crítica como cámaras de seguridad, sensores de movimiento y sistemas de monitoreo de salud. La interrupción de estos dispositivos puede tener consecuencias graves, desde la pérdida de funcionalidad en el hogar inteligente hasta riesgos significativos en la seguridad y el bienestar de las personas.

Un estudio destacado muestra cómo un solo "nodo codicioso" (greedy node) puede disrumpir una red entera, previniendo el envío de paquetes al punto de acceso. Este nodo actúa de manera egoísta, monopolizando el canal de comunicación y obligando a otros dispositivos a esperar indefinidamente, lo que efectivamente paraliza la red. La investigación de Sadek et al. (2022) demuestra la facilidad con la que se puede implementar este tipo de ataque y los desafíos que plantea para la seguridad de las redes inalámbricas.[2].

Para solventar este hecho, se creó un proyecto que utiliza varias sondas para capturar

tramas enviadas en diferentes redes de un entorno, permitiendo su análisis mediante la detección de portadora y su representación. Este trabajo de fin de grado forma parte de dicho proyecto, donde decidimos crear una infraestructura de registro de mensajes WiFi que actúa como centro de control para las sondas desplegadas. Este sistema centralizado y flexible guarda los archivos capturados por las sondas para su análisis posterior.

Una de las grandes ventajas de este proyecto es la flexibilidad y centralización que aporta, ya que todo el control de las tramas se gestiona desde una única plataforma. El sistema se compone principalmente de una API que, a través de un servidor y una base de datos, maneja cualquier operación relevante para la captura, como enviar comandos a las sondas. Además, utiliza una interfaz gráfica para facilitar la comunicación con los usuarios y el uso efectivo de la API.

2.2. Herramientas utilizadas

Para poder empezar el desarrollo de nuestra aplicación web necesitamos una API (Application Programming Interface), que es un estilo arquitectónico para sistemas de software distribuidos reconocido globalmente. En el estudio [3] de las API nos encontramos con un planteamiento que compara los servidores que funcionan a través de sockets (los cuales eran muy prevalentes en el año 2007) frente a una estructura tipo API. Como menciona el artículo, las API están orientadas a servicios donde no siempre ocurre la situación de una conexión host-host y el servicio no siempre tiene porque saber los detalles de cada transferencia de información como nombres de host, direcciones y flujos de bytes; en su lugar, deberían tratar con datos/servicios directamente nombrados y unidades de datos. Es por tanto que las API son actualmente la mejor forma de proceder para poder crear un tipo de servicio web para poder controlar las sondas desde el equipo que deseemos.

Existen numerosos tipos de servicios web [4] para operar una API, aunque los principales son REST (Representational State Transfer) y SOAP (Simple Object Access Protocol). La principal diferencia entre estos es que SOAP es un protocolo subyacente que permite la comunicación entre aplicaciones que utilizan diferentes lenguajes y plataformas, y cuenta con estándares de cumplimiento integrados, como WS-Security, lo cual es importante en nuestro estudio para analizar los paquetes enviados y recibidos de forma anónima. Sin embargo, SOAP se basa en el intercambio de mensajes en XML y está diseñado para aplicaciones que requieren un alto nivel de seguridad, como transacciones bancarias, lo que resulta en un mayor tiempo de envío y respuesta, ralentizando la aplicación. Por otro lado, REST es utilizado por aplicaciones más livianas y permite comunicaciones más eficientes, además de estar basado en recursos, lo que facilita el acceso, la adición y la modificación de estos. Dado que necesitamos un manejo eficiente de recursos para operar las sondas con comodidad, REST es la opción más adecuada en este contexto. La seguridad que ofrece es menor, pero suficiente, ya que utiliza HTTPS, que es adecuado para nuestro caso a pesar de sus limitaciones [5].

Para poder realizar una API, hace falta una herramienta que nos ayude a modelar la API. Esto es importante para poder establecer las bases implementación y la integración de la API en aplicaciones cliente y, en un futuro, para poder entregar una documentación completa y fácil de entender para el uso del servicio web. Para ello

existen numerosas formas de crear este tipo de documentos aunque nos vamos a centrar en 3: Swagger, RAML, API Blueprint. Según este estudio [6], estos son los lenguajes de modelización mas comunes y cada uno tiene sus ventajas y desventajas.

En primer lugar, swagger (o también conocido actualmente como OpenApi) nos sirve para crear un documento el cual lista los diferentes recursos en los que se puede acceder y las operaciones que se pueden realizar sobre esos recursos. Además nos puede especificar que contenidos añadir en el cuerpo para poder hacer las llamadas de forma satisfactoria. Swagger trae consigo varias herramientas, entre las que podemos encontrar un editor junto con el documento para poder ver el resultado, una UI la cual sirve para poder probar la API que hemos construido, pudiendo cambiar el cuerpo e inspeccionar el resultado de la llamada y por último Codegen el cual nos genera un SDK (Software Development Kit) con nuestra especificación de API para así no tener que lidiar con el transporte de HTTP.

Por otro lado existe RAML, el cual provee un entorno estructurado y sin amigüedades. RAML es bastante parecido a Swagger, con él puedes describir cualquier API pero tiene una ventaja, gracias a la modularización que existe en RAML, se pueden reutilizar partes del código escrito en otras partes para así mantener la coherencia y así poder no duplicar líneas de código. Esto es interesante en un proyecto grande con una API muy extensa.

En ultimo lugar API Blueprint, que se va a ser descartado ya que se centra en la colaboración entre diferentes entidades como developers, stakeholders y clientes lo cual sobrepasa la magnitud de esta proyecto.

El lenguaje mas apropiado sería Swagger con Open Api Specification, principalmente por ser el más reconocido tanto institucional como comúnmente pero también por la gran comunidad que lleva consigo la cual ayudará a resolver las dudas que surgan en la realización. Otro punto a favor son las herramientas que este lenguaje trae implementadas, las cuales ayudarán a luego la realización de código y a la hora de probar los diferentes métodos. La API que vamos a crear no prevee ser muy extensa, por tanto no van a existir duplicaciones de código, descartando el uso de RAML.

Para poder utilizar la API vamos a necesitar de algun framework que actue como backend y pueda comunicarse con nuestra API, preferiblemente en el lenguaje de programación de Python ya que muchos de los programas que vamos a hacer están escritos en Python, además de la gran cantidad de información con la que vamos a manejar de todas las sondas mandando continuamente su información va a ser un gran trabajo poder manejarla. Esto con Python se vuelve una tarea mas sencilla [7]. Existen numerosos frameworks en Python pero los más interesantes por su facilidad de uso y la comunidad que tienen detrás son dos, Flask y Django [8]. Este estudio da un pequeño resumen de todos los frameworks que ofrecen servicios para desarrollar con APIs pero termina haciendo una comparativa entre los dos que nos conciernen. Por un lado, Flask es un framework que se caracteriza por una mayor simplicidad. Su premisa esta en la flexibilidad que este ofrece y te permite un mayor control sobre toda la aplicación. El problema recae en que como tal, Flask no tiene servicios como una conexión a base de datos o una autenticación pero se puede solventar utilizando paquetes que vienen hechos para Flask, tales como SQLAlchemy para la base de datos, Flask-security para la seguridad de la app en general o Flask-Login para las autenticaciones.

Por el lado de Django, este framework viene totalmente equipado, con todo tipo de

librerías para hacer conexiones a bases de datos, autenticaciones y demás funcionalidades que quieras tener para tu servicio web además de tener un gran foco en la seguridad. El problema de Django es el hecho de ser más complejo de entrada y orientado para un tipo de aplicaciones grandes donde se manejen muchos endpoints. En nuestro caso, el servicio web que vamos a desarrollar pese a ser grande ya que requiere una gran carga de trabajo si necesitamos manejar varias sondas al mismo tiempo, además de necesitar la base de datos, servicios de login y seguridad en general sobre toda la aplicación, vamos a decantarnos por Flask ya que con los diferentes paquetes que incorpora, es capaz de manejar esto y su fácil sintaxis va a hacer que este proyecto sea mucho más sencillo ya que se va a requerir mucha personalización en los diferentes métodos, aspecto que Django no nos ofrece. Esto sumado a que está diseñado para trabajar con una API REST convierte a Flask en nuestro candidato ideal para el desarrollo del backend en nuestra aplicación.

Para hacer el despliegue, hay distintas opciones pero la más reconocida y más utilizada es Apache Web Server. El Proyecto Servidor HTTP Apache [9] es un esfuerzo colaborativo de desarrollo de software cuyo objetivo es crear una implementación de código fuente de un servidor HTTP (Web) robusto, de calidad comercial, lleno de características y disponible gratuitamente. Este es capaz de manejar grandes volúmenes web sin ningún problema y nos vamos a servir de su potencia para poder enrutar solicitudes y actuar como intermediario para webs dinámicas lo cual nos va a ayudar a poder utilizar la aplicación.

Las sondas necesitan recibir ordenes para poder funcionar. Estas ordenes serán utilizando programas que preinstalemos en estas, los cuales nos ayuden a poder capturar el tráfico inalámbrico para su futuro procesamiento. Para esto nos vamos a servir de dos herramientas que, pese a ser parecidas, nos van a proporcionar información valiosa. Estas herramientas se conocen como airodump-ng[10] y tcpdump[11] y son las herramientas más utilizadas en el mundo de los packet sniffers o “análisis de paquetes”. Con estas herramientas vamos a poder diseñar comandos que, por defecto, cambien las sondas de modo para poder monitorear el aire, utilizando también el backend con Flask como elemento de personalización de los comandos, pudiendo este modificar distintos parámetros y filtros que se utilicen. Airodump-ng será utilizado principalmente para poder monitorizar las redes que están disponibles, canales que se pueden acceder y tipos de paquetes que se están enviando, tanto desde el punto de acceso al dispositivo final como viceversa. Tcpdump será utilizado por otro proyecto del grupo en el apartado de captura, ya que es el programa más eficaz en este apartado, además de que tcpdump nos ayudará a cubrir las lagunas en las que airodump-ng no pueda actuar.

La idea principal es que el usuario final de la aplicación no tenga que interactuar de forma directa con estos comandos, sino a través de una interfaz la cual facilite el uso de esta librerías.

La API está pensada para poder ser utilizada directamente desde la consola dado que las sondas van a necesitar interactuar con ella directamente llamando a los endpoints correspondientes. Sin embargo, no es usual para las personas comunicarse de esta forma [12] con los diferentes servicios que se ofrecen y por tanto es importante el uso de una herramienta front-end. Esta herramienta nos va a permitir tener un servicio web mucho más usable y accesible con el fin de facilitar la interacción con las sondas y su posterior registro. Ya que el front-end actúa como cliente, al

fin y al cabo queremos realizar un servicio web, solo va a limitarse a recurrir de los endpoints establecidos y utilizar la informacion que estos le regresan, asi pudiendo utilizar muchos proveedores de este tipo de herramientas. Esta informacion vendrá dada en forma de JSON cuya especificacion viene dada por la parte de Swagger/Open API Specification ya mencionado. Existen diversas herramientas en el mercado para poder hacer servicios web. La más extendida se conoce como JavaScript y de este muchos frameworks famosos han emergido, aunque los mas conocidos son React, Angular y Vue. Estos 3 frameworks prevalecen por su utilidad, apoyo por parte de la comunidad y facilidad de uso, haciendo de estos referentes para la creacion de servicios web [13]. Por una parte, Angular fue creado por desarrolladores de google. Su propuesta fue tan relevante que el producto fue hecho público en el año 2010 y fue una de las primeras en adoptar el Modelo-Vista-Controlador. Inicialmente el producto fue lanzado en el lenguaje JavaScript aunque en 2016 salió una nueva version completamente diferente a esta, con la peculiaridad de utilizar el lenguaje TypeScript. Nosotros vamos a analizar la version de TypeScript ya que es el mas nuevo y otra posibilidad a tener en cuenta de cara al proyecto. En cuanto a características, Angular es de los tres el framework menos popular, tanto en GitHub como en StackOverflow. Esto puede deberse a que aprender un lenguaje nuevo como es TypeScript es complicado ya que primero requieres de una solida base de JavaScript para poder entender este lenguaje. Esto le hace tambien el peor en terminos de facilidad de aprendizaje. En el apartado de rendimiento, Angular esta pensado para utilizarse en servicios web grandes los cuales requieran de mucho computo de información por parte del front-end [14]. Por otro lado tenemos React, este framework utiliza JSX que es una version extendida de JavaScript. React fue creado por la compañía Facebook (ahora Meta) en el año 2013. Este es uno de los frameworks mas utilizados y populares actualmente, siendo uno de los repositorios con más estrellas de toda la plataforma GitHub y uno de los mas preguntados de estos 3 frameworks en StackOverflow. Gracias a el JSX el cual permite la creacion de estructuras HTML directamente en JavaScript, su arquitectura basada en componentes la cual facilita la escalabilidad en el código y su "Virtual DOM" el cual se limita las actualizaciones de componentes, ayudando con el rendimiento, React es una herramienta muy útil a la hora de realizar servicios web y gracias a su flexibilidad, es capaz de adaptarse a cualquier tipo de aplicación, tanto grande como pequeña. Para finalizar, Vue fue creado en 2013 y desde entonces ha ganado mucha tracción, convirtiendose en el proyecto mas popular de los tres, siendo uno de los proyectos mas destacado de todo Github, aunque no supera a React en StackOverFlow. En cuanto a facilidad, Vue es de los tres el más facil de aprender gracias a su sistema de plantillas basadas en html lo que facilita el aprendizaje inicial. Posee también componentes simples y faciles de crear lo cual ayuda a la hora de tener el codigo limpo. En el apartado de rendimiento, Vue es muy competente haciendo proyectos de pequeña escala, habiendo sido 5 veces más rapido que Angular. Para nuestra situación, ya que queremos crear una aplicación mediana, con un número elevado de llamadas al backend, necesitamos una herramienta robusta capaz de manejar esta carga de datos. Para este caso, React es una buena elección, no solo por las funciones que este trae consigo como el Virtual DOM u otras características explicadas anteriormente, sino también por la gran comunidad que este trae consigo, donde infinidad de recursos como tutoriales, recursos y soporte están disponibles para los desarrolladores que lo deseen, haciendo de React una herramienta muy accesible y cómoda para trabajar.

Capítulo 3

Desarrollo

En este capítulo nos vamos a centrar en el proceso de desarrollo que ha llevado el programa, comentado paso a paso de forma cronológica las tecnologías que se han ido utilizando, así con el fin de poder dar a ver el trabajo que se ha realizado, las complicaciones que han surgido durante este, como se han logrado abordar y cual ha sido el resultado final. También es importante recalcar la tarea de estructuración que se ha llevado desde el principio para poder crear un software de calidad. Nuestro objetivo es proporcionar una visión clara y detallada de cada fase del desarrollo, mostrando cómo cada decisión técnica y cada metodología aplicada ha contribuido al producto final.

Exploraremos desde la configuración inicial de las sondas y la infraestructura de red hasta el desarrollo de la API y la implementación del front-end. Este recorrido no solo ilustra las capacidades técnicas utilizadas, sino que también destaca la importancia de una planificación cuidadosa y una ejecución disciplinada para superar los desafíos y alcanzar los objetivos del proyecto.

A lo largo de este capítulo, se pondrá énfasis en cómo se ha estructurado el trabajo para asegurar la calidad y sostenibilidad del software. La integración de prácticas modernas de desarrollo, la gestión eficiente de versiones y la adopción de estrategias de pruebas continuas han sido pilares fundamentales en nuestro enfoque. De esta manera, no solo pretendemos describir el proceso técnico, sino también reflejar el aprendizaje y la adaptación continua que han sido esenciales para el éxito del proyecto.

Finalmente, este capítulo servirá como una documentación exhaustiva del desarrollo, proporcionando una referencia valiosa para futuros proyectos y facilitando el entendimiento y la continuación del trabajo para nuevos integrantes del equipo. La transparencia en cada etapa del desarrollo es crucial para mantener la cohesión del equipo y garantizar que todos los aspectos del proyecto sean bien comprendidos y valorados.

3.1. Puesta en marcha de las sondas

3.1.1. Esquema de conexión

Para empezar el desarrollo de la aplicación, fue necesario la realización de un esquema de conexión de las sondas el cual nos indicara claramente la forma en la que estas intercambian información con nuestro servidor y API REST. Este esquema es fundamental para la comprensión del flujo de los datos y la arquitectura de red a la que nos enfrentamos:

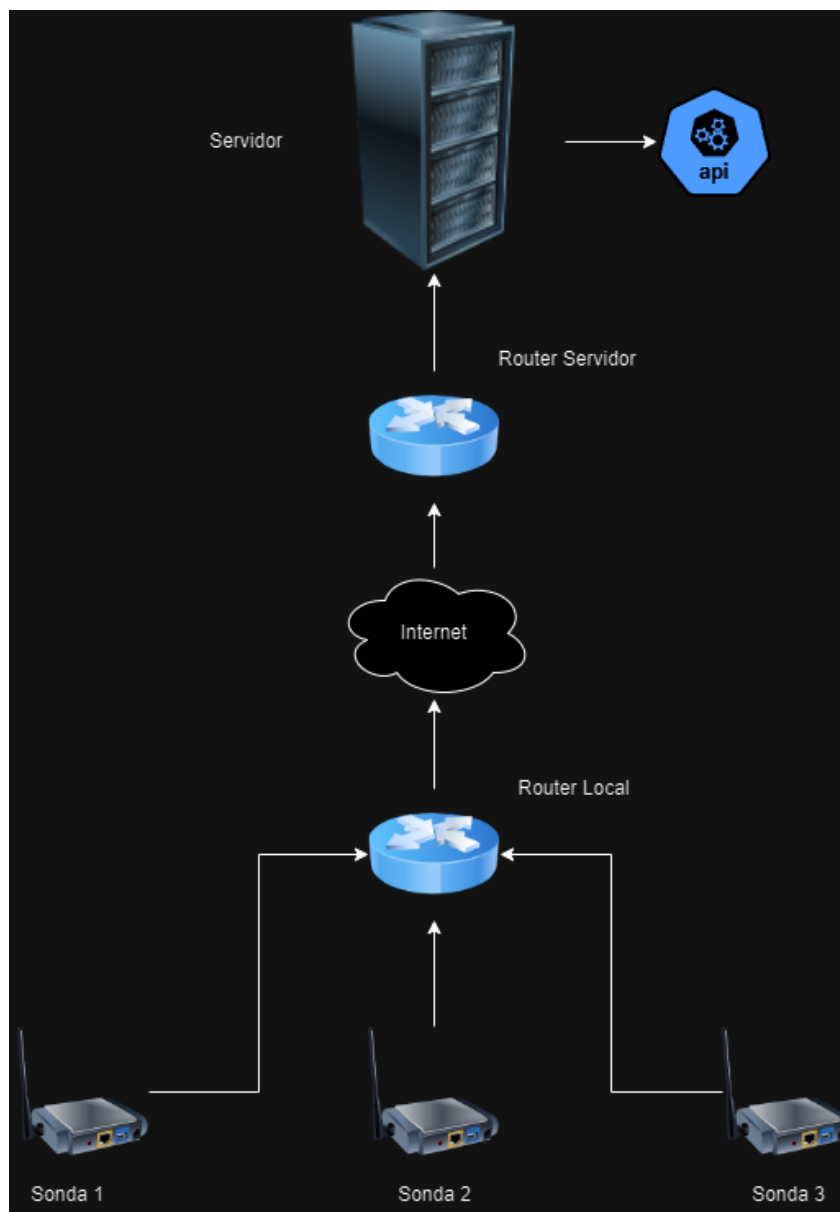


Figura 3.1: Esquema de conexión de las sondas

Este esquema detalla como las sondas son capaces de comunicarse con el servidor a través de una red local hacia el servidor pasando por varias etapas. A continuación describimos las conexiones lógicas y físicas expuestas:

- **Sondas** : Equipadas con módulos de comunicación Wi-Fi y conectadas a una fuente de alimentación. Existe una peculiaridad a la hora de transmitir por IP ya que las sondas se comunican con el router, pero no tienen asignadas direcciones estáticas ya que están pensadas para poder utilizarse en diferentes redes.
- **Router Local** : Las sondas estan conectadas inalámbricamente al router local, que a su vez está conectado físicamente a un módem de Internet. El router local maneja la transmisión de datos desde las sondas utilizando NAT. Este router puede ser cualquiera al que se tenga acceso.
- **Internet** : El módem del router local está conectado a una línea de Internet, permitiendo la transmisión de datos a través de la red global.
- **Router Servidor** : Ubicado en la red del servidor central, este router gestiona el tráfico entrante y saliente. Redirige los datos recibidos a la dirección IP del servidor central utilizando TCP/IP.
- **Servidor central y API** : El servidor central procesa y almacena los datos enviados por las sondas. Aunque tiene una IP pública, el tráfico pasa a través del router de la red del servidor. La API corre en el servidor central y expone los endpoints necesarios para la gestión de los datos, utilizando HTTP/HTTPS para la comunicación.

Una sonda (ya registrada) despues de leer que el trabajo disponible en el endpoint "current job" ha finalizado, se dispone a enviar los datos por la red. Estos datos atravesarán el router local que, tras la traducción NAT, enviará los datos a la dirección del servidor. Después de que el router del servidor detecte el paquete, este sera recibido por el servidor y la API donde se procederán a guardar y almacenar los datos de captura para su posterior análisis

3.1.2. Configuración / Especificación de las sondas

Las sondas que mi equipo y yo hemos utilizado para este proyecto son las Single Board Computer OrangePi Zero 3, en su versión de 1.5GB de memoria. Estas sondas cuentan con un procesador Cortex-A53 de cuatro núcleos H618 de 64 bits de memoria completa, GPU Arm Mali-G31 MP2, y es compatible con OpenGL ES 1.0/2.0/3.2, OpenCL 2.0, Vulkan 1.1 por lo cual son aptas para esta tarea. Esta placa la utilizamos junto a otros dispositivos ya que la funcionalidad que tiene es reducida. Por un lado la tarjeta de red WIFI que viene equipada no posee modo monitorización por tanto utilizamos tarjetas de red que se conectan a la placa mediante USB de la marca Realtek. Por otro lado, no podemos mandar la información por WIFI ya que esto añadiría tramas no deseadas al estudio de la red, lo que provoca lecturas extra no relevantes o erróneas. Para solventar esto, la OrangePi tiene equipado un puerto ethernet por el cual la sonda es capaz de transmitir la información al servicio web. En el apartado de configuración, todas las sondas han sido instaladas con uno de los sistemas operativos Kali Linux o Ubuntu. Ubuntu es uno de los sistemas operativos más populares de windows y nos permite operar con los comandos de airodump de forma eficiente.



Figura 3.2: Tarjeta de red Realtek utilizada

Kali Linux es una distribución basada en Debian (al igual que Ubuntu), con la peculiaridad de estar enfocado al mundo de la ciberseguridad en todos sus aspectos. Cada sonda tiene activado el tunel SSH para poder hacer conexiones directas a esta, comprobar archivos de log, instalación de paquetes y programas además de configuración de varias variables locales. Las sondas tienen por defecto un nombre con la cual poder registrarse y establecerán una ID en la respuesta de este para futuras transacciones. Además de estas características, cuentan con una dirección directa al servidor con los endpoints para poder comunicarse con este. Siempre que las sondas se inicien, intentarán establecer una conexión con el servidor, de tal manera que durante el proceso se registren en el servicio para ser utilizadas.

3.1.3. Complicaciones con las sondas

Una de las principales razones por la cual se decidió que el proyecto fuera un web service, fueron las complicaciones que surgieron al intentar contactar con las sondas. Existen 2 problemas principales:

- **Conexión a través de NAT:** Las sondas siempre estarán conectadas a un router local y, por tanto, no tendrán una IP pública individual. Esto imposibilita la tarea de crear grupos de trabajo y juntar varias sondas, ya que causa problemas al intentar enviar información y comunicarse con una sonda específica dentro del grupo. La tecnología NAT, comúnmente utilizada en los routers, es la causante de este problema. NAT funciona de tal manera que una dirección pública puede representar muchas direcciones privadas, lo que significa que los equipos dentro de una NAT están “escondidos” del público.
- **Falta de permisos de configuración:** Los dispositivos estarán conectados a routers a los cuales tendremos acceso, pero no permisos de configuración ni de administrador. Esto repercute directamente en el funcionamiento de nuestro programa, ya que opciones como “Port Forwarding” o “Universal Plug and Play” (UPnP), que son soluciones rápidas, no podríamos utilizarlas en nuestro caso. Otra técnica que planteamos fue el “Reverse SSH Tunnel”, la cual se descartó por motivos de seguridad, ya que estaríamos exponiendo el servidor principal.

Por estas razones, pensamos que la mejor manera no era intentar contactar con las sondas. Optamos por lo contrario: hacer que las sondas nos contacten a través de un modelo Cliente-Servidor, donde las sondas actúan como clientes y nosotros como el servidor. Este enfoque no solo ayuda a solventar los problemas mencionados, sino que también hace nuestro software mucho más flexible. Las sondas tienen preconfigurado el servidor al que deben conectarse, lo que permite una preparación más

rápida al cambiar las sondas de red o router, consiguiendo así lo que necesitamos.

3.1.4. Uso de airodump-ng

Airodump-ng [10] es una herramienta de monitorización de redes, especializada en la captura y análisis de redes inalámbricas. Esta herramienta nos permite capturar los paquetes emitidos por los puntos de acceso y los clientes, proporcionándonos información relevante sobre estos. En nuestro caso, esta herramienta se va a utilizar para ver que redes existen cerca de las sondas y cuales son los clientes asociados a estas. Airodump-ng es la herramienta principal en la que nos vamos a enfocar. En este proyecto, el objetivo es utilizar esta herramienta en su máximo potencial, ejecutando comandos complejos en múltiples sondas al mismo tiempo de forma automatizada. Esto significa que el aprendizaje de esta herramienta es vital para la correcta realización del trabajo.

3.1.4.1. Modo de utilización de airodump-ng

Airodump-ng puede filtrar mediante numerosos comandos, haciendo de este programa una herramienta bastante completa. Aunque otros programas como tshark pueden realizar tareas similares o más avanzadas, airodump-ng tiene la ventaja de escanear redes, reflejando este comportamiento en sus filtros, aunque sean más concisos. Además, podemos utilizar la información de cada paquete que guarda airodump-ng para analizarla con tshark, haciendo que esta herramienta cumpla su función con creces.

Estos comandos solo se utilizarán al escanear redes. Para las órdenes de monitorización, se utilizará el comando airodump, por lo que no se aplicarán los comandos que se mostrarán a continuación. En otras palabras, los parámetros que se enseñarán ahora funcionan principalmente como filtros de la información total, y no incrementan la cantidad de datos que recibe la sonda.

Este es un pequeño resumen de los diferentes filtros que nos podemos encontrar:

Opción de Filtro	Descripción
-encrypt <suite>	Filtra los APs por la suite de cifrado utilizada.
-netmask <netmask>	Filtra los APs según una máscara de red específica.
-bssid <bssid>	Filtra los APs por su BSSID (dirección MAC del AP).
-essid <essid>	Filtra los APs por su ESSID (nombre de la red).
-essid-regex <regex>	Filtra los APs por su ESSID usando una expresión regular.
-a	Filtra los clientes no asociados.

Opción de Canal	Descripción
-ht20	Configura el canal a HT20 (802.11n).
-ht40-	Configura el canal a HT40- (802.11n).
-ht40+	Configura el canal a HT40+ (802.11n).

-channel <channels>	Captura en canales específicos.
-band <abg>	Especifica la banda en la que airodump-ng debe operar (2.4GHz, 5GHz, etc.).
-C <frequencies>	Utiliza frecuencias específicas en MHz para hacer hopping.
-cswitch <method>	Configura el método de cambio de canal: <ul style="list-style-type: none"> ▪ 0: FIFO (First In, First Out) – Predeterminado. ▪ 1: Round Robin – Cambia de canal en un orden cíclico. ▪ 2: Hop on last – Salta al último canal utilizado.
-s	Similar a la opción -cswitch.
-help	Muestra la pantalla de ayuda con el uso y las opciones de airodump-ng.

Estos son los diferentes filtros en los que nos gustaría centrarnos. De estos, solo el filtro `-essid-regex` será omitido debido a la ligera complejidad que este supone; será añadido en futuras actualizaciones del software.

3.1.4.2. Ejemplos de Uso

A continuación, se presentan algunos ejemplos prácticos de cómo utilizar airodump-ng para capturar datos específicos:

- Capturar todos los paquetes en el canal 6 y guardar la salida en un archivo:

```
airodump-ng --channel 6 --write captura canal6
```

- Filtrar por un punto de acceso específico usando su BSSID:

```
airodump-ng --bssid 00:14:6C:7E:40:80 --write captura_bssid
```

3.1.4.3. El archivo generado

Vamos a comentar el archivo que se genera en formato CSV al iniciar una monitorización sin ningún filtro. Este archivo contiene las siguientes columnas:

BSSID	PWR	RXQ	Beacons	#Data, #/s	CH	MB	ENC	CIPHER	AUTH	ESSID
00:09:5B:1C:AA:1D	11	16	10	0	0	11	54.	OPN		NETGEAR
00:14:6C:7A:41:81	34	100	57	14	1	9	11e	WEP	WEP	bigbear
00:14:6C:7E:40:80	32	100	752	73	2	9	54	WPA	TKIP	PSK teddy

Estas columnas nos muestran la información de las diferentes redes y AP (Access Points o Puntos de Acceso) que existen alrededor de la sonda.

Columna	Descripción	Implicación en Filtros
---------	-------------	------------------------

3.1. Puesta en marcha de las sondas

BSSID	Dirección MAC del punto de acceso.	Directamente relevante para <code>-bssid <bssid></code> y <code>-netmask <mask></code> . Estos filtros permiten centrarse en APs específicos o en rangos de direcciones MAC.
Power	Potencia de la señal del punto de acceso, medida en dBm.	No tiene implicación.
Beacons	Número de beacons enviados por el AP.	No tiene implicación.
Data	Número de paquetes de datos enviados por el AP.	No tiene implicación.
Channel	Canal de radio en el que opera el AP.	Relevante para <code>-channel <channels></code> . Permite filtrar y capturar datos específicamente en canales designados.
Speed	Velocidad máxima soportada por el AP.	No tiene implicación.
Privacy (Encryption)	Tipo de cifrado utilizado por el AP, como WEP, WPA, WPA2.	Relacionado con <code>-encrypt <suite></code> . Este filtro puede ser usado para enfocarse en redes con tipos específicos de cifrado.
Cipher	Algoritmo de cifrado utilizado para proteger las comunicaciones, como CCMP o TKIP.	No tiene implicación directa, pero puede informar decisiones de filtrado similares a Privacy.
Authentication	Método de autenticación, como PSK (Pre-Shared Key) o MGT (gestión).	No tiene implicación.
ESSID	Nombre de la red transmitido por el AP.	Directamente relevante para <code>-essid <essid></code> y <code>-essid-regex <regex></code> . Permite filtrar y capturar datos basados en el nombre exacto o patrones en el nombre de la red.

Como se puede apreciar, este análisis de las tablas nos indica que, en relación a los comandos disponibles en airodump-ng, no existen muchas columnas que nos sirvan para filtrar. A la hora de analizar la información, todas estas columnas pueden sernos de utilidad y, por tanto, todas serán guardadas. Sin embargo, en nuestro caso solo se van a utilizar cuatro columnas: BSSID, ESSID, Channel y Privacy.

Estas columnas se extraerán en un proceso de “monitorización” en el cual se pedirá a la sonda que escanee todas las redes sin ningún filtro, con el fin de poder, posteriormente, filtrar desde el servicio web qué tipo de red u opción serían convenientes para continuar.

Los filtros restantes tienen en común que son booleanos (verdadero o falso) o que aceptan valores dentro de un conjunto cerrado y ninguna de las columnas del documento generado nos dan información sobre ellos, por tanto, se dejará a gusto del

consumidor si desean activarlos o no.

3.2. Especificación de requisitos

3.2.1. Requisitos iniciales

En esta sección nos centraremos en exponer los requisitos para esta aplicación, los cuales deben ser cumplidos. Esto supone un paso crucial para el proyecto, ya que una buena definición de requisitos es indispensable para que todos los componentes de nuestra aplicación sean realizados en consonancia a un esquema general. En los siguientes puntos iremos describiendo como cada requisito es cumplido y en el final se detallarán los requisitos no logrados y nuevos para incluir. La lista de requisitos es la siguiente:

1. **Operaciones CRUD para Sondas:** La aplicación debe ofrecer funcionalidades completas para crear, leer, actualizar y eliminar (CRUD) para cada sonda.
2. **Consulta de trabajo de sonda:** La aplicación debe ofrecer el trabajo que se requiere realizar a la sonda, tomando en cuenta los diferentes tipos de trabajo “scan”, “monitorize” y “stop”
3. **Operaciones CRUD para Grupos de sondas:** La aplicación debe ofrecer funcionalidades completas CRUD para cada grupo.
4. **Gestión de trabajos:** La aplicación debe ofrecer la posibilidad de crear trabajos para los diferentes grupos. Un grupo solo puede realizar un trabajo al mismo tiempo. Estos trabajos deberán poder ser descargados posteriormente para su análisis.

3.2.2. Requisitos no funcionales

Es importante que la aplicación se mantenga en un estándar de funcionamiento, lo cual implica la creación de este tipo de requisitos para mantener un rendimiento adecuado y poder tomar decisiones a partir de ellos.

1. **Rendimiento:** La API debe procesar múltiples solicitudes simultáneamente sin degradar el rendimiento.
2. **Confiabilidad:** La API debe proveer resultados consistentes y estar disponible sin interrupciones significativas.
3. **Usabilidad:** La API debe ser fácil de utilizar para los desarrolladores, con documentación clara y concisa.

3.2.3. Roles de la aplicación

Para definir de forma apropiada el flujo de trabajo, se van a definir una serie de roles o perfiles que se pueden tomar en la aplicación, con el fin de poder diferenciar

ADMIN (Administrador del Sistema):

- Gestión total del sistema, incluyendo usuarios, configuración de seguridad, y actualizaciones.

- Monitoreo y análisis del rendimiento general de la plataforma.
- Resolución de problemas técnicos a nivel de sistema y mantenimiento regular.
- Creación y gestión de políticas y procedimientos para la operación de la plataforma.

CLIENTE (Gestor y Operador de Sondas):

- Gestión de grupos de sondas.
- Operación de sondas para realizar tareas específicas.
- Monitoreo del estado y actividad de las sondas y grupos de sondas.
- Análisis de los datos recogidos y generación de informes.
- Gestión de trabajos asignados a las sondas, incluyendo la subida y descarga de resultados.

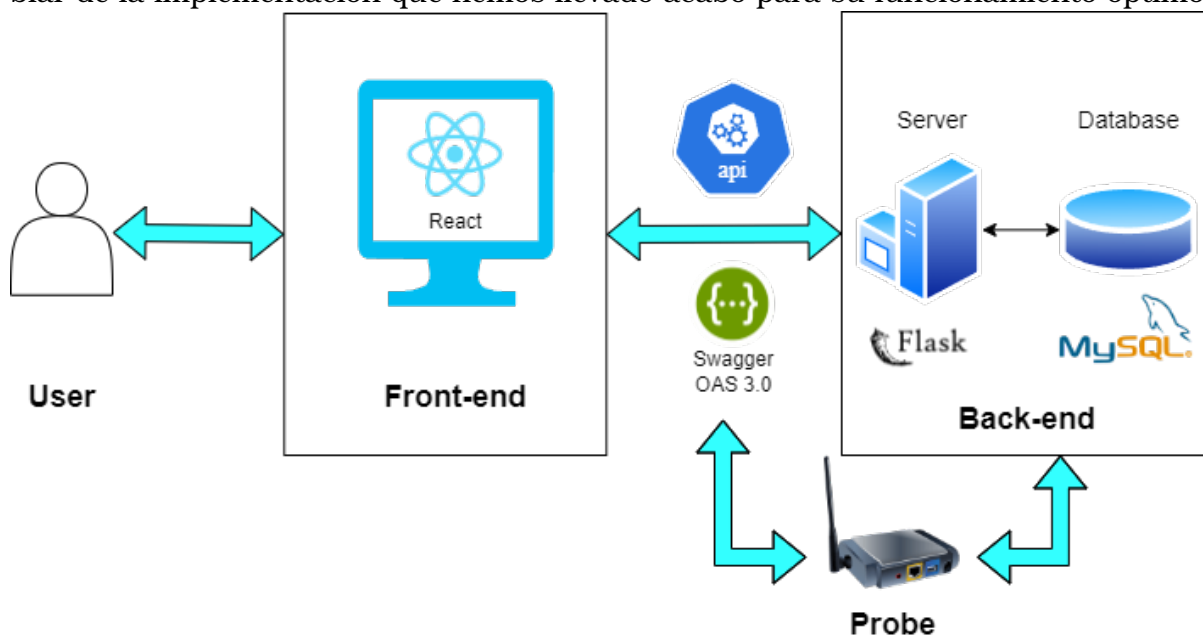
SONDA (Entidad Automatizada):

- Capacidad para registrarse automáticamente en el sistema al inicio o cuando se reintegra después de un mantenimiento.
- Ejecución automática de tareas asignadas según la configuración establecida por los gestores.
- Recolección y transmisión de datos hacia la plataforma.
- Auto-monitorización de su estado, reportando problemas técnicos y necesidades de mantenimiento.

3.3. Arquitectura del proyecto

3.3.1. Diseño de Sistema

La arquitectura del sistema es sencilla, sin embargo, tiene el objetivo de proporcionar a las sondas y usuarios una plataforma robusta y escalable que soporte la monitorización y administración eficiente de las sondas. Esta estructura es la típicamente utilizada en los servicios web del tipo cliente-servidor. Ahora vamos a abordar en detalle las diferentes partes y como se relacionan cada una de ellas para, más adelante, hablar de la implementación que hemos llevado acabo para su funcionamiento óptimo.



En este sistema podemos diferenciar los siguientes componentes:

- **Sonda:** Un dispositivo integrado que recopila datos del entorno y envía esta información al servidor a través de la API para su procesamiento. La sonda es crucial para operaciones específicas donde se requiere monitoreo en tiempo real o recopilación de datos ambientales. Se comunica directamente con el servidor, permitiendo una respuesta rápida y acciones basadas en los datos recogidos.
- **Usuario:** El usuario final interactúa con el sistema a través del frontend, utilizando la interfaz para enviar solicitudes y recibir información. Este componente es esencial para la experiencia del usuario, ya que es el principal punto de contacto con el sistema.
- **Frontend:** La interfaz de usuario creada en React que presenta la información de manera visual y permite al usuario interactuar con el sistema. Es responsable de enviar las solicitudes del usuario al servidor a través de la API y mostrar las respuestas recibidas. Facilita una interacción clara y efectiva, asegurando que las necesidades del usuario se manejen de manera intuitiva.
- **API:** Actúa como el canal de comunicación entre el frontend y/o las sondas con el servidor, especificada por la herramienta OAS 3.0 (Open API Specification). Procesa y dirige las solicitudes y respuestas entre estos dos componentes, asegurando que las interacciones sean fluidas y eficientes. Es fundamental para la integración del frontend y el backend, facilitando una separación clara de

responsabilidades y una mejor escalabilidad del sistema.

- **Servidor:** El centro de procesamiento escrito en Flask del sistema donde se ejecuta toda la lógica de negocio y se gestionan las operaciones de datos. Responde a las solicitudes de la API, ejecuta las operaciones necesarias y devuelve los resultados al frontend a través de la API. Este componente es crucial para mantener la lógica de negocio centralizada y eficiente.
- **Base de Datos:** Almacena toda la información necesaria para el funcionamiento del sistema, incluyendo datos del usuario, configuraciones, y estados de operaciones. El servidor interactúa con esta base para recuperar y almacenar datos, lo que es fundamental para la persistencia y la gestión eficiente de la información.

3.3.2. Patrones de Diseño

El proyecto está diseñado como un servicio web que opera bajo un modelo cliente-servidor, donde tanto las sondas como los usuarios actúan como clientes que interactúan con el servidor. Los clientes realizan solicitudes para obtener recursos del servidor, los cuales son necesarios para alcanzar objetivos específicos. Estas solicitudes se gestionan a través de una interfaz bien definida conocida como API (Interfaz de Programación de Aplicaciones), la cual establece los protocolos de comunicación entre los clientes y el servidor. Esta también es encargada de facilitar esta comunicación, además de definir y restringir las operaciones posibles, asegurando así una interacción efectiva y segura.

Por otro lado, nuestra implementación del modelo cliente-servidor se ha realizado a través del patrón de diseño MVC [15] o Modelo-Vista-Controlador. Este patrón de diseño se basa en tener una estructura de carpetas modular, donde cada modulo es completamente independiente del otro, haciendo cada un responsable de una parte de la aplicación.

- **Modelo:** Tiene la responsabilidad de representar los datos y la lógica empresarial de la aplicación. Cada uno de estos modelos representa típicamente una tabla en de la base de datos de la aplicación. Esta función es llevada a cabo por una parte del servidor denominada "models" (Ver imagen abajo) donde se pueden apreciar como cada archivo representa no solo una tabla de la base de datos, sino también tipos de respuesta, organizados por codigos. Existen clases ORM, que también entran en esta categoría. Su principal función es relacionar una entrada de la base de datos con un objeto creado por el Codegen para poder devolver objetos de forma estandarizada
- **Vista:** Encargada de todos los aspectos visuales de la aplicación. Estos aspectos suelen manejar plantillas de HTML como base, los cuales pueden requerir interacción del usuario para poder enseñar la información requerida. La Vista suele ser pasiva y solo se comunica con el Controlador. En nuestro caso, este rol se desempeña por parte del front-end, y se divide típicamente en componentes (Ver imagen abajo).
- **Controlador:** La tarea del Controlador es muy amplia y depende de muchos factores pero vamos a resumirla. Un Controlador actua como el intermediario entre el back-end y el front-end, recogiendo el input del usuario (o la sonda) y realizando los cambios necesarios tanto en Modelo como en la Vista. Esta parte recibe toda la lógica del negocio, tanto la validación de las entradas como la transfor-

Desarrollo

mación de los datos. Este apartado esta realizado por otra parte del servidor, la cual esta dividida en 2 ramas principales que más adelante especificaremos (Ver imagen abajo).

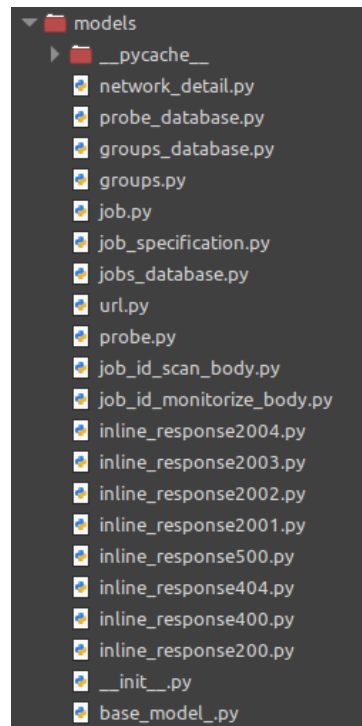


Figura 3.3: Estructura de carpetas para **Modelo**

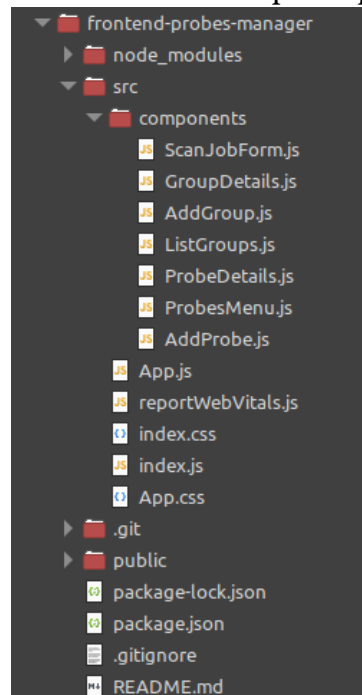


Figura 3.4: Estructura de carpetas para **Vista**

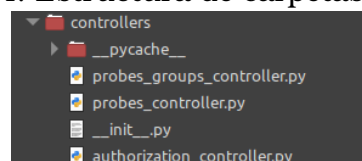


Figura 3.5: Estructura de carpetas para **Controlador**

3.4. Desarrollo de la API

3.4.1. Descripción general de la API

Como se ha mencionado anteriormente, la API se ha realizado utilizando Swagger/OpenAPI Specification 3.0 con el objetivo de documentar las diferentes posibilidades que la aplicación ofrece. Esto facilita el aprendizaje de la herramienta tanto para su uso como para construir sobre ella.

En nuestra API, existen dos caminos principales:

- **Probes:** Este camino se encarga de realizar todas las operaciones CRUD (Crear, Leer, Actualizar, Eliminar) con las sondas. Además, permite consultar el trabajo que en ese momento está realizando la sonda en cuestión.

probes Acceso al control de sondas		^
GET	/probes Lista y filtra entre todas las sondas	📋 ↩️ 🔗 ✓
POST	/probes Registra una nueva sonda	📋 ↩️ 🔗 ✓
GET	/probes/{Id} Devuelve la sonda seleccionada	📋 ↩️ 🔗 ✓
PUT	/probes/{Id} Actualizar una sonda existente	📋 ↩️ 🔗 ✓
DELETE	/probes/{Id} Elimina una sonda	📋 ↩️ 🔗 ✓
GET	/probes/{Id}/current-job Endpoint donde la sonda preguntará por su trabajo a hacer	📋 ↩️ 🔗 ✓

Figura 3.6: Path **Probes**

- **Probes-Groups:** Como su propio nombre indica, este camino se encarga de manejar no solo los grupos que puede crear el usuario con las sondas previamente registradas, sino también los trabajos asignados a estos grupos, consiguiendo una estrecha relación entre los trabajos y los grupos. En este endpoint se pueden utilizar las operaciones CRUD de la misma manera.

probes-groups Acceso al control de grupos		
GET	/probes-groups	Lista y filtra entre todos los grupos de sondas
POST	/probes-groups	Registra un nuevo grupo de sondas
GET	/probes-groups/{groupId}	Accede a la información del grupo de sondas
PUT	/probes-groups/{groupId}	Actualizar un grupo de sondas existente
DELETE	/probes-groups/{groupId}	Eliminar un grupo de sondas
GET	/probes-groups/{groupId}/jobs	Accede a todos los trabajos realizados por el grupo
POST	/probes-groups/{groupId}/jobs	Registra el trabajo al grupo
GET	/probes-groups/{groupId}/jobs/{jobId}	Accede a las especificaciones del trabajo
GET	/probes-groups/{groupId}/jobs/{jobId}/scan	Descarga el archivo procesado resultante de una operación scan
POST	/probes-groups/{groupId}/jobs/{jobId}/scan	Envía el fichero resultante de una operación scan
GET	/probes-groups/{groupId}/jobs/{jobId}/monitorize	Descarga el fichero resultante o el archivo procesado de una operación monitorize
POST	/probes-groups/{groupId}/jobs/{jobId}/monitorize	Envía el fichero resultante de una operación monitorize

Figura 3.7: Path **Probes-Groups**

3.4.2. Endpoints de la API

En esta sección se detallan los endpoints de la API, incluyendo sus rutas, métodos HTTP, y una breve descripción de su funcionalidad. En la imagen que se muestra con cada uno de ellos se pueden ver más en profundidad los atributos que utiliza y las respuestas que pueden dar.

3.4.2.1. Probes

- **GET /probes**

- **Descripción:** Lista y filtra entre todas las sondas. Los filtros disponibles

son por Nombre y por Estado

URI	http://localhost:8080/v1/probes/{probe_id}		
Metodo	GET		
Devuelve	404 Not found 200 OK JSON {Probe} 500 Internal Server Error		

Figura 3.8: Endpoint GET /probes

■ POST /probes

- **Descripción:** Registra una nueva sonda.

URI	http://localhost:8080/v1/probes		
Metodo	POST		
Cuerpo de la petición (solo PUT ó POST)	JSON {Probe}		
Devuelve	200 OK (/v1/probes/{id} + JSON) 500 Internal Server Error 400 Bad Request		

Figura 3.9: Endpoint POST /probes

■ GET /probes/Id

- **Descripción:** Devuelve la sonda seleccionada.

URI	http://localhost:8080/v1/probes		
Metodo	GET		
Cadena de consulta (SOLO GET)			
	Nombre	Aparecen resultados con el nombre correspondiente	
	is_active	Aparecen solo resultados activos	
Devuelve			
	404 Not found		
	200 OK JSON List[Probe]		
	500 Internal Server Error		

Figura 3.10: Endpoint GET /probes/Id

■ PUT /probes/Id

- **Descripción:** Actualizar una sonda existente.

URI	http://localhost:8080/v1/probes/{prob_id}		
Metodo	PUT		
Cuerpo de la petición (solo PUT ó POST)	JSON {Probe}		
Devuelve	404 Not found 200 OK (XML o JSON) 500 Internal Server Error 400 Bad Request		

Figura 3.11: Endpoint PUT /probes/Id

■ DELETE /probes/Id

- **Descripción:** Elimina una sonda.

URI	http://localhost:8080/v1/probes/{probe_id}		
Metodo	DELETE		
Devuelve	204 Content Deleted 500 Internal Server Error 400 Bad Request		

Figura 3.12: Endpoint DELETE /probes/Id

■ GET /probes/Id/current-job

- **Descripción:** Endpoint donde la sonda preguntará por su trabajo a hacer.

URI	http://localhost:8080/v1/probes/{probe_id}/current-job		
Metodo	GET		
Devuelve	404 Not found 200 OK JSON {Job} 500 Internal Server Error		

Figura 3.13: Endpoint GET /probes/Id/current-job

3.4.2.2. Probes Groups

■ GET /probes-groups

- **Descripción:** Lista y filtra entre todos los grupos de sondas.

Desarrollo

URI	http://localhost:8080/v1/probes_groups	
Metodo	GET	
Cadena de consulta (SOLO GET)	Nombre	Aparecen resultados con el nombre correspondiente
	is_active	Aparecen solo resultados activos
Devuelve	404 Not found 200 OK JSON List[probes_group] 500 Internal Server Error	

Figura 3.14: Endpoint GET /probes-groups

▪ POST /probes-groups

- **Descripción:** Registra un nuevo grupo de sondas.

URI	http://localhost:8080/v1/probes-groups	
Metodo	POST	
Cuerpo de la petición (solo PUT ó POST)	JSON {Probe-group}	
Devuelve	200 OK (/v1/probes-groups/{id} + JSON) 500 Internal Server Error 400 Bad Request	

Figura 3.15: Endpoint POST /probes-groups

▪ GET /probes-groups/groupId

- **Descripción:** Accede a la información del grupo de sondas.

URI	http://localhost:8080/v1/probes-groups/{group_id}	
Metodo	GET	
Devuelve	404 Not found 200 OK JSON {Group} 500 Internal Server Error	

Figura 3.16: Endpoint GET /probes-groups/groupId

▪ PUT /probes-groups/groupId

- **Descripción:** Actualizar un grupo de sondas existente.

URI	http://localhost:8080/v1/probes-groups/{group_id}		
Metodo	PUT		
Cuerpo de la petición (solo PUT ó POST)	JSON {group}		
Devuelve	404 Not found 200 OK (XML o JSON) 500 Internal Server Error 400 Bad Request		

Figura 3.17: Endpoint PUT /probes-groups/groupId

▪ DELETE /probes-groups/groupId

- **Descripción:** Eliminar un grupo de sondas.

URI	http://localhost:8080/v1/probes-groups/{group_id}		
Metodo	DELETE		
Devuelve	204 Content Deleted 500 Internal Server Error 400 Bad Request		

Figura 3.18: Endpoint DELETE /probes-groups/groupId

▪ GET /probes-groups/groupId/jobs

- **Descripción:** Accede a todos los trabajos realizados por el grupo.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs		
Metodo	GET		
Devuelve	404 Not found 200 OK JSON List[Job] 500 Internal Server Error		

Figura 3.19: Endpoint GET /probes-groups/groupId/jobs

▪ POST /probes-groups/groupId/jobs

- **Descripción:** Registra el trabajo al grupo.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs		
Metodo	POST		
Cuerpo de la petición (solo PUT ó POST)	JSON {Job}		
Devuelve	200 OK (/v1/probes-groups/{id}/jobs/{id} + JSON) 500 Internal Server Error 400 Bad Request		

Figura 3.20: Endpoint POST /probes-groups/groupId/jobs

▪ GET /probes-groups/groupId/jobs/jobId

- **Descripción:** Accede a las especificaciones del trabajo.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_id}		
Metodo	GET		
Devuelve	404 Not found 200 OK JSON {Job} 500 Internal Server Error		

Figura 3.21: Endpoint GET /probes-groups/groupId/jobs/jobId

▪ GET /probes-groups/groupId/jobs/jobId/scan

- **Descripción:** Descarga el archivo procesado resultante de una operación scan.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_id}/scan		
Metodo	GET		
Devuelve	404 Not found 200 OK JSON{application/octet-stream} 500 Internal Server Error		

Figura 3.22: Endpoint GET /probes-groups/groupId/jobs/jobId/scan

▪ POST /probes-groups/groupId/jobs/jobId/scan

- **Descripción:** Envía el fichero resultante de una operación scan.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_id}/scan	
Metodo	POST	
Cuerpo de la petición (solo PUT ó POST)	multipart/form-data{application/octet-stream(file) & application/json (probe_id_sender)}	
Devuelve	200 OK (/v1/probes-groups/{id}/jobs/{id} + JSON) 500 Internal Server Error 400 Bad Request	

Figura 3.23: Endpoint POST /probes-groups/groupId/jobs/jobId/scan

▪ GET /probes-groups/groupId/jobs/jobId/monitorize

- **Descripción:** Descarga el fichero resultante (application/octet) o el archivo procesado(application/json) de una operación monitorize.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_id}/monitorize	
Metodo	GET	
Devuelve	404 Not found 200 OK {application/octet-stream}/ JSON {networkSpecification} 500 Internal Server Error	

Figura 3.24: Endpoint GET /probes-groups/groupId/jobs/jobId/monitorize

▪ POST /probes-groups/groupId/jobs/jobId/monitorize

- **Descripción:** Envía el fichero resultante de una operación monitorize.

URI	http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_id}/monitorize	
Metodo	POST	
Cuerpo de la petición (solo PUT ó POST)	multipart/form-data{application/octet-stream(file) & application/json (probe_id_sender)}	
Devuelve	200 OK (/v1/probes-groups/{id}/jobs/{id} + JSON) 500 Internal Server Error 400 Bad Request	

Figura 3.25: Endpoint POST /probes-groups/groupId/jobs/jobId/monitorize

3.4.3. Componentes

Estos son los principales componentes que se estan utilizando, llamados por Nombre, en cualquier llamada de la API.

▪ Probe

```
probe ▾ {  
  id                ▾ integer  
  Name              ▾ string  
  LastRecordedIP    ▾ string  
  InUse             ▾ boolean  
  is_active         ▾ boolean  
  Group_id          ▾ integer  
}
```

Figura 3.26: Componente Probe

▪ Group

```
groups ▾ {  
  id                > [...]   
  Name              > [...]   
  probes_id        > [...]   
  is_active         > [...]   
}
```

Figura 3.27: Componente Group

▪ Job

Job ▾ {	
JobID	> [...]
BSSID	> [...]
ESSID	> [...]
Channel	> [...]
WaveLenght	> [...]
group_id	> [...]
Status	> [...]
created_at	> [...]
is_active	> [...]
band	> [...]
cswitch	> [...]
security	> [...]
wildcard	> [...]
associated_clients	> [...]
ChannelMode	> [...]
}	

Figura 3.28: Componente Job

■ NetworkDetail

NetworkDetail ▾ {	
BSSID*	> [...]
ESSID	> [...]
Security	> [...]
Channel	> [...]
}	

Figura 3.29: Componente NetworkDetail

3.5. Base de Datos

3.5.1. Estructura de la base de datos

La base de datos es una pieza fundamental de nuestro programa y está diseñada para soportar todas las operaciones que se requieran de la API, además de almacenar la información que luego servirá para poder formar nuestros modelos. A lo largo del proyecto, esta base de datos se ha ido actualizando hasta lo que conforma ahora. A continuación tenemos el esquema relacional que conforma nuestra aplicación, el cual comentaremos en el siguiente apartado:

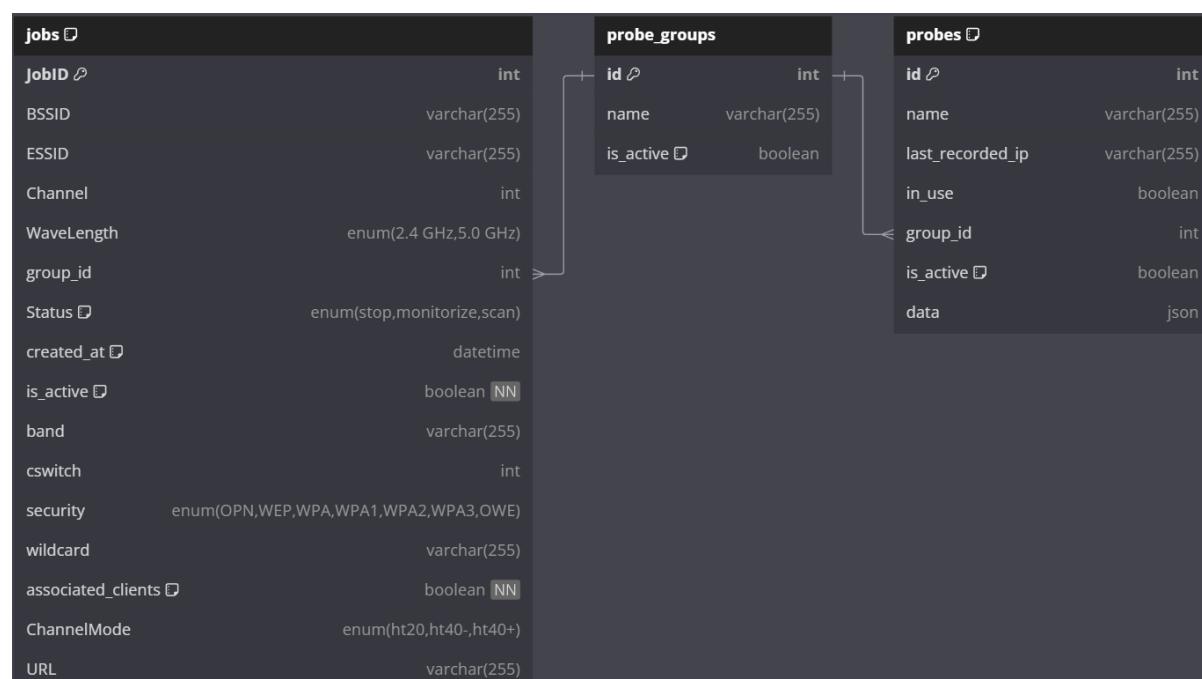


Figura 3.30: Base de datos

3.5.2. Tablas y Columnas

■ **Tabla: jobs**

Esta tabla se creo no solo para poder modelizar un trabajo, sino tambien con la intención de poder tener un registro histórico de todos los trabajos que se han realizado, pudiendo ordenar estos por grupo y/o fecha. Esta tabla tiene numerosas columnas relacionadas a la especificación del trabajo, tal y como se habló en el apartado 3.1.4, los cuales nos servirán para cubrir todos los casos posibles con el fin de explotar al máximo esta herramienta.

- **JobID:** Identificador único para cada trabajo.
- **BSSID:** Identificador MAC del punto de acceso (Basic Service Set Identifier).
- **ESSID:** Nombre de la red Wi-Fi (Extended Service Set Identifier).
- **Channel:** Canal de frecuencia en el que opera la red Wi-Fi.
- **WaveLength:** Longitud de onda de la red Wi-Fi (2.4 GHz o 5.0 GHz).
- **group_id:** Identificador del grupo al que pertenece el trabajo, referencia a **probe_groups.id**.
- **Status:** Estado del trabajo (stop, monitorize, scan).
- **created_at:** Fecha y hora en que se creó el trabajo.
- **is_active:** Indicador de si el trabajo está activo (1) o no (0).
- **band:** Banda de frecuencia de la red Wi-Fi.
- **cswitch:** Información relacionada con el cambio de canal.
- **security:** Tipo de seguridad de la red (OPN, WEP, WPA, WPA1, WPA2, WPA3, OWE).
- **wildcard:** Criterios adicionales o comodín para la red.
- **associated_clients:** Indicador de si hay clientes asociados (1) o no (0).
- **ChannelMode:** Modo del canal (ht20, ht40-, ht40+).
- **URL:** URL asociada con el trabajo.

■ **Tabla: probe_groups**

Esta tabla, pese a ser más pequeña es el núcleo de nuestro esquema. Esta es la tabla que relaciona y organiza las sondas con los trabajos, consiguiendo así armonía en el sistema. Las demás tablas referencian a esta.

- **id:** Identificador único para cada grupo de sondas.
- **name:** Nombre del grupo de sondas.
- **is_active:** Indicador de si el grupo está activo (1) o no (0).

■ **Tabla: probes** La tabla probes es la que nos deja mantener un control sobre las sondas, sus nombres y sus estados. La finalidad de esta tabla es la representación de una sonda real.

- **id:** Identificador único para cada sonda.

- **name:** Nombre de la sonda.
- **last_recorded_ip:** Última dirección IP registrada de la sonda.
- **in_use:** Indicador de si la sonda está en uso (1) o no (0).
- **group_id:** Identificador del grupo al que pertenece la sonda, referencia a **probe_groups.id**.
- **is_active:** Indicador de si la sonda está activa (1) o no (0).
- **data:** Redes guardadas después de una operación de monitorización, almacenados en formato JSON.

3.6. Servidor

Como ya se ha mencionado antes, el servidor está escrito en Flask, un microframework de python cuya principal funcionalidad es el desarrollo de APIs.

Para la construcción del servidor, nos hemos ayudado de la herramienta Codegen[16], desarrollada por Swagger. Esta herramienta nos construye un entorno de desarrollo a través de una especificación de una API, por eso se le ha dado mucha importancia a la esta. Desde Codegen podemos crear un Server Stub de muchos lenguajes, entre ellos Flask. El entorno es muy útil en algunos aspectos, aunque al ser relativamente reciente y algo bastante nuevo en el campo, tiene algunos fallos bastante claros. Una de las ventajas de usar el codegen es que realiza toda la parte de Modelos (MVC) y transporte/recepción de mensajes, haciendo que solo nos tengamos que centrar en la parte de Controladores. Al entrar el programa nos encontramos con esta estructura de carpetas:

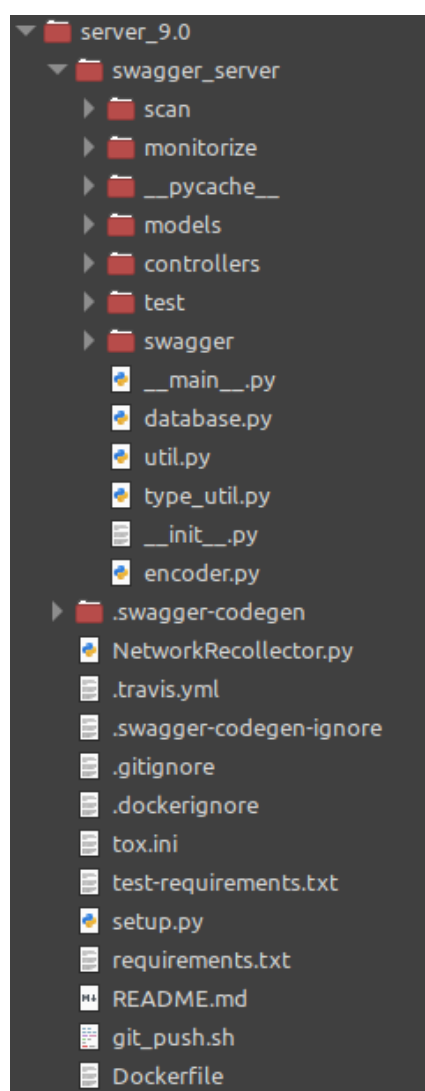


Figura 3.31: Carpeta Servidor

3.6.1. Implementación de métodos

Vamos a ir revisando los archivos más importantes en este sistema de carpetas. En esta explicación, varios de los archivos son generados automáticamente por Codegen. En primer lugar nos podemos encontrar el archivo `__main__.py` el cual se encarga de generar una instancia de nuestra API en el ordenador, específicamente en el puerto 8080. También se ha añadido para que cree una conexión con la base de datos y el Cross-Origin Resource Sharing para permitir a la aplicación manejar solicitudes de diferentes orígenes.

```
def main():
    app = connexion.App(__name__, specification_dir='./swagger/')
    app.json_encoder = encoder.JSONEncoder
    app.add_api('swagger.yaml', arguments={'title': 'Control de sondas'}, pythonic_params=True)

    # Configuración de SQLAlchemy
    app.app.config['SQLALCHEMY_DATABASE_URI'] = 'mysql+pymysql://[usuario]:[contraseña]@[nombre_de_base_de_datos]'
    # Reemplaza 'usuario', 'contraseña' y 'nombre_de_base_de_datos' con tus propios valores
    app.app.config['SQLALCHEMY_TRACK_MODIFICATIONS'] = False
    db.init_app(app.app)

    CORS(app.app)

    # Esto te permite acceder a la instancia de SQLAlchemy en tu aplicación utilizando 'app.app.db'

    app.run(port=8080)

if __name__ == '__main__':
    main()
```

Figura 3.32: Clase Main

Existe también `database.py` encargada de la inicialización de la base de datos, `encoder.py` que se encarga de la serialización a JSON de los objetos recibidos y el archivo `util.py` el cual es el encargado de deserializar todo tipo de información, como objetos, modelos o diccionarios.

Por otro lado, podemos visualizar `NetworkRecollector.py`, archivo creado por mí y que trata de, al recibir una captura de redes creada por airodump-ng como las anteriores vistas en 3.1.4, generar una estructura de datos en JSON por salida estándar, de forma que mi servidor pueda llamar a esta función y recoger la salida para poder usarla en la lógica.

```
def extract_first_table(input_file):
    with open(input_file, 'r') as file:
        lines = file.readlines()

    if lines[0].strip() == '':
        lines = lines[1:]

    table_end_index = None
    for index, line in enumerate(lines):
        if line.strip() == '':
            table_end_index = index
            break

    if table_end_index is None:
        table_end_index = len(lines)

    first_table_lines = lines[:table_end_index]

    if not first_table_lines or len(first_table_lines) < 2:
        return pd.DataFrame()

    header = first_table_lines[0].strip().split(',')
    data = [line.strip().split(',') for line in first_table_lines[1:] if line.strip()]
    data = [row if len(row) == len(header) else row[:len(header)] for row in data]

    df_first_table = pd.DataFrame(data, columns=header)

    rename_columns = {'BSSID': 'bssid', 'channel': 'channel', 'Privacy': 'security', 'ESSID': 'ssid'}
    df_first_table.rename(columns=rename_columns, inplace=True)

    columns_of_interest = ['bssid', 'channel', 'security', 'ssid']
    return df_first_table[columns_of_interest]

Codiumate: Options | Test this function
def convert_to_json(df):
    networks = df.to_dict(orient='records')
    data = {'networks': networks}
    print(json.dumps(data, indent=4))

if __name__ == '__main__':
    if len(sys.argv) != 2:
        print("Usage: python script_name.py <input_file_path>")
        sys.exit(1)

    input_file_path = sys.argv[1]

    df_first_table = extract_first_table(input_file_path)

    if df_first_table.empty:
        print("No data found in the input file.")
    else:
        convert_to_json(df_first_table)
```

Figura 3.33: CaptureNetworks

Existen varias carpetas, entre ellas:

- **Models:** Carpeta que vamos a comentar más adelante, forma una parte fundamental en el MVC. Ver la foto 3.5
- **Controllers:** Igual que la anterior
- **scan:** Carpeta donde se guardan los diferentes escaneos hechos por los grupos. Estos archivos pueden ser de diferentes tipos pero por ahora solo se contempla .csv Se guardan para su futuro análisis
- **monitorize:** Carpeta donde se guardan las diferentes monitorizaciones hechas por los grupos. Estos archivos pueden ser de diferentes tipos pero por ahora solo se contempla .csv Se utilizan junto a 3.33 para poder sacar la información de las redes disponibles
- **test:** Carpeta para todo el testing de los endpoints

Vamos a centrarnos en las 2 carpetas más importantes, Models y Controllers

3.6.1.1. Models

Models 3.5 es una carpeta compuesta por todos los modelos que podemos encontrar en el programa, en particular en el apartado de “components” de la especificación de la API. Estos puede estar entre respuestas *inline_response200.py* la cual significa una respuesta del servidor OK 200 con un mensaje personalizado hasta *probe.py* el cual define un objeto tipo sonda.

```
def __init__(self, id: int=None, name: str=None, last_recorded_ip: str=None, in_use: bool=None, is_active: bool=None, group_id: int=None): # noqa: E501
    """Probe - a model defined in Swagger"""
    :param id: The id of this Probe. # noqa: E501
    :type id: int
    :param name: The name of this Probe. # noqa: E501
    :type name: str
    :param last_recorded_ip: The last_recorded_ip of this Probe. # noqa: E501
    :type last_recorded_ip: str
    :param in_use: The in_use of this Probe. # noqa: E501
    :type in_use: bool
    :param is_active: The is_active of this Probe. # noqa: E501
    :type is_active: bool
    :param group_id: The group_id of this Probe. # noqa: E501
    :type group_id: int
    """
    self.swagger_types = {
        'id': int,
        'name': str,
        'last_recorded_ip': str,
        'in_use': bool,
        'is_active': bool,
        'group_id': int
    }
    self.attribute_map = {
        'id': 'id',
        'name': 'Name',
        'last_recorded_ip': 'LastRecordedIP',
        'in_use': 'inuse',
        'is_active': 'is_active',
        'group_id': 'Group_id'
    }
    self._id = id
    self._name = name
    self._last_recorded_ip = last_recorded_ip
    self._in_use = in_use
    self._is_active = is_active
    self._group_id = group_id
```

Figura 3.34: Constructor de una sonda

Esta carpeta viene pregenerada por el Codegen de Swagger, pero aunque advierte en contra de la modificación de los modelos, nosotros hemos tenido que realizar algunos ajustes para que la integración de la base de datos funcionara. Se puede apreciar en los archivos como existen varios de ellos son llamados *xxx_database.py* Estos archivos los hemos tenido que crear nosotros y representan el objeto directamente sacado de la base de datos. Con SQLAlchemy, existen un tipo de clases con la peculiaridad de actuar como ORM(Object Related Mapping), los cuales unen un modelo de código con uno en la base de datos, con el fin de construir objetos desde una base de datos de forma sencilla. Este es un ejemplo de la misma clase probes pero en objeto ORM:

```
class ProbeDataBase(db.Model):
    __tablename__ = 'probes'
    id = db.Column(db.Integer, primary_key=True, autoincrement=True)
    name = db.Column(db.String(255))
    last_recorded_ip = db.Column(db.String(255))
    in_use = db.Column(db.Boolean)
    group_id = db.Column(db.Integer, db.ForeignKey('probe_groups.id'))
    is_active = db.Column(db.Boolean, default=True) # Campo adicional para manejar el estado activo/inactivo
    data = db.Column(db.JSON) # Campo JSON para almacenar datos adicionales
```

Figura 3.35: Constructor ORM de una sonda

Ahora bien, al utilizar el controlador, es importante devolver objetos que sean de

```

@classmethod
def from_db(cls, probe_db):
    """
    Método para construir una instancia de sonda desde la base de datos,
    con opción para incluir sondas inactivas.

    :param probe_db: Instancia de la base de datos de la sonda.
    :type probe_db: <Database Model Instance>
    :param include_inactive: Si se incluyen sondas inactivas en la instancia creada.
    :type include_inactive: bool
    :return: Una instancia del modelo Probe.
    :rtype: Probe
    """

    return cls(
        id=probe_db.id,
        name=probe_db.name,
        last_recorded_ip=probe_db.last_recorded_ip,
        in_use=probe_db.in_use,
        is_active=probe_db.is_active,
        group_id=probe_db.group_id
    )

```

Figura 3.36: Función para poder pasar de un objeto sonda ORM a un objeto sonda serializable

los creados por el Codegen, con el fin de mantener las capacidades de serialización intactas, por eso para cada uno de las clases, *probe.py*, *job.py* y *groups.py* tiene su conversor de una clase ORM a un objeto automáticamente generado.

Aunque poco común, esta solución fue la única efectiva a la hora de hacer la integración en la base de datos. Otras opciones planteadas generaban múltiples errores y necesitaban quitar funcionalidades a la clase porque hacía conflicto con los métodos de esta y, al nosotros necesitar obligatoriamente la librería de SQLAlchemy para poder mantener la eficiencia en las búsquedas de la base de datos, así como la robustez ante SQL injections, decidimos proseguir por esta ruta.

Por último la clase *base_model.py* existe como una abstracción de los modelos que existen, permitiendo operaciones entre las clases como, serialización, deserialización y comparación con otros objetos. Además esta clase garantiza consistencia en el manejo y representación de los datos

```
@classmethod
def from_dict(cls: typing.Type[T], dikt) -> T:
    """Returns the dict as a model"""
    return util.deserialize_model(dikt, cls)

Codiumate: Options | Test this method
def to_dict(self):
    """Returns the model properties as a dict

    :rtype: dict
    """
    result = {}

    for attr, _ in six.iteritems(self.swagger_types):
        value = getattr(self, attr)
        if isinstance(value, list):
            result[attr] = list(map(
                lambda x: x.to_dict() if hasattr(x, "to_dict") else x,
                value
            ))
        elif hasattr(value, "to_dict"):
            result[attr] = value.to_dict()
        elif isinstance(value, dict):
            result[attr] = dict(map(
                lambda item: (item[0], item[1].to_dict())
                if hasattr(item[1], "to_dict") else item,
                value.items()
            ))
        else:
            result[attr] = value

    return result

Codiumate: Options | Test this method
def to_str(self):
    """Returns the string representation of the model

    :rtype: str
    """
    return pprint.pformat(self.to_dict())

Codiumate: Options | Test this method
def __repr__(self):
    """For `print` and `pprint`"""
    return self.to_str()

Codiumate: Options | Test this method
def __eq__(self, other):
    """Returns true if both objects are equal"""
    return self.__dict__ == other.__dict__

Codiumate: Options | Test this method
def __ne__(self, other):
    """Returns true if both objects are not equal"""
    return not self == other
```

Figura 3.37: Clase base_model.py

3.6.1.2. Controllers

La clase controllers 3.5 es la más trabajada de todas ya que contiene toda la lógica detrás de la aplicación, ya además de ser el intermediario entre la vista y los modelos, es el responsable de la validación de datos y la gestión del flujo. Existen 2 archivos, uno para cada rama de nuestra API:

- **Probes_controller.py** La clase más sencilla de estas dos, existen 6 métodos en esta clase, uno por cada endpoint existente en la especificación de la API. Esta clase tiene la misión de poder realizar cualquier operación con un grupo de sondas y sus trabajos asociados. Cada uno de estos métodos realiza las comprobaciones necesarias antes de procesar la información. En este caso vamos a ver un unico método, *get_prob_job.py* el cual es el más ilustrativo. Este se relaciona con el endpoint 3.4.2.1.

```
def get_probe_job(id_): # noqa: E501
    """Endpoint donde la sonda preguntará por su trabajo a hacer

    Devuelve el trabajo realizado por la sonda # noqa: E501

    :param id: ID de la sonda a consultar
    :type id: int

    :rtype: JobSpecification
    """
    probe = db.session.query(ProbeDataBase).filter_by(id=id_, is_active=True).one_or_none()
    if not probe:
        return {"error": "Probe not found or is inactive"}, 404

    if not probe.group_id:
        return {"error": "Probe is not assigned to any group"}, 400

    group = db.session.query(GroupDataBase).filter_by(id=probe.group_id, is_active=True).one_or_none()
    if not group:
        return {"error": "Group not found or is inactive"}, 404

    job = db.session.query(JobDataBase)\
        .filter_by(group_id=group.id, is_active=True)\
        .order_by(JobDataBase.created_at.desc())\
        .first()
    if not job:
        return {"error": "No active job found for the group"}, 404

    job_spec = JobSpecification.from_db(job)
    return job_spec
```

Figura 3.38: Programa Current Job

Como se puede apreciar, el código pasa por 3 fases

1. **Comprobación y obtención:** Esta fase a menudo se pasa por alto. Aquí, aplicamos varios filtros mientras intentamos obtener la información necesaria para realizar la tarea. Durante este proceso, verificamos la existencia de los datos y, si es necesario, devolvemos el mensaje de error correspondiente junto con su código de error. Además, comprobamos si el elemento tiene el flag de ".Activada". En nuestra aplicación, utilizamos un "soft delete" para mantener un buen historial, permitiendo así desactivar elementos sin eliminarlos completamente.
2. **Uso y queries:** Con la información ya correcta, hacemos uso de esta para obtener la información que necesitamos, abdicando siempre por la legibilidad y el uso de las funciones de SQLAlchemy para poder hacer queries de forma efectiva.

3. **Creación del objeto a devolver:** Por el problema descrito en 3.6.1.1, tenemos que utilizar la función *from_db.py* para la transformación en un objeto serializable. Si se devuelve el objeto, el código siempre será 200 OK en este ejemplo.

■ **Probes_groups_controller.py**

Con un total de más de 450 líneas de código, este controlador se encarga de manejar los grupos de sondas y los trabajos de la aplicación. La clase de controladores es crucial para la lógica de la aplicación, ya que gestiona la creación, actualización y eliminación de grupos de sondas, así como la asignación y supervisión de trabajos realizados por estos grupos. Además, maneja la carga y descarga de archivos relacionados con las operaciones de monitoreo y escaneo realizadas por las sondas. Vamos a explicar 3 de sus funciones más significativas:

- **create_job:** Esta función analiza la adición de un nuevo trabajo a un grupo y respeta los pasos vistos anteriormente. Para poder crear el trabajo, la función comprueba que tipo de trabajo hay que crear para poder realizar una query a la base de datos correcta con la información necesaria.

```

def create_job(group_id, body=None): # noqa: E501
    """Registra el trabajo al grupo

    Registra el trabajo al grupo # noqa: E501

    :param group_id: ID del grupo a consultar
    :type group_id: int
    :param body:
    :type body: dict | bytes

    :rtype: InlineResponse2002
    """
    group = Groups.query.filter_by(id=group_id).first()
    if not group:
        return InlineResponse404(message="Group not found"), 404
    if connexion.request.is_json:
        body = JobSpecification.from_dict(connexion.request.get_json())

    # Marcar todos los trabajos anteriores activos como inactivos
    previous_jobs = JobDataBase.query.filter(
        and_(JobDataBase.group_id == group_id, JobDataBase.is_active == True)
    ).all()
    for job in previous_jobs:
        job.is_active = False

    # Crear el nuevo trabajo
    job_db = JobDataBase(
        group_id=group_id,
        Status=body.job.status,
        is_active=True # Asume que los nuevos trabajos son activos por defecto
    )

    # Si el trabajo es "scan" asignamos los detalles adicionales
    if body.job.status == "scan":
        job_db.BSSID = body.job.bssid
        job_db.ESSID = body.job.essid
        job_db.Channel = body.job.channel
        job_db.WaveLength = body.job.wave_lenght
        job_db.Status = body.job.status
        job_db.band = body.job.band
        job_db.cswitch = body.job.cswitch
        job_db.security = body.job.security
        job_db.wildcard = body.job.wildcard
        job_db.associated_clients = body.job.associated_clients
        job_db.ChannelMode = body.job.channel_mode

    db.session.add(job_db)
    db.session.flush() # Flush to get the JobID before committing

    # Asignar URL si es necesario y no es un trabajo 'stop'
    if body.job.status != "stop":
        job_db.URL = f"http://localhost:8080/v1/probes-groups/{group_id}/jobs/{job_db.JobID}/{job_db.Status}"

    db.session.commit()
    return InlineResponse2002(job_id=job_db.JobID) # Asume que InlineResponse2002 puede manejar el job_id

```

Figura 3.39: Función create_job

Esta función contempla varios casos de trabajo, tanto “stop”, “scan” y “monitorize” y dependiendo de que acción esta tomando, utiliza diferentes atributos para copiar, así con el fin de mantener los diferentes tipos de trabajos conforme a una norma. Para finalizar, hace la operación *flush()* para poder obtener el id asignado y así devolverlo al usuario.

- **delete_group:** Esta función como su nombre indica, borra un grupo.


```
def delete_group(group_id): # noqa: E501
    """Eliminar un grupo de sondas

    Eliminar un grupo de sondas dado el ID # noqa: E501

    :param group_id: ID del grupo de sondas a eliminar
    :type group_id: int

    :rtype: None
    """
    group = db.session.query(GroupDataBase).filter_by(id=group_id).one_or_none()
    if not group:
        return {"error": "Group not found"}, 404

    # Desasignar todas las sondas asociadas con este grupo
    probes_in_group = db.session.query(ProbeDataBase).filter_by(group_id=group_id).all()
    for probe in probes_in_group:
        probe.group_id = None

    group.is_active = False # Desactivar el grupo
    db.session.commit()
    return {"message": "Group deactivated and all assigned probes unassigned"}, 200
```

Figura 3.40: Función delete_group()

Esta función elimina un grupo pero con nuestra política de “soft delete” en vez de borrar, dejamos el grupo desactivado. También es importante lidiar con la redundancia en el código de forma responsable y eliminar el grupo de las sondas que lo conformaban sin eliminar a estas mismas, con el fin de mantener la integridad de la base de datos.

- **send_work_monitorize:** Esta función es la encargada de recoger los datos de monitorización del usuario, con el fin de que este elija la red de todas las disponibles para poder escanear y así, generar un archivo estudiable.

```

def send_work_monitorize(body, group_id, job_id): # noqa: E501
    """
    Envía el fichero resultante de una operación monitorize

    Envía el fichero resultante de una operación monitorize # noqa: E501

    :param body: The body of the request containing the file and probe ID.
    :type body: dict
    :param group_id: ID del grupo a consultar
    :type group_id: int
    :param job_id: ID del trabajo a gestionar
    :type job_id: int

    :rtype: InlineResponse2003
    """

    if 'file' not in connexion.request.files or 'probeId' not in connexion.request.form:
        return {"mensaje": "Falta el archivo o el ID de la sonda"}, 400

    # Obtener el archivo de la solicitud
    file = connexion.request.files['file']
    if file.filename == '':
        return {"mensaje": "No se ha seleccionado ningún archivo"}, 400

    # Intentar obtener el probeId, asegurándose de que se pueda convertir a entero
    try:
        probe_id = int(connexion.request.form['probeId'])
    except ValueError:
        return {"mensaje": "El ID de la sonda debe ser un número entero"}, 400

    # Verificar que el grupo y el trabajo existen y están activos
    grupo = GroupDataBase.query.filter_by(id=group_id, is_active=True).first()
    if not grupo:
        return {"error": "Group not found or is inactive"}, 404

    trabajo = JobDataBase.query.filter_by(JobID=job_id, group_id=group_id).first()
    if not trabajo:
        return {"error": "Job not found or is not associated with this group"}, 404

    # Ruta base del código
    base_path = Path(__file__).resolve().parent.parent
    upload_folder = base_path / 'monitorize' / f'Group{group_id}Job{job_id}'
    upload_folder.mkdir(parents=True, exist_ok=True)

    # Crear nombre de archivo seguro
    filename = secure_filename(f"monitorize_{group_id}_{job_id}_{probe_id}.csv") # Asumiendo que el archivo es un PDF
    filepath = upload_folder / filename

    # Guardar archivo
    file.save(filepath)

    # Utilizamos este archivo para poder sacar los json
    base_dir = os.path.dirname(__file__)
    nombre_script = os.path.join(base_dir, '..', '..', 'NetworkRecollector.py')
    result = subprocess.run(['python3', nombre_script, str(filepath)], capture_output=True, text=True)

    if result.returncode != 0:
        return {"error": "Failed to process file"}, 500
    logger.debug(f"{result.stdout}")
    data = json.loads(result.stdout)
    probe = ProbeDataBase.query.get(probe_id)
    probe.data = data

    db.session.commit()
    return InlineResponse2003(mensaje="File uploaded successfully")

```

Figura 3.41: Función send_work_monitorize()

En resumen, cuando se hace una petición de envío de monitorización, de la petición se obtienen 2 valores, el archivo csv enviado y el id de la sonda que lo envía. El id es necesario ya que cada sonda manda su propia monitorización y necesitamos saber que sondas están detectando que redes para poder dar una aproximación mas cercana. Después confirmamos que el grupo y el trabajo existe, para a continuación guardar el archivo en una carpeta llamada monitorize y se le asigna un nombre identificativo. Una vez en el equipo, se envía la ruta al programa “NetworkRecollector.py” (ver 3.33) que nos enviará a esta función los datos procesados del archivo. Después de confirmar el código de salida del programa, se procede a guardar el JSON resultante en la base de datos donde podrá ser accedido facilmente.

3.7. Desarrollo del front-end

Como se ha comentado anteriormente, el frontend conforma la parte de la Vista en el modelo MVC 3.3.2, y por tanto es la manera en la que el usuario puede interactuar con el sistema sin tener un conocimiento profundo del mismo, consiguiendo así una mayor adaptabilidad a la herramienta. Se ha utilizado React, como se mencionó en el estado del arte. El frontend de esta versión se ha diseñado para ser minimalista y usable, sin añadir demasiados detalles a los diferentes componentes que se describirán en esta sección.

3.7.1. Componentes

Existen en total 9 componentes o pantallas que forman el front-end, cada uno con una función diferente. Estas pantallas, cubren en total todos los requisitos expuestos anteriormente, asegurando una experiencia de usuario completa. A continuación se exponen todos ellos:

- **Menú principal:** Componente más sencillo del programa, este solo tiene el título del componente junto a los 2 caminos principales, administrar sondas y administrar grupos, disponibles a través de los botones azules en pantalla.

Menú Principal

Probes

Probes Groups

Figura 3.42: Componente Menú principal

- **Gestión de sondas:** En la parte superior, hay una barra de búsqueda que permite filtrar las sondas por nombre, facilitando la localización de una sonda específica entre muchas. Debajo de esta barra, hay una casilla de verificación que permite al usuario optar por mostrar solo las sondas activas. El botón verde “Añadir Sonda” sugiere que el usuario puede registrar nuevas sondas. Debajo del botón, se muestra una lista de sondas ya registradas, cada una con un ID único y un nombre asociado. Los enlaces de cada sonda probablemente llevan a una página con más detalles sobre la sonda seleccionada.

Gestión de Sondas

☐ Mostrar solo activasAñadir Sonda

[26 - Hola1](#)

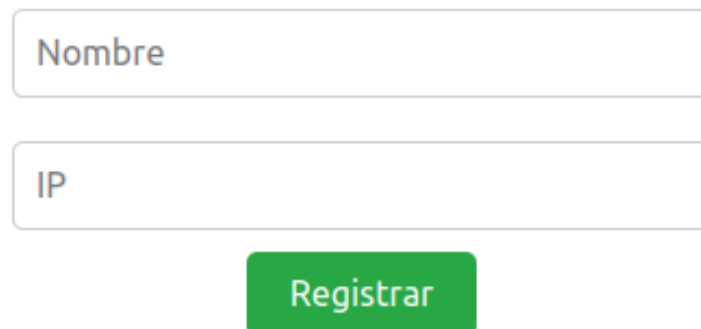
[27 - Prueba](#)

Figura 3.43: Componente Gestión de sondas

- **Registrar Nueva Sonda:** Este formulario solicita dos campos esenciales: “Nombre” y “IP”. El campo “Nombre” permite al usuario asignar una identificación amigable a la sonda, mientras que el campo “IP” requiere la dirección IP de la sonda. Una vez que se han completado estos campos, el usuario puede hacer clic en el botón verde “Registrar” para añadir la nueva sonda al sistema. Este proceso es fundamental para mantener un inventario actualizado de las sondas que se están utilizando.

Nota: Esta función del front-end será eliminada en el despliegue final de la aplicación ya que debería hacerse automáticamente desde las sondas.

Registrar Nueva Sonda



Formulario de registro de nueva sonda. Incluye dos campos de entrada: "Nombre" y "IP", y un botón verde "Registrar".

Nombre

IP

Registrar

Figura 3.44: Proceso de registro de sonda

- **Detalles de la Sonda:** La página muestra detalles de una sonda específica, en este caso, una sonda llamada “Hola1” con el ID 26. Se presenta información detallada como la última dirección IP registrada (1.3.1.1), el estado de uso (Sí), y varios parámetros técnicos relevantes para su funcionamiento actual, como el status (scan), grupo, BSSID, ESSID, canal, longitud de onda (2.4 GHz), y la fecha y hora de creación. Esta página también ofrece botones para actualizar la información de la sonda, eliminarla del sistema, o simplemente cerrar la vista de detalles. Esta funcionalidad es crucial para la gestión y monitoreo continuo de cada sonda en el sistema.

Detalles de la Sonda

ID: 26

Nombre:

Hola1

Última IP registrada:

1.3.1.1

En uso: Sí

Trabajando actualmente en:

Status: scan

Grupo: 16

BSSID: 98:97:D1:33:6A:1C

ESSID: MOVISTAR_6A1B

Channel: 6

WaveLenght: 2.4 GHz

Created At: 6/15/2024, 11:18:23 PM

Actualizar

Eliminar

Cerrar

Figura 3.45: Detalles de una sonda

- **Lista de Grupos de Sondas:** Esta pantalla muestra una lista de grupos de sondas existentes en el sistema. En la parte superior, hay un botón verde con la etiqueta *Añadir Grupo*, que permite al usuario agregar un nuevo grupo de sondas. Debajo del botón, se enumeran los grupos de sondas ya creados, cada uno con un identificador único y su nombre correspondiente. Esta interfaz proporciona una vista clara y organizada de los grupos de sondas disponibles.



Figura 3.46: Gestión de grupos

- **Registrar Nuevo Grupo:** Esta pantalla permite al usuario registrar un nuevo grupo de sondas. En la parte superior, hay un campo de entrada de texto donde se puede introducir el nombre del nuevo grupo. Debajo del campo de texto, se presentan opciones para seleccionar las sondas que se incluirán en el grupo, cada una con una casilla de verificación. Un botón verde etiquetado *Registrar Grupo* permite confirmar y registrar el nuevo grupo con las sondas seleccionadas.

Registrar Nuevo Grupo

Seleccionar Sondas:

- ☐ Hola1
☐ Prueba

Figura 3.47: Proceso de registro de un grupo

- **Detalles del Grupo:** Esta pantalla muestra los detalles de un grupo específico de sondas. En la parte superior, se muestra el identificador y el nombre del grupo, seguido por su estado (Activo). Debajo, se listan las sondas en el grupo, con la opción de ver detalles para volver al menú de detalles de la sonda. La sección de trabajos realizados muestra información detallada sobre los trabajos asociados al grupo, incluyendo el ID del trabajo, BSSID, ESSID, canal, longitud de onda, estado y la fecha de creación. Esta sección tiene botones cuya función es navegar en orden de ejecución los trabajos realizados. Existe la opción de descargar trabajo, si está disponible. También hay botones para parar el trabajo y asignar un nuevo trabajo.

Detalles del Grupo

16 - Grupo1
Estado: Activo
Sondas en el Grupo:
26 - Hola1 [Ver Detalles](#)

Trabajos Realizados:

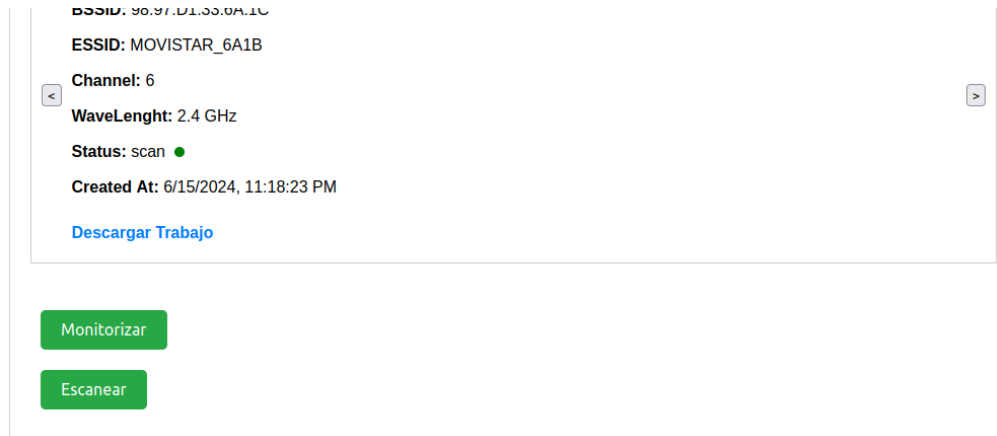
ID: 30
BSSID: 98:97:D1:33:6A:1C
ESSID: MOVISTAR_6A1B
Channel: 6
WaveLenght: 2.4 GHz
Status: scan ●
Created At: 6/15/2024, 11:18:23 PM
[Descargar Trabajo](#)

[Parar Trabajo](#)
[Asignar Nuevo Trabajo](#)

Figura 3.48: Detalles de un grupo

Desarrollo

Cuando se quiere asignar un nuevo trabajo, al presionar el botón “Asignar Nuevo Trabajo” se desplegarán 2 opciones que distinguen los dos tipos de trabajo disponibles, Monitorizar y Escanear.



The screenshot displays a form for assigning a new task. It includes the following information:

- BSSID:** 98:97:D1:55:0A:1C
- ESSID:** MOVISTAR_6A1B
- Channel:** 6
- WaveLenght:** 2.4 GHz
- Status:** scan (indicated by a green dot)
- Created At:** 6/15/2024, 11:18:23 PM
- [Descargar Trabajo](#)

Below the form, there are two green buttons: "Monitorizar" and "Escanear".

Figura 3.49: Asignar un nuevo trabajo

- **Especificar un nuevo trabajo:** Al entrar en el apartado de escanear, una nueva pantalla estilo formulario nos pedirá todos los diferentes filtros que podemos introducir al Airodump. Este formulario es inteligente, lo que significa que intentará descargar los datos de monitorización del grupo para poder discernir las diferentes redes que tienen a su alcance. Por otro lado, una vez se seleccione un parámetro, los demás dependientes de este se ajustarán para solo enseñar opciones compatibles, haciendo la experiencia más sencilla y pudiendo siempre realizar escaneos satisfactorios. Si no se puede obtener esta información de monitorización, se podrá seguir filtrando por los demás filtros disponibles.

Assign Scan Job

BSSID:

All BSSIDs ▼

ESSID:

All ESSIDs ▼

Security:

All Security Types ▼

Channel:

All Channels ▼

WaveLength:

Select WaveLength ▼

Channel Mode:

Select Channel Mode ▼

Channel Switch:

Select Switch Method ▼

Netmask:

Band:

☐ A (5 GHz)

☐ B (2.4 GHz)

☐ G (2.4 GHz)

Show Associated Clients: ☐

Submit

Figura 3.50: Proceso para realizar un nuevo escaneo

Capítulo 4

Pruebas de ejecución

Las pruebas de software son esenciales en el desarrollo de una herramienta como la que se ha presentado en este trabajo. En este capítulo nos vamos a centrar no solo en evaluar el programa para poder conocer su eficacia y robustez sino también analizaremos si se han cumplido los requisitos establecidos en apartados anteriores. Además, nos enfocaremos en mostrar el funcionamiento interno del programa de manera detallada. Esto implica explicar los procesos subyacentes, las llamadas a funciones específicas y cómo interactúan entre sí para lograr los resultados esperados. Asimismo, destacaremos las pruebas específicas realizadas para verificar cada componente del software y garantizar su correcto desempeño en diferentes escenarios y condiciones.

Para llevar a cabo las pruebas, diseñaremos varias pruebas de extremo a extremo que simularán las diferentes acciones que un usuario puede ir tomando además de detallar cada paso necesario para completarla. De tal manera se enseñará la aplicación en funcionamiento a la par para su comprensión.


4.1. Definición de pruebas

Prueba	Descripción y Resultado Esperado
Prueba 1: Añadir sondas	Descripción: Añadir varias sondas para iniciar el proceso de escaneo. Resultado esperado: Las sondas se añaden correctamente sin errores.
Prueba 2: Formar grupos	Descripción: Agrupar las sondas después de añadirlas. Resultado esperado: Las sondas se agrupan correctamente.
Prueba 3: Escaneo del entorno	Descripción: Permitir que las sondas escaneen su entorno. Resultado esperado: Las sondas proporcionan datos precisos del entorno.
Prueba 4: Realizar escaneo efectivo	Descripción: Utilizar los datos obtenidos para realizar un escaneo efectivo. Resultado esperado: El escaneo es preciso y útil para los propósitos establecidos.
Prueba 5: Eliminar grupo	Descripción: Eliminar un grupo de sondas previamente creado. Resultado esperado: El grupo se elimina haciendo "soft delete" y sin errores.
Prueba 6: Editar sonda	Descripción: Editar las propiedades de una sonda existente. Resultado esperado: Las propiedades de la sonda se actualizan correctamente sin errores.

Estas son las pruebas que se han decidido realizar. Pese a su simpleza, muchas de ellas son dependientes de sus anteriores para poder funcionar, por tanto se está testeando no solo la funcionalidad, sino también la armonía que tiene el sistema implementado.

4.2. Realización de pruebas

- **Prueba 1: Añadir sondas** Para esta prueba, primero navegaremos a la pantalla de listar sondas y accederemos al boton de “Registrar nueva sonda”*. Aclaración: Esta tarea es realizada por la sonda de forma automática al iniciarse. Con tal de poder replicar el funcionamiento, se permite añadir una nueva sonda pero esto puede cambiar en el futuro. Aqui tenemos la base de datos antes de la ejecución:



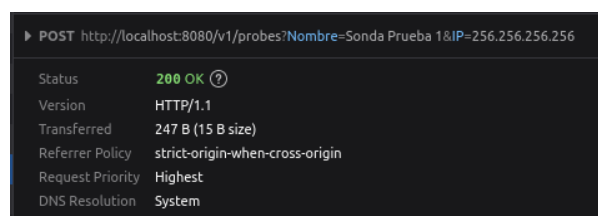
#	id	name	last_recorded_ip	in_use	group_id	is_active	data
1	26	Hola1	1.3.1.1	1	16	1	NULL
2	27	Prueba	2.2.2.2	0	17	1	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Introducimos los datos de la sonda a asignar y hacemos la llamada

Registrar Nueva Sonda

Sonda registrada con ID: 30

La llamada se realiza de manera satisfactoria, la respuesta 200 OK nos confirma el hecho. Se devuelve el ID de la sonda



▶ POST http://localhost:8080/v1/probes?Nombre=Sonda Prueba 1&IP=256.256.256.256	
Status	200 OK ?
Version	HTTP/1.1
Transferred	247 B (15 B size)
Referrer Policy	strict-origin-when-cross-origin
Request Priority	Highest
DNS Resolution	System

Ahora si queremos filtrar en la lista de sondas por el nombre que le hemos dado, nos aparecerá

Gestión de Sondas

☐ Mostrar solo activasAñadir Sonda[30 - Sonda Prueba 1](#)

- **Prueba 2: Formar Grupos** Para esta prueba utilizaremos 2 sondas, la creada en el apartado anterior y “aa”, una sonda creada aparte.

Registrar Nuevo Grupo

Seleccionar Sondas:

- ☐ Hola1
- ☐ Prueba
- ☐ SondaPrueba1
- ☒ aa
- ☒ Sonda Prueba 1

Registrar Grupo

Grupo creado con éxito: 18

Pruebas de ejecución

Como podemos observar, la respuesta es un 200 OK lo cual indica que la operación ha sido satisfactoria, devolviendo el ID 18

▶ POST http://localhost:8080/v1/probes-groups?Nombre=GrupoPrueba2	
Status	200 OK ?
Version	HTTP/1.1
Transferred	247 B (15 B size)
Referrer Policy	strict-origin-when-cross-origin
Request Priority	Highest
DNS Resolution	System

Este es el body de la respuesta donde va la información de las sondas que hemos añadido, cumpliendo así con nuestro cometido

▶	Headers	Cookies	Request	Response	Time
Filter Request Parameters					
JSON					
▼ 0: {...}					
id: 29					
InUse: false					
is_active: true					
LastRecordedIP: "1.1.1.1"					
Name: "aa"					
▼ 1: {...}					
id: 30					
InUse: false					
is_active: true					
LastRecordedIP: "256.256.256.256"					
Name: "Sonda Prueba 1"					

- **Prueba 3: Escaneo del entorno** Para esta prueba vamos a utilizar el grupo que hemos creado y asignarle un trabajo de monitorización, consiguiendo así tener datos de monitorización a los que accederemos más adelante.

Detalles del Grupo

18 - GrupoPrueba2

Estado: Activo

Sondas en el Grupo:

29 - aa [Ver Detalles](#)

30 - Sonda Prueba 1 [Ver Detalles](#)

Trabajos Realizados:

No jobs found for this group.

[Parar Trabajo](#)

[Asignar Nuevo Trabajo](#)

Cuando pulsamos “Asignar un Nuevo Trabajo”->“Monitorizar” nos saltará este aviso, indicando que un proceso de monitorización en el entorno ha comenzado

Trabajos Realizados:

No jobs found for this group.

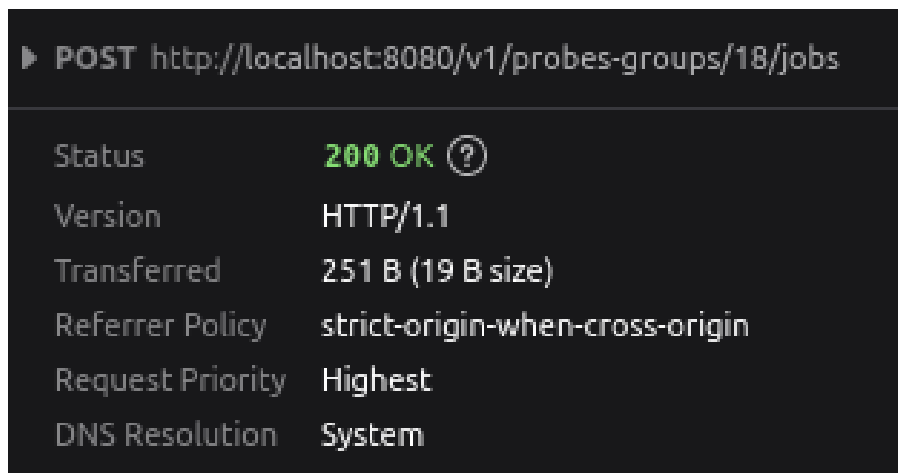
Proceso de monitorización activado!

[Monitorizar](#)

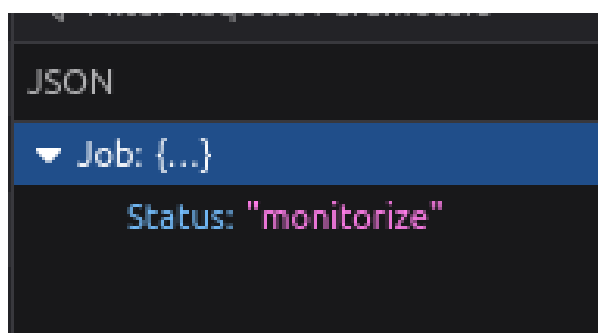
[Escanear](#)

Pruebas de ejecución

Como podemos observar, la respuesta ha salido satisfactoriamente



Este es el cuerpo con el que la aplicación hace la petición. Si bien es verdad que puede parecer simple, esto se realiza para poder hacer un escaneo general, sin ningún tipo de limitación para que de esta forma obtengamos toda la información de nuestro entorno sin excepciones.



Esta es la entrada que se nos crea en la base de datos, relacionando el trabajo con el ID del grupo, además de crear una fecha para el histórico, aunque en la imagen no se vea por temas de espacio, se añade una dirección URL que la sonda podrá recibir para dejar el archivo.

#	JobID	BSSID	ESSID	Channel	WaveLength	group_id	Status	created_at	is_active	band	cswitch	security	wildcard	associated_clients
1	23					16	monitorize	2024-06-15 17:00:32	0					0
2	24					16	monitorize	2024-06-15 17:02:10	0					0
3	25					16	stop	2024-06-15 18:13:34	0					0
4	26					16	monitorize	2024-06-15 18:15:07	0					0
5	27					16	monitorize	2024-06-15 18:39:34	0					0
6	28	string	string	0	2.4 GHz	17	scan	2024-06-15 21:01:43	1	string	0	OPN	string	0
7	29					16	scan	2024-06-15 21:12:39	0	ab				0
8	30	98:97:D1:33:6A:1C	MOVISTAR_6A1B	6	2.4 GHz	16	scan	2024-06-15 21:18:23	1	ag	6	WPA2	FF:FF:F...	1
9	31					18	monitorize	2024-06-26 18:39:42	0					0

Como esto es una simulación, la sonda, cuando el trabajo se termine debería mandar el archivo de monitorización al servidor. Como no tenemos sondas en estos momentos lo vamos a replicar haciendo las llamadas directamente en el swagger. Se indican factores como el ID del grupo y el ID del trabajo, pero es importante saber que estos la sonda solo tendrá que utilizar la URL enviada. En el body se manda el archivo y la sonda en cuestión que esta mandando la monitorización. Esto se hace para poder llevar un control de las monitorizaciones

4.2. Realización de pruebas

de cada sonda. En este caso, la sonda que sube el trabajo es la 30, es decir la sonda que añadimos en 4.2

Envía el fichero resultante de una operación monitorize

Parameters

Name	Description
groupid * required integer(\$int64) (path)	ID del grupo a consultar
<input type="text" value="18"/>	
jobId * required integer(\$int64) (path)	ID del trabajo a gestionar
<input type="text" value="31"/>	

Request body required

file
string(\$binary) Archivo resultante del trabajo de scan.

capturas-02.csv

☐ Send empty value

probeld
integer(\$int64) Id de la sonda que sube el archivo.

☐ Send empty value

Esta es la respuesta a la llama anterior, resultando en la subida correcta del archivo.

server response

Code	Details
200	<p>Response body</p> <pre>{ "mensaje": "File uploaded successfully" }</pre> <p>Response headers</p> <pre>access-control-allow-origin: http://localhost:8080 connection: close content-length: 46 content-type: application/json date: Wed, 26 Jun 2024 16:44:35 GMT server: Werkzeug/2.2.3 Python/3.10.12 vary: Origin</pre>

En la base de datos se ha rellendo el campo de la sonda 30, añadiendo las diferentes redes en el campo “data”

Pruebas de ejecución

Result Grid							
Filter Rows:							
#	id	name	last_recorded_ip	in_use	group_id	is_active	data
1	26	Hola1	1.3.1.1	1	16	1	NULL
2	27	Prueba	2.2.2.2	0	17	1	NULL
3	28	SondaPrueba1	256.256.256.256	0	18	1	NULL
4	29	aa	1.1.1.1	0	18	1	NULL
5	30	Sonda Prueba 1	256.256.256.256	0	18	1	{ "networks": [{"bssid": "CC:29:BD:21:BD:9F", "essid": "", "channel": " 2", "security": ""}], {"b...
*							

- **Prueba 4: Realizar un escaneo efectivo** Para poder realizar un escaneo efectivo, necesitaremos la información que hemos recopilado en la prueba anterior. Esta nos ayudará a rellenar las tablas con los diferentes atributos que puede tomar cada apartado. Así quedarían todos los BSSIDs recogidos en la monitorización

Assign Scan Job

BSSID:

All BSSIDs

All BSSIDs

CC:29:BD:21:BD:9F

08:7E:64:F2:D7:D8

74:93:DA:38:D0:FF

10:62:D0:8C:65:8E

44:FF:BA:2D:94:0D

D4:B9:2F:36:9C:00

34:49:5B:8E:0D:16

A4:08:F5:E6:0D:76

F4:69:42:41:56:FF

44:AD:B1:C9:FC:5E

FC:34:97:B3:07:50

CC:ED:DC:79:0F:40

30:B5:C2:22:8E:FE

18:FD:74:A6:73:AC

E4:AB:89:23:B1:0B

60:8D:26:F6:CA:54

D8:E8:44:B2:2A:00

9C:A5:70:3F:A0:05

84:AA:9C:A9:AA:85

64:CC:22:F6:78:C8

A8:02:DB:02:13:6C

08:F4:58:97:4F:10

Si por ejemplo elegimos el ESSID con el nombre “thecastanos” se nos restringirá la tabla de los BSSIDs a solo el que pertenece a ese ESSID. Este comportamiento ayuda a tener escaneos más sensatos y ayuda a los errores que pueda tener el

usuario

Assign Scan Job

BSSID:

All BSSIDs

▼

All BSSIDs

D4:5D:64:85:DA:30

thecastanos

▼

Security:

All Security Types

▼

Channel:

All Channels

▼

WaveLength:

Select WaveLength

▼

Channel Mode:

Select Channel Mode

▼

Channel Switch:

Select Switch Method

▼

Netmask:

Band:

- ☐ A (5 GHz)
- ☐ B (2.4 GHz)
- ☐ G (2.4 GHz)

Show Associated Clients: ☐

Submit

Como se puede ver también funciona en el campo channel, indicandonos solo que ha encontrado información de “thecastanos” en el canal 10. También la seguridad tiene este comportamiento

Assign Scan Job

BSSID:

All BSSIDs

ESSID:

thecastanos

Security:

WPA2

Channel:

All Channels

All Channels

10

Select WaveLength

Channel Mode:

Select Channel Mode

Channel Switch:

Select Switch Method

Netmask:

Band:

- ☐ A (5 GHz)
- ☐ B (2.4 GHz)
- ☐ G (2.4 GHz)

Show Associated Clients: ☐

Submit

Para asignar un netmask, aparecen unas opciones de ejemplo de uso, pero pue-

Pruebas de ejecución

de usarse la mascara que sea mas conveniente para el caso

Netmask:

Band:

- ☐ A (5 GHz)
- ☐ B (2.4 GHz)
- ☐ G (2.4 GHz)

FF:FF:FF:FF:FF:FF

00:00:00:00:00:00

FF:FF:00:FF:00:00

00:FF:FF:00:FF:FF

Esto sería la petición enviada correctamente, con todos los campos asignados.

BSSID:

D4:5D:64:85:DA:30

ESSID:

thecastanos

Security:

WPA2

Channel:

10

WaveLength:

5.0 GHz

Channel Mode:

HT40+

Channel Switch:

2

Netmask:

Band:

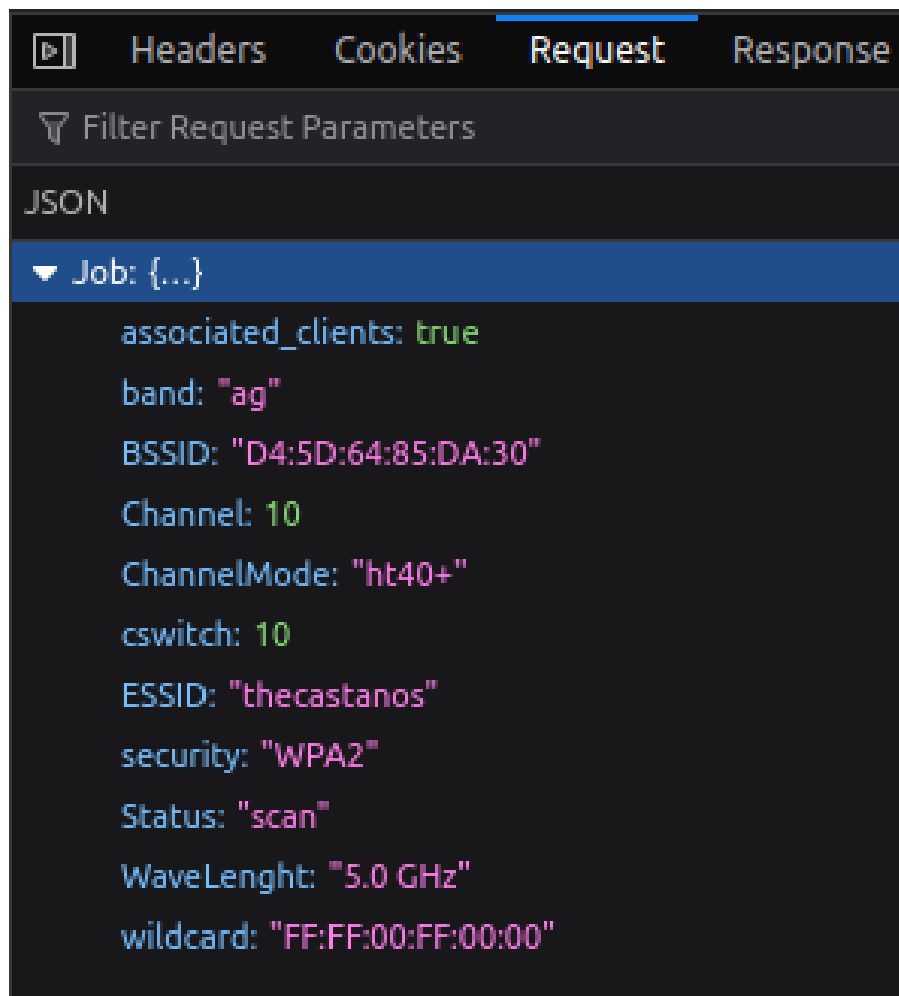
- ☒ A (5.0 GHz)
- ☐ B (2.4 GHz)
- ☒ G (2.4 GHz)

Show Associated Clients: ☒

Submit

Trabajo de escaneo asignado correctamente.

Aquí tenemos el body enviado, mucho más completo que el monitorize pero también mucho más específico para una red en concreto



La sonda recibe la información y puede consultarla en tiempo real, sin saber ni el trabajo ni el grupo a a que está asociada, reduciendo la complejidad de la sonda.

Detalles de la Sonda

ID: 30

Nombre:

Sonda Prueba 1

Última IP registrada:

256.256.256.256

En uso: No

Trabajando actualmente en:

Status: scan

Grupo: 18

BSSID: D4:5D:64:85:DA:30

ESSID: thecastanos

Channel: 10

WaveLenght: 5.0 GHz

Created At: 6/26/2024, 8:53:07 PM

Actualizar

Eliminar

Cerrar

- **Prueba 5: Eliminar Grupo** Para la eliminación de un grupo se deberá hacer una llamada al grupo correspondiente. En este caso vamos a borrar el grupo creado anteriormente con ID 18

DELETE

/probes-groups/{groupId}

Eliminar un grupo de sondas

Eliminar un grupo de sondas dado el ID

Parameters

Cancel

Name

Description

groupId * required

integer(int64)

ID del grupo de sondas a eliminar

(path)

18

Execute

Clear

Responses

Curl

curl -X 'DELETE' \

'http://localhost:8080/v1/probes-groups/18' \

-H 'accept: */*' \

Request URL

http://localhost:8080/v1/probes-groups/18

Server response

Si la llamada es correcta, tendremos un 204 content deleted

Code

Details

200

Undocumented

Response body

{

"message": "Group deactivated and all assigned probes unassigned"

}

Download

Response headers

access-control-allow-origin: http://localhost:8080

connection: close

content-length: 72

content-type: application/json

date: Wed, 26 Jun 2024 19:05:12 GMT

server: Werkzeug/2.2.3 Python/3.10.12

vary: Origin

Responses

Como podemos observar, el grupo 18 tiene un 0 en la columna “is_active”

Result Grid					Filter Rows:		Edit: [
#	id	name	is_active				
1	15	Grupo1	0				
2	16	Grupo1	1				
3	17	Prueba	1				
4	18	GrupoPrueba2	0				
*	NULL	NULL	NULL				

En la tabla de las sondas, observamos que las sondas han sido desasignadas del grupo 18 pero siguen manteniendo todas sus funcionalidades, como los datos subidos y sus características

4	29	aa	1.1.1.1	0	NULL	1	NULL
5	30	Sonda Prueba 1	256.256.256.256	0	NULL	1	{ "networks": [{"bssid": "CC:29:BD:21:BD:9F", "essid": "", "channel": " 2", "security": ""}, {"b...

Pruebas de ejecución

- **Prueba 6: Actualizar una sonda** Para la actualización de una sonda, se podrá editar el nombre y su IP. En este caso vamos a actualizar la sonda SondaPrueba1 para que se llame SondaPrueba2 y tenga una IP de 1.2.3.4

Detalles de la Sonda

Sonda actualizada correctamente

ID: 30

Nombre:

Sonda Prueba 2

Última IP registrada:

1.2.3.4

En uso: No

No hay trabajo asignado actualmente.

Actualizar

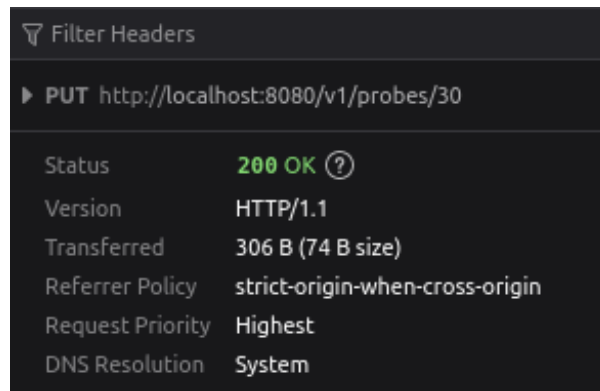
Eliminar

Cerrar

La sonda se actualiza en la base de datos de forma satisfactoria, sin alterar la columna “data” ni “is_active”

Result Grid								Filter Rows	Edit	Export/Import	Wrap Cell Content
#	id	name	last_recorded_ip	in_use	group_id	is_active	data				
1	26	Hola1	1.3.1.1	1	16	1	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...				
2	27	Prueba	2.2.2.2	0	17	1	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...				
3	28	SondaPrueba1	256.256.256.256	0	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...	1	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...				
4	29	aa	1.1.1.1	0	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...	1	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...				
5	30	Sonda Prueba 2	1.2.3.4	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...	1	[{"networks": [{"bssid": "CC:29:BD:21:BD:9F", "ssid": " ", "channel": " 2", "security": " "}], {"b...				
*											

Esta es la petición PUT al servidor, respondido con un 200 OK para indicar que el contenido ha sido modificado.



The screenshot shows a web browser's developer console with the 'Filter Headers' tab selected. It displays the details of a PUT request to the URL 'http://localhost:8080/v1/probes/30'. The status is '200 OK' with a help icon. The response is in HTTP/1.1 format, with a size of 306 B (74 B size). The Referrer Policy is 'strict-origin-when-cross-origin', the Request Priority is 'Highest', and the DNS Resolution is 'System'.

Filter Headers	
▶ PUT http://localhost:8080/v1/probes/30	
Status	200 OK ?
Version	HTTP/1.1
Transferred	306 B (74 B size)
Referrer Policy	strict-origin-when-cross-origin
Request Priority	Highest
DNS Resolution	System

4.2.1. Conclusiones de las pruebas

Estas son algunas de las numerosas pruebas que se han realizado al sistema. En estas pruebas se han considerado los comportamientos de un usuario familiarizado con la herramienta. Aunque la aplicación está diseñada para guiar al usuario, la API ofrece las mismas capacidades que la interfaz, e incluso más, lo que requiere un proceso de prueba exhaustivo.

Todas las pruebas realizadas han sido exitosas, demostrando que el sistema es robusto y fiable. La capacidad del sistema para manejar diversas situaciones y escenarios de uso confirma su calidad. La estabilidad y la precisión de los resultados obtenidos reflejan un diseño cuidadoso y una implementación efectiva. Además, la flexibilidad del sistema, permite tanto a usuarios finales como a desarrolladores interactuar de forma óptima la API, resalta su adaptabilidad. En conclusión, estas pruebas no solo validan la funcionalidad del sistema, sino que también subrayan su desempeño adecuado y su solidez técnica.

Capítulo 5

Impacto del trabajo

5.1. Impacto general

En este capítulo se realizará un análisis del impacto potencial de los resultados obtenidos durante la realización del TFG en diferentes contextos:

- Personal
- Empresarial
- Social
- Económico
- Medioambiental
- Cultural

En dicho análisis se destacarán los beneficios esperados, así como también los posibles efectos perjudiciales. Además, se recomienda analizar el potencial impacto haciendo referencia a los Objetivos de Desarrollo Sostenible (ODS) de la Agenda 2030 que sean relevantes para la solución propuesta¹. En este capítulo se destacarán también aquellas decisiones tomadas a lo largo del trabajo que tienen como base la consideración del impacto. Las soluciones presentadas en este documento contemplan el proyecto en su totalidad, en el cual este trabajo está integrado, ya que su existencia, por sí misma, carece de sentido en términos de resultados directos.

5.1.1. Impacto personal

La herramienta puede tener un impacto significativo a nivel personal, facilitando la gestión y optimización de redes domésticas y personales:

- **Optimización del rendimiento de la red doméstica:** Los usuarios pueden monitorear su red WiFi doméstica para identificar problemas de conectividad y optimizar la configuración del router y los puntos de acceso.
- **Seguridad en el hogar:** Los usuarios pueden ver todos los dispositivos conectados a su red doméstica y detectar posibles intrusos o accesos no autorizados.

¹<https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/>

- **Gestión del ancho de banda:** Los usuarios pueden asignar prioridades de ancho de banda a ciertos dispositivos o aplicaciones, asegurando que actividades críticas como el trabajo remoto o la educación online tengan una conexión estable.
- **Asistencia técnica simplificada:** Con una visión clara del estado de la red, los usuarios pueden identificar y resolver problemas técnicos por sí mismos o proporcionar información detallada a su proveedor de servicios para una resolución más rápida.

5.1.2. Impacto empresarial

En el ámbito empresarial, la herramienta desarrollada puede ser adaptada para su uso en diferentes industrias que requieren monitoreo constante y preciso de sus redes. Esto incluye, pero no se limita a, empresas de telecomunicaciones, centros de datos y organizaciones que manejan infraestructuras críticas:

- **Optimización de redes empresariales:** Las empresas pueden utilizar la herramienta para monitorear y optimizar sus redes internas, asegurando un rendimiento óptimo y minimizando el tiempo de inactividad.
- **Mejora de la seguridad:** La capacidad de detectar intrusiones y accesos no autorizados en tiempo real puede ayudar a proteger datos sensibles y prevenir brechas de seguridad. Además, la idea principal de la herramienta es ser capaz de detectar abusos y eliminarlos, proporcionando una capa adicional de protección.
- **Eficiencia operativa:** La herramienta permite una gestión más eficiente de los recursos de red, reduciendo costos y mejorando la productividad.
- **Base para el análisis de soluciones heurísticas:** Esta herramienta sirve como una base fundamental para el análisis de soluciones que deben configurarse como heurísticas. Permite identificar y ajustar parámetros críticos, asegurando que las configuraciones se basen en necesidades reales y no en suposiciones.

5.1.3. Impacto social

El impacto social de la herramienta se manifiesta en la mejora de la conectividad y la accesibilidad a internet, especialmente en comunidades rurales y áreas desatendidas. Además, este trabajo ofrece una plataforma de referencia para futuros estudiantes e investigadores interesados en la implementación y el análisis de redes WiFi mediante el uso de sondas y APIs RESTful:

- **Mejora de la conectividad:** La herramienta puede ayudar a maximizar el uso eficiente de los recursos de red disponibles, mejorando la conectividad en comunidades rurales y áreas desatendidas.
- **Enseñanza práctica de la seguridad de redes:** Los estudiantes pueden utilizar la herramienta para capturar y analizar paquetes de datos WiFi, observando en tiempo real cómo se transmiten y reciben los datos a través de la red. Este enfoque práctico permite a los alumnos comprender mejor los protocolos de comunicación, detectar posibles vulnerabilidades y aprender métodos de mitigación. La posibilidad de simular diferentes escenarios de ataque y defensa en

Impacto del trabajo

un entorno controlado mejora significativamente el aprendizaje y la preparación de los estudiantes para situaciones del mundo real.

5.1.4. Impacto económico

El impacto económico de la herramienta puede ser significativo, tanto a nivel empresarial como en el ámbito público:

- **Reducción de costos operativos:** La optimización de redes puede reducir costos asociados con el mantenimiento y la resolución de problemas técnicos.
- **Mejora de la productividad:** Una conectividad estable y eficiente puede mejorar la productividad de los empleados y la eficiencia operativa de las empresas.
- **Fomento de la innovación:** La disponibilidad de herramientas avanzadas de monitoreo de redes puede estimular la innovación y el desarrollo de nuevas tecnologías y servicios.

5.1.5. Impacto medioambiental

El impacto medioambiental de la herramienta también es relevante, ya que una gestión eficiente de las redes puede contribuir a la sostenibilidad:

- **Reducción del consumo energético:** La optimización de redes puede contribuir a reducir el consumo energético, disminuyendo la huella de carbono.
- **Gestión de recursos:** Una mejor gestión de los recursos de red puede minimizar el desperdicio y mejorar la eficiencia en el uso de infraestructura tecnológica.

5.1.6. Impacto cultural

El impacto cultural de la herramienta se refleja en la promoción de la digitalización y la integración de tecnologías avanzadas en la vida cotidiana:

- **Promoción de la digitalización:** La herramienta puede contribuir a la adopción de tecnologías avanzadas y a la integración de la digitalización en diversos aspectos de la vida cotidiana.
- **Fomento de la cultura tecnológica:** Al facilitar el acceso y el uso de tecnologías avanzadas, la herramienta puede fomentar una cultura tecnológica y aumentar la alfabetización digital en la sociedad.

5.2. Objetivos de Desarrollo Sostenible

1. ODS 4: Educación de Calidad

- **Meta 4.4:** Aumentar el número de jóvenes y adultos que tienen habilidades relevantes, incluidas habilidades técnicas y vocacionales, para el empleo, el trabajo decente y el emprendimiento.

La herramienta desarrollada puede ser utilizada en entornos educativos para enseñar habilidades técnicas avanzadas en redes y seguridad informática, proporcionando a los estudiantes experiencias prácticas valiosas.

2. ODS 9: Industria, Innovación e Infraestructura

- **Meta 9.1:** Desarrollar infraestructuras fiables, sostenibles, resilientes y de calidad, incluidas infraestructuras regionales y transfronterizas, para apoyar el desarrollo económico y el bienestar humano.

La herramienta ayuda a mejorar y optimizar infraestructuras de redes WiFi, haciendo que estas sean más eficientes y resilientes.

- **Meta 9.5:** Mejorar la investigación científica, mejorar las capacidades tecnológicas de los sectores industriales en todos los países.

El proyecto proporciona una base para investigaciones futuras y desarrollos tecnológicos en el campo de las redes y la seguridad informática.

3. ODS 11: Ciudades y Comunidades Sostenibles

- **Meta 11.3:** Aumentar la urbanización inclusiva y sostenible y la capacidad para la planificación y gestión participativa, integrada y sostenible de los asentamientos humanos.

Al optimizar las redes WiFi en entornos urbanos, el proyecto contribuye a una mejor gestión y uso de los recursos tecnológicos en las ciudades.

4. ODS 12: Producción y Consumo Responsables

- **Meta 12.2:** Lograr la gestión sostenible y el uso eficiente de los recursos naturales.

La herramienta permite una gestión más eficiente de los recursos de red, minimizando el desperdicio y reduciendo el consumo energético asociado con el uso de redes WiFi.

5. ODS 13: Acción por el Clima

- **Meta 13.2:** Integrar medidas relativas al cambio climático en las políticas, estrategias y planes nacionales.

La optimización del uso de redes y la reducción del consumo energético contribuyen a la mitigación del cambio climático.

6. ODS 17: Alianzas para lograr los Objetivos

- **Meta 17.6:** Mejorar el acceso a la ciencia, la tecnología y la innovación y mejorar la cooperación Norte-Sur, Sur-Sur y triangular en estos ámbitos.

El proyecto puede facilitar la cooperación tecnológica y el intercambio de conocimientos entre diferentes regiones y sectores, promoviendo un acceso más equitativo a tecnologías avanzadas.

Capítulo 6

Resultados y conclusiones

6.1. Resultados obtenidos y dificultades encontradas

Los resultados obtenidos tras haber trabajado en este proyecto han sido los esperados, ajustándose a los requisitos establecidos, obteniendo así un resultado satisfactorio. De forma personal, este proyecto me ha ayudado enormemente a aplicar los conocimientos de asignaturas como “Sistemas Orientados a Servicios”, mejorando mis capacidades de montar API REST y pudiendo utilizar el conocimiento aprendido en el programa de Swagger. En general, no poseía experiencia en ninguna de las herramientas utilizadas en este proyecto, causando que mucha parte del comienzo del trabajo y de su desarrollo se haya gastado aprendiendo estas tecnologías. También he aprendido mucho más del tema de redes, ya que la parte de Wi-Fi a penas se da en nuestra escuela y la ciberseguridad es un área en la que tengo mucho afán además de ser una de las razones principales por las que seleccioné este trabajo. Entre todas, la más complicada y con la que más he tenido que lidiar ha sido Flask ya que al estar restringido por la estructura de Swagger Codegen a la hora de tener que implementar el servidor, se han tenido que realizar muchos ajustes que probablemente no sean óptimos. Por la parte de React, al contrario que Python, JavaScript nunca lo había utilizado lo que complicó la tarea de empezar, elevando mucho la curva de aprendizaje. También el HTML modificado que utiliza no fue nada fácil de entender, haciendo que pudiera estar horas hasta saber donde se originaban los errores. Pese a ello, he conseguido superar mis expectativas en este trabajo, habiendo conseguido una herramienta no solo funcional sino que también útil, utilizando herramientas accesibles a todo el mundo.

Otro de los problemas encontrados fue la pobre integración de Codegen con una herramienta de CI/CD como puede ser Gitlab. Esto ha creado muchos problemas a la hora de actualizar la API, algo que a priori debería ser sencillo, cada cambio que se orquestaba significaba tener que volver a implementar el servidor con un código nuevo, ya que el anterior servidor no era válido. Esto repercutía mucho en el avance del proyecto ya hacía todo el proceso de cambiar/añadir un endpoint mucho más tedioso de lo que debería haber supuesto en una primera instancia.

En conclusión me ha gustado mucho poder ser parte de este proyecto y estoy encantado con la experiencia. Me siento entusiasmado por haber conseguido el objetivo final y también de haber construido una arquitectura potente y usable, algo compli-

cado de hacer en ninguna asignatura por el tiempo que conlleva.

6.2. Trabajo futuro

A lo largo del desarrollo de este trabajo, se han identificado varias áreas donde la aplicación puede beneficiarse de mejoras adicionales para aumentar su funcionalidad, usabilidad y eficiencia. A continuación, se describen algunas de las mejoras potenciales:

6.2.1. Implementación de un sistema de login

La implementación de un sistema de login proporcionaría una capa adicional de seguridad y personalización para los usuarios. Esto permitiría:

- **Autenticación de usuarios:** Garantizando que solo usuarios autorizados puedan acceder a la aplicación.
- **Perfiles personalizados:** Permitiendo que los usuarios guarden sus configuraciones y preferencias.
- **Seguimiento de actividad:** Ofreciendo la posibilidad de registrar y revisar la actividad de cada usuario, lo cual es útil para auditorías de seguridad.

6.2.2. Mejora del diseño del front-end

Una interfaz de usuario mejorada puede hacer que la aplicación sea más intuitiva y agradable de usar. Las mejoras en el diseño del front-end podrían incluir:

- **Diseño responsivo:** Asegurando que la aplicación sea accesible y funcional en una variedad de dispositivos y tamaños de pantalla.
- **Interfaz amigable:** Rediseñando elementos de la interfaz para que sean más intuitivos, con menús claros, botones visibles y una navegación sencilla.
- **Actualización visual:** Utilizando un diseño moderno y atractivo que mejore la experiencia del usuario.

6.2.3. Mejora del componente de visualización de sondas

El componente de visualización de sondas puede ser optimizado para ser más intuitivo y funcional:

- **Interactividad:** Añadiendo capacidades interactivas como el arrastre y soltar (drag-and-drop) para organizar sondas.
- **Filtros y búsqueda:** Implementando filtros avanzados y una función de búsqueda para facilitar la localización de sondas específicas.
- **Visualización gráfica:** Incorporando gráficos y diagramas que muestren las conexiones y estados de las sondas de forma visual.

6.2.4. Mejora del escaneo

Para mejorar el proceso de escaneo y hacerlo más eficiente y fácil de usar, se pueden implementar las siguientes mejoras:

- **Selección de redes disponibles:** Mejorar la interfaz para la selección de redes disponibles, permitiendo una visualización clara y organizada de las redes detectadas.
- **Escaneo personalizado:** Permitir que los usuarios configuren y guarden perfiles de escaneo personalizados, facilitando la repetición de escaneos con parámetros específicos.

6.2.5. Mejora de los endpoints de la API

La API podría beneficiarse de una revisión para optimizar su rendimiento y ampliar sus capacidades:

- **Documentación exhaustiva:** Proporcionando documentación completa y detallada para facilitar el uso y la integración por parte de los desarrolladores.
- **Optimización de rendimiento:** Refinando los endpoints actuales para reducir tiempos de respuesta y mejorar la eficiencia.
- **Mejora de endpoints:** Ajuste de los endpoints para que sigan las normas REST establecidas por la comunidad.

6.2.6. Aumento de funcionalidades de la API

Para hacer la API más poderosa y versátil, se podrían agregar las siguientes funcionalidades:

- **Control remoto:** Permitiendo el control remoto completo de las sondas y otros dispositivos conectados.
- **Análisis avanzados:** Incorporando capacidades de análisis de datos más avanzadas, como la detección de patrones y alertas automáticas.
- **Integración con otros servicios:** Facilitando la integración con servicios externos, como bases de datos y sistemas de gestión de redes.

6.2.7. Adición de soporte para nuevos programas

Integrar soporte para programas adicionales ampliará las capacidades de monitoreo y análisis de la aplicación:

- **Soporte para Tshark:** Permitiendo el uso de Tshark para capturar y analizar el tráfico de red directamente desde la aplicación.
- **Extensibilidad:** Facilitando la incorporación de otros programas y herramientas de análisis en el futuro.
- **Configuración simplificada:** Ofreciendo interfaces de configuración fáciles de usar para integrar nuevas herramientas sin necesidad de conocimientos avanzados.

Bibliografía

- [1] AusCERT. *Denial of Service Vulnerability in IEEE 802.11 Wireless Devices*. <https://web.archive.org/web/20161214160125/https://www.auscert.org.au/render.html?it=4091>. Accessed: 2024-06-28. 2004.
- [2] Fatima Salma SADEK et al. «Modeling the Greedy Behavior Attack and Analyzing its Impact on IoT Networks». En: *Procedia Computer Science* 198 (2022). 12th International Conference on Emerging Ubiquitous Systems and Pervasive Networks / 11th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare, págs. 770-775. ISSN: 1877-0509. DOI: <https://doi.org/10.1016/j.procs.2021.12.320>. URL: <https://www.sciencedirect.com/science/article/pii/S187705092102559X>.
- [3] Michael J. Demmer et al. «Towards a Modern Communications API». En: *ACM Workshop on Hot Topics in Networks*. 2007. URL: <https://api.semanticscholar.org/CorpusID:9163799>.
- [4] «API Features Individualizing of Web Services: REST and SOAP». En: *International Journal of Innovative Technology and Exploring Engineering* (2019). URL: <https://api.semanticscholar.org/CorpusID:241888945>.
- [5] Johanna Amann et al. «Mission accomplished? HTTPS security after diginotar». En: *Proceedings of the 2017 Internet Measurement Conference*. IMC '17. London, United Kingdom: Association for Computing Machinery, 2017, págs. 325-340. ISBN: 9781450351188. DOI: 10.1145/3131365.3131401. URL: <https://doi.org/10.1145/3131365.3131401>.
- [6] Vijaya B. Surwase. «REST API Modeling Languages - A Developer's Perspective». En: *International Journal For Science Technology And Engineering* 2 (2016), págs. 634-637. URL: <https://api.semanticscholar.org/CorpusID:54773316>.
- [7] Wes McKinney. *Python for Data Analysis*. O'Reilly Media, 2017. ISBN: 978-1491957660.
- [8] Devndra Ghimire. *Comparative study on python web frameworks: Flask and django*. Mayo de 2020. URL: <https://www.theseus.fi/handle/10024/339796>.
- [9] Robert Martin McCool. *Apache HTTP Server Project*. URL: https://httpd.apache.org/ABOUT_APACHE.html.
- [10] URL: <https://www.aircrack-ng.org/doku.php?id=airodump-ng>.
- [11] URL: <https://www.tcpdump.org/manpages/tcpdump.1.html>.
- [12] Reena Hensley. *What is front-end development and how does it impact your website?* Ene. de 2024. URL: <https://www.sitecrafting.com/articles/what-is-front-end-development/>.
- [13] Elar Saks. «JavaScript Frameworks: Angular vs React vs Vue.» En: (2019).

BIBLIOGRAFÍA

- [14] Envision Apex. *Why Angular is the Right Choice for Enterprise Applications*. Accessed: 2024-06-17. 2023. URL: <https://envisionapex.com/blog/why-angular-is-the-right-choice-for-enterprise-applications/>.
- [15] *MVC Design Pattern*. Último acceso: [24-06-21]. GeeksforGeeks. URL: <https://www.geeksforgeeks.org/mvc-design-pattern/>.
- [16] SmartBear Software. *Swagger Codegen*. <https://swagger.io/tools/swagger-codegen/>. Accessed: 2024-06-22. 2024.