UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

REIMPLEMENTACIÓN, EXTENSIÓN Y OPTIMIZACIÓN DE APLICACIÓN DE DETECCIÓN DE HARDWARE ENFOCADO AL CIBERACOSO

MEMORIA PARA OPTAR AL TÍTULO DE
INGENIERA CIVIL EN COMPUTACIÓN

NATALIA JAVIERA QUINTEROS RETAMAL

PROFESOR GUÍA:
ALEJANDRO HEVIA ANGULO

PROFESOR CO-GUIA:
EDUARDO RIVEROS ROCA

MIEMBROS DE LA COMISIÓN:
IVANA BACHMANN ESPINOZA
TOMÁS BARROS ARANCIBIA

SANTIAGO DE CHILE
2024

Resumen

En un mundo donde el avance tecnológico se encuentra en constante aumento, el mal uso de estas tecnologías también se ve incrementado. Uno de los abusos de la tecnología más preocupantes es el ciberacoso, en particular el *cyberstalking*, conducta que trae consigo graves repercusiones para sus víctimas.

Esta memoria se enfoca en abordar el problema del *cyberstalking* y estudiar formas de mitigarlo. Por ello se estudian distintas opciones de disminuir estas conductas, entre las cuales destaca la aplicación *Lumos* y la solución enfocada a la detección de dispositivos *IoT* que ésta ofrece.

El trabajo se centra en la reimplementación del código de *Lumos*, la aplicación de detección de cámaras y micrófonos usados en *cyberstalking*. El objetivo es transformarla en una aplicación de fácil uso y multiplataforma, de manera de identificar en un espacio físico y acotado los paquetes enviados por distintos dispositivos IoT y reconocer la naturaleza de éstos.

Para ello se fijaron una serie de objetivos. En primer lugar, se consideró una tarea importante estudiar exhaustivamente el diseño y lógica detrás del artículo de *Lumos* y de su correspondiente código fuente, para comprender a fondo el proceso de detección de dispositivos. Una vez estudiada la aplicación *Lumos*, el trabajo se enfocaría en replicar la aplicación para que la solución otorgada sea de fácil acceso, funcional y compatible con la mayoría de los dispositivos móviles utilizados. Esta solución debía ser desarrollada de forma que sea escalable y expansible, de forma que se facilite el trabajo futuro y se incentive la colaboración de externos al desarrollo de la aplicación. Lamentablemente, el código fuente de Lumos resultó infactible de usar, por lo que hubo de explorarse una ruta alternativa.

La solución consistió en una aplicación que se despliega localmente en una Raspberry Pi, con una conexión a un dispositivo móvil mediante WiFi. La Rasberry se encarga de rescatar cabeceras de paquetes mediante el escaneo de canales de red. Tras ser recolectada, la data es procesada para ser mostrada al usuario en el dispositivo móvil, mostrando la cercanía de estos dispositivos mediante la intensidad de señal captada.

De esta forma, se entrega un producto mínimo viable (MVP) de una aplicación que permite la visualización fácil y directa de la intensidad de señal de dispositivos sospechosos, la cual se ve directamente relacionada con la posición aproximada de estos mismos, cumpliendo el objetivo de permitir al usuario identificar potenciales áreas donde debe buscar dispositivos espía.

A Felipe y a mi familia.

Agradecimientos

En primer lugar, me gustaría agradecer a Felipe, mi esposo, mejor amigo y amor de mi vida, por apoyarme durante todo este proceso, por siempre darme fuerzas y motivación para seguir, y por inspirarme cada día a ser mejor persona.

A mi hermana, Daniela, por escucharme y estar dispuesta a ayudarme siempre. A mi mamá por ser siempre tan buena, darme tanto amor y mimarme. A mi papá por darme su apoyo y acompañarme siempre que pudiese a donde yo necesitase durante todos estos años, a mi hermano Cristián por alivianarme la vida constantemente.

A mi abuelita Cristy por demostrarme siempre lo orgullosa que está de mí, y a mi abuelita Tere por siempre ser tan cálida conmigo.

Doy también las gracias a mi otra familia, a Myriam que siempre me ha tratado como una hija más, a Manuel, que se ha preocupado de integrarme, a los hermanos de Felipe, Matías, Manuel, Osiel, Mauricio, que nunca dudaron en tenderme una mano cuando lo necesitase. A la abuelita de Felipe, Eloisa, que en cada oportunidad que se le presentó, me demostró su cariño y amor.

A mis profesores, Eduardo, que fue un pilar fundamental en el desarrollo de mi trabajo de título, entregándome los mejores consejos, dándome mucho de su tiempo, y teniendo mucha paciencia conmigo, y a Alejandro, que me entregó todas las herramientas necesarias, me orientó, motivó e impulsó a integrarme en el mundo de la ciberseguridad, dándome la oportunidad de desarrollar este tema de memoria que me hizo confirmar que esto es lo que quiero hacer.

A mis amigas y amigos, a Elías que siempre fue un muy buen amigo, Lucas que siempre apañó a todo, a Bryan que nunca se negó a prestarme ayuda y que siempre me apoyó mucho, y a Chio que en todo momento me hizo sentirme integrada. A mis amigues shanos, Javi, Benja, Milan, Isi, Dano, Poup, Vale, por hacerme siempre reír tanto y compartir conmigo esas tardes de D&D y juegos de mesa, a Claudia y Javi por enseñarme que las amistades pueden perdurar hasta el infinito. Al team Michil, que fue el mejor equipo que he tenido y que me hizo darme cuenta de que trabajar en equipo puede ser entretenido.

A los demás integrantes del CLCERT también, que me apoyaron mucho con mi trabajo de título, mostraron su entusiasmo por mi trabajo, me enseñaron un montón de cosas y compartieron tantos días conmigo en el lab. Hicieron el desarrollo de esta memoria mucho más ameno y enriquecedor.

A cada persona que formó parte de este proceso, todos ocupan un lugar muy especial en mi corazón, y me entregaron recuerdos que me van a acompañar por siempre. Ustedes me hicieron la persona que soy hoy.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Contexto

El acecho o *stalking*[1] (en inglés) es un problema común en nuestra sociedad, el cual afecta especialmente a las mujeres. De hecho, según un estudio de Smith, S.G., Basile, K.C., & Kresnow (2022), cerca de 1 de cada 3 mujeres estadounidenses reporta haber sido víctima de *stalking* en algún momento de su vida mientras que, en el mismo período, sólo 1 de cada 6 hombres estadounidenses reportó haber sufrido lo mismo [7].

Por otro lado, si bien el desarrollo de la tecnología de las ultimas décadas ha traído muchos beneficios a la humanidad, algunas personas han encontrado la forma de utilizar dichas herramientas para facilitar malas prácticas. En específico es posible reconocer el *cyberstalking* como un tipo de *stalking* que ocurre en el ciberespacio con ayuda de herramientas tecnológicas, como micrófonos, cámaras y dispositivos de rastreo.

Este tipo de conducta invasiva representa una seria amenaza para la privacidad y la seguridad de las víctimas, ya que permite a los perpetradores acceder a su vida digital y física sin su consentimiento. El *cyberstalking* puede tomar diversas formas, desde la vigilancia constante mediante cámaras y micrófonos ocultos hasta el rastreo de la ubicación de la víctima a través de dispositivos de geolocalización o el acceso no autorizado a sus cuentas en redes sociales o correos electrónicos.

Las ocurrencias más comunes de *stalking* se llevan a cabo mediante el uso de tecnología, pues tal como se menciona en un informe realizado por "*Congressional Research Service*" de Estados Unidos emitido el 2022, en una encuesta donde alrededor de 3.4 milliones de víctimas de *stalking* tomaron parte, el 80 % de estas aseguró que se involucró tecnología en la realización de éste [3].

[1] Se califica como *stalking* a la vigilancia y seguimiento indeseado de una entidad, hacia otra persona.

1.2. Soluciones para contrarrestar el *cyberstalking*

Desde el mundo académico han surgido algunas propuestas para el problema del *spyware* (en específico para este caso *stalkerware*) como *MalwareBytes* o *Certo Antispy*[2] que se enfocan en la detección de aplicaciones de tracking instaladas en el dispositivo móvil de la víctima. Asimismo, para enfrentar el problema de los dispositivos de rastreo GPS, se encuentra el dispositivo *GPS bug detector*. Sin embargo el problema de identificar dispositivos IoT[3] escondidos como cámaras o micrófonos no ha tenido muchos avances en el mercado, por lo que se considera un caso de estudio interesante.

En *"Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment"* (*USENIX*) los autores se enfocan en el problema específico de identificar cámaras escondidas en lugares de residencia transitoria como moteles, hoteles y *Airbnb's*. Lumos considera el diseño y desarrollo de una aplicación para la detección de dispositivos IoT en espacios acotados mediante detección de paquetes de red. La solución implementada en *Lumos* resulta entonces interesante de investigar, pues permite identificar dispositivos espía en un espacio determinado sin la necesidad de tener información de antemano sobre ellos.

1.3. Problema y relevancia

En la actualidad, la única solución reportada para la detección de dispositivos espía ocultos por medio de paquetes es *Lumos*. Esta aplicación concentra su valor en la identificación de dispositivos espía sin necesidad de conocer el lugar de antemano, razón por la cual ha sido ampliamente reconocida en el mundo académico. Al examinar el mercado de aplicaciones disponibles, se encontró que ninguna de ellas aborda de manera específica el desafío de identificar cámaras ocultas mediante un procedimiento similar al de *Lumos*. Esta carencia muestra la necesidad de desarrollar una nueva aplicación que llene esta brecha y proporcione a los usuarios una herramienta confiable y efectiva para proteger su privacidad.

Sin embargo, el estado actual de *Lumos* muestra ciertas falencias que afectan su funcionalidad y utilidad. La complejidad y dificultad en la ejecución y uso de la aplicación (necesidad de herramientas específicas como *XCode* para ejecutar la interfaz gráfica) son factores que limitan su accesibilidad y su efectividad. Además, la falta de una metodología de diseño, la ausencia de documentación adecuada, e implementación poco rigurosa y confusa, dificultan la comprensión y extensión del código disponible.

Las mencionadas falencias acarrearon consigo ciertas repercusiones en el desarrollo del proyecto, tales como que se invirtieron aproximadamente tres meses en intentar levantar la aplicación sin éxito. Tras una extensiva revisión del código se concluye que el trabajo mencionado en el articulo no se encontraba disponible en el repositorio del proyecto. Esta situación evidencia la urgencia de desarrollar una alternativa a *Lumos* que funcione y sea más fácil de instalar y utilizar.

[2] https://www.certosoftware.com/

[3] Dispositivos Internet of Things. Se comunican mediante el uso de redes de internet.

Ahora, ¿por qué es necesario elaborar una alternativa a *Lumos* enfocada a afrontar el problema del *stalking*? El *stalking* causa sobre sus víctimas múltiples trastornos mentales, tales como: paranoia, sensación de inseguridad, depresión y miedo entre otros [1]. Ofrecer una solución que permita otorgarles tranquilidad y certeza de que están siendo víctimas de *stalking* resulta entonces crucial, pues permite a las víctimas buscar ayuda y un lugar seguro, con evidencia que les respalde a tal punto de poder iniciar un proceso judicial contra el perpetrador.

De esta forma, el crear una aplicación que cumpla las mismas funciones que *Lumos*, que tenga una instalación clara y que sea mas directa en su interfaz de usuario, resulta esencial para alcanzar una mayor utilidad. Además, se busca asegurar que esta nueva herramienta sea compatible con diversos dispositivos y sistemas operativos, lo que permitiría una amplia accesibilidad.

1.4. Objetivos

Para la realización de la memoria, se definen una serie de objetivos. En primer lugar, se consideró indispensable estudiar detalladamente la aplicación Lumos, para obtener una percepción más profunda de las partes que conforman la aplicación y distinguir cuales de estos elementos pueden ser reutilizados en el desarrollo de una aplicación que aborde el problema de la misma manera que propone Lumos.

En segundo lugar, se debe replicar la aplicación *Lumos* en la forma de una aplicación web. Se busca que para utilizar la nueva aplicación, la persona deba solamente instalar un sistema operativo con todo lo necesario para ejecutar la aplicación ya preparado en una *Raspberry*. Esto permitiría que la solución sea de fácil acceso (el tiempo en configurarla sea aceptable), compatible con la mayoría de los dispositivos usados, utilizable y funcional.

En tercer lugar, se planea implementar una aplicación amigable con el usuario, de forma que una persona que no posea experiencia en el uso de aplicaciones móviles no tenga problemas de entendimiento con el uso de la aplicación. De la misma forma la configuración de los dispositivos deberá tener instrucciones directas y fáciles de seguir, de manera que usuarios sin conocimientos especializados pueda saber que hacer.

Finalmente, se fija como último objetivo desarrollar una aplicación escalable y extensible, es decir, que posea una metodología sólida de diseño, una documentación clara y completa para facilitar su mantenimiento, trabajo futuro y comprensión del código.

Capítulo 2

Estado del Arte

2.1. Problemática

La evolución de la tecnología ha traído consigo varios cambios, tanto para bien como para mal. En particular, como se mencionó, uno de los problemas relativos a los avances tecnológicos es el *cyberstalking*. Según un estudio publicado en *RAND Corporation*, los casos de *cyberstalking* procesados por un juicio en Estados Unidos han ido aumentando con el paso de los años [2]. Más específicamente, el estudio mencionado muestra que los casos de *cyberstalking* han ido en crecimiento desde el 2014, alcanzando un peak de 80 casos procesados por un juicio. Gráficamente estos se ven de la siguiente forma:

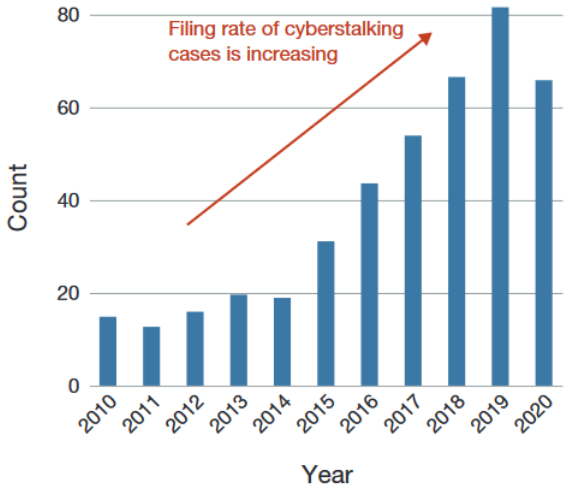

Figura 2.1: Gráfico de casos de *cyberstalking* vs año, obtenida del reporte *"cyberstalking A Growing Challenge for the U.S. Legal System."*

Los resultados anteriores evidencian el aumento del cyberstalking a lo largo de los años, lo cual puede ser vinculado al aumento tanto en la utilización de dispositivos tecnológicos como en la creciente disponibilidad de los mismos.

2.2. Tecnologías involucradas en el *cyberstalking*

Dentro del mismo estudio mencionado [2], se muestran las tecnologías más utilizadas para llevar a cabo *cyberstalking*. Las cifras mostradas en la tabla 2.1 corresponden a un total de 412 casos judiciales de ciberacoso.

Tecnologías	Número de casos
Mensajes de texto, llamadas telefónicas, etc	252
Redes sociales	177
Comunicación anónima (mensajes encriptados, remailers, etc)	67
Hacking (*pishing, malware*, etc)	41
Dispositivos de seguimiento	29
Multimedia (cámaras, fotos, grabaciones, etc)	202

Tabla 2.1: Número de casos por tecnología usada

De la tabla anterior se desprende que en general las tecnologías más utilizadas para el *cyberstalking* corresponden a llamadas telefónicas y mensajes de texto no solicitados, y el uso de multimedia obtenida generalmente de dispositivos ocultos para extorsionar a las víctimas.

Además, como se puede ver, existen diversos tipos de herramientas que pueden vulnerar la privacidad de un individuo mediante el *stalking*. Entre ellas, es posible notar el *stalkerware*, que consiste en *spyware*[1] o software de monitoreo utilizado específicamente para ciberacoso, el cual, en gran parte de los casos involucra el seguimiento por GPS. Otras herramientas comúnmente usadas para facilitar el *stalking* corresponden a dispositivos tecnológicos como cámaras, micrófonos o dispositivos de seguimiento GPS, los cuales en general corresponden a dispositivos IoT. Si bien estos dispositivos y aplicaciones son usualmente pensados para el cuidado de menores de edad, podrían ser utilizados para seguir a personas adultas sin su consentimiento, lo cual muchas veces escala a otros abusos y crímenes [9].

Cuando una persona que realiza *cyberstalking* utiliza *stalkerware*, lo hace generalmente con el objetivo de identificar su ubicación geográfica, mensajes e historial de búsqueda de la víctima. Para ello usualmente se utilizan aplicaciones destinadas al seguimiento de personas, mediante la intromisión al dispositivo móvil de la persona que desean acosar, sin que esta se dé cuenta. A pesar de que esto sea un problema grave, en general no hay leyes que resguarden a las víctimas de estas actividades, debido a que la situación que se enfrenta es relativamente nueva [5].

[1]Software que recolecta información de una persona con la finalidad de divulgarla a terceros sin el expreso consentimiento del afectado. Puede encontrarse tanto en malware como aplicaciones legales.

2.3. Soluciones disponibles para identificar aplicaciones de *stalkerware* y dispositivos de seguimiento

Afortunadamente hoy en día existen diversas soluciones para combatir el *stalkerware*, entre las cuales se encuentran aplicaciones como *MalwareBytes* y *Certo Antispy*, entre otras. El funcionamiento de estas aplicaciones es bastante sencillo y eficiente. Tras ser instaladas, estas herramientas realizan un escaneo completo en busca de posibles rastros de *stalkerware* en el sistema. Luego comparan los archivos y procesos presentes con una base de datos extensa, que se actualiza de manera constante con información sobre amenazas conocidas y características típicas de *stalkerware*. De esta manera, la detección se lleva a cabo siempre y cuando el *stalkerware* que se quiere identificar se encuentre en esta base de datos, lo que permite que los usuarios estén protegidos la mayoría del tiempo de posibles intrusiones no deseadas en su privacidad.

Sin embargo, el proceso difiere un poco cuando se trata de dispositivos de rastreo GPS en lugar de aplicaciones de rastreo. En estos casos, por lo general, la mejor opción es buscar los dispositivos de rastreo GPS en lugares conocidos (como por ejemplo, cuando se trata de dispositivos ocultos en autos, estos generalmente están conectados a la toma de energía). Otra opción es utilizar un detector de dispositivos de rastreo GPS. En el caso particular de los dispositivos de rastreo *AirTag*, la aplicación *Find My* de *Apple* emite automáticamente una alerta si detecta comportamientos inusuales [4].

2.4. Incidencia de dispositivos IoT en el *cyberstalking*

Dentro de las tecnologías empleadas para llevar a cabo *ciberacoso* como micrófonos y cámaras, los dispositivos IoT han cobrado gran relevancia. El aumento de estos dispositivos en el mercado, ha traído consigo facilidades para realizar *cyberstalking*, pues existen personas que utilizan estas tecnologías en su beneficio para llevar a cabo actos invasivos para la privacidad de otras personas. El problema de identificar estos dispositivos se hace más notorio cuando éstos son escondidos para acceder a la información privada de las víctimas o cuando se realiza un seguimiento sin consentimiento de otros individuos. La detección de dispositivos IoT ocultos plantea un desafío relativamente reciente, directamente vinculado al progreso tecnológico de estos dispositivos y al constante incremento en la cantidad de personas que los utilizan.

Dado que estas tecnologías comenzaron a masificarse a principios de los años 2000 [8], los protocolos utilizados para comunicar estos dispositivos entre sí y con la Internet aún no se encuentran completamente estandarizados, lo que complica la tarea de detectarlos directamente. Existen proyectos en curso para abordar y mitigar estos problemas, tales como el detector *JMDHKK*, el cual se enfoca en encontrar dispositivos espía dentro de espacios pequeños mediante el uso de detectores de campo magnético, detectores de frecuencias de radio y detectores GPS simultáneamente. Sin embargo, son de corto alcance, costosos y conspicuos. Para la detección de dispositivos mediante conexiones WiFi se encuentra al citado *Lumos* [6]. Esta solución resulta interesante debido a que parece accesible y comprensible.

Es por ello que se toma la decisión de utilizar ésta aplicación como guía para elaborar una solución que resultase fácil de desplegar y de esta forma ayudar a los usuarios que necesitasen usar la aplicación.

2.5. Problemas encontrados en la aplicación "*Lumos*"

Lamentablemente, y como fue descubierto durante el desarrollo de esta memoria, *Lumos* se encuentra en una fase preliminar de desarrollo, no disponible para su uso masificado. Esta limitación en la accesibilidad dificulta la tarea de identificar dispositivos potencialmente involucrados en situaciones de *cyberstalking* y la detección de dispositivos espía en entornos desconocidos. A pesar de las falencias de esta aplicación, no existen herramientas en el mercado comparables (que puedan localizar dispositivos IoT con facilidad). Esto transforma la solución de *Lumos* en un acercamiento indispensable para comprender el proceso de detección estos dispositivos.

Además, la aplicación (*Lumos*) es difícil de instalar y de usar. Para un usuario sin conocimientos en programación que se encuentre en una situación donde necesite detectar la presencia de cámaras a su alrededor urgentemente, el uso de *Lumos* podría resultar una tarea complicada (y muchas veces imposible bajo esas condiciones). La aplicación requiere habilidades técnicas avanzadas que un usuario común no posee, lo que dificulta la obtención rápida y efectiva de resultados. En situaciones en las que el tiempo es limitado y se necesita actuar con prontitud, la complejidad de *Lumos* representa un obstáculo significativo para obtener la información necesaria en el momento preciso.

2.6. Propósito del trabajo

El propósito de este trabajo está intrínsecamente asociado a Lumos: require su estudio y re-implementación, simplificando el proceso necesario para tenerla funcionando. La focalización en *Lumos* es debido a que no hay avances considerables en el desarrollo de otras aplicaciones de esta naturaleza, aparte de la aplicación existente ya mencionada. Además, debido a que las víctimas de *stalking* podrían no tener conocimientos avanzados de tecnología, resulta necesario que la solución entregada sea consciente de la privacidad de sus usuarios, compatible con la mayoría de los dispositivos móviles comunes, y fácilmente utilizable.

La solución ofrecida es novedosa entonces, pues integra elementos que son difíciles de usar, y facilita su accesibilidad a usuarios que no tengan conocimientos informáticos, y permite que estos usuarios obtengan resultados tangibles, algo que, como se mencionó, no se encuentra disponible hoy en día.

Capítulo 3

Situación actual

Una de las grandes inspiraciones de esta aplicación fue el trabajo[1] de Jessica Matus, abogada de la Universidad de Chile, donde se aborda la violencia digital, los problemas que acarrean los avances de la tecnología y la falta de evidencia a la hora de realizar denuncias. Es por ello que surge la idea de realizar una aplicación que permita evidenciar situaciones de *cyberstalking*.

En el contexto de la búsqueda de soluciones para la detección de dispositivos IoT ocultos, emerge como una opción la aplicación conocida como "*Lumos: Identifying and Localizing Diverse Hidden IoT Devices in an Unfamiliar Environment*". Esta herramienta se destaca por su capacidad para identificar dispositivos IoT a través de la detección de paquetes de datos, mediante la conexión de una Raspberry que captura headers de paquetes de datos mediante el uso de un *dongle WiFi* que soportase modo monitor, y el uso de un dispositivo iOS para realizar un mapa de las ubicaciones de los dispositivos mediante el uso de VIO (*Visual Inertial Odometry*). Un estudio preliminar del artículo académico publicado en la prestigiosa conferencia *USENIX* revela el procedimiento utilizado para llevar a cabo la detección de estos dispositivos en entornos desconocidos.

Lumos afirma en su paper, poseer una interfaz gráfica que se encarga de identificar el espacio en el que se encuentra el usuario. Esta interfaz se encuentra implementada mediante el uso de *Unity* y *XCode* de *Apple*[2], y debía ser ejecutada mediante el mismo. En un principio la ejecución de esta parte de *Lumos* se vió restringida por muchos factores, tales como que las versiones de *XCode* requeridas no eran compatibles con el *MacBook* utilizado para correr la aplicación, o que la aplicación tuviese escasa documentación y nulas metodologías de diseño.

La segunda sección del código de *Lumos* correspondía a una aplicación realizada para ser usada en *Raspberries* que tuviesen *monitor mode* disponible. El estudio e intento de ejecución de esta parte de la aplicación evidenció graves problemas de diseño de ésta, que a posteriori significaron que no se pudiese ejecutar la aplicación.

Tras haber analizado un poco el funcionamiento de *Lumos*, se evaluaron los aspectos de los

[1] https://datosprotegidos.org/jessica-matus-participa-del-conversatorio-proteccion-de-datos-personales-con-enfoque-de-genero-organizado-por-sula-batsu/

[2] Programa para realizar aplicaciones de dispositivos iOS.

que una reimplementación de *Lumos* podía prescindir. En primer lugar, dado que el análisis de paquetes de red mediante *monitor mode* y *Visual Inertial Odometry* de *Lumos* no extrae las coordenadas de altitud (solo las de latitud y longitud), se consideró que no resultaba necesario mostrar las posiciones mediante realidad aumentada (como lo hace *Lumos*). Es por ello que se tomó la decisión de simplificar la interfaz, dejando de lado el uso de realidad aumentada, de forma que se redujeran los recursos necesitados. Gracias a esta decisión, se notó que bastaba con crear una aplicación web para mostrar la interfaz en el dispositivo móvil que solo requiriese el uso de acelerómetro, esto pues una aplicación web puede acceder a los datos del acelerómetro del celular sin dificultades.

Una evaluación más extensiva del código de *Lumos*, disponible en su repositorio de Github, refleja que el código entregado se encuentra incompleto.

3.1. Solución inicial propuesta

Incialmente se propuso el desarrollo de una aplicación que cumpliese con los siguientes requisitos:

1. Utilizar como base el código de *Lumos* para desarrollar una nueva aplicación, implementada con *polyfill*[3] para mantener la compatibilidad visual entre dispositivos.
2. La aplicación debe poder ser accedida por medio de un dispositivo móvil externo a través de *Bluetooth*, generando una red local PAN[4].
3. La aplicación debe estar levantada en un dispositivo que permita detección de paquetes usando monitoreo de red (En especifico una Raspberry que permita modo monitor).
4. Implementar la aplicación utilizando el *framework Django*[5].
5. Ajustar un sistema operativo basado en Debian, para que la configuración de la *Raspberry* sea lo más estándar posible, lo que se planea hacer con una *golden master image*[6], o con *pi-gen*.
6. La data de localización se extrae del acelerómetro presente en el celular del usuario, la cual se envía a la Raspberry por medio de una conexión Bluetooth. Una vez realizada esta conexión, la Raspberry inicia un proceso de detección de paquetes, procesando la información en base a la información capturada
7. La data procesada se despliega en una interfaz gráfica basada en la que es utilizada para la búsqueda de Airtags de Apple (Find My).

[3]Módulo que permite compatibilidad entre navegadores web.
[4]Red que conecta dispositivos cercanos, generalmente utilizada para las conexiones *Bluetooth*.
[5]*Framework* que permite desarrollar aplicaciones web mediante el uso de modelo vista-controlador.
[6]Imagen de un computador con todas sus configuraciones y archivos.

Sin embargo, una vez realizado un estudio más extensivo con respecto al acercamiento que se deseaba tomar, se notaron ciertos problemas:

1. **Conexiones bluetooth:** Pruebas preliminares e intentos de conexión evidenciaron que en ciertos dispositivos tenían problemas de compatibilidad con el chip de *Bluetooth* integrado en la *Raspberry*.
2. **Raspberry sin chip externo no permite detección de paquetes:** El uso de modo monitor no está siempre disponible, un factor obviado en la propuesta de la solución es que se requiere un *chipset* de red específico para poder realizar *sniffing* de red.
3. **Uso del código de *Lumos*:** Como se mencionó, el código de *Lumos* se encuentra aparentemente incompleto y con muy poca documentación, lo que hace imposible el uso de *Lumos* para obtener elementos a ser usados de base para el desarrollo de una aplicación.

3.2. Segunda iteración de solución propuesta

Por consiguiente, hubo que diseñar una segunda solución. Ésta está compuesta por cuatro módulos que poseen las siguientes características:

1. **Módulo de detección de dispositivos:** Se concentra en encontrar señales de red para identificar los dispositivos cercanos al usuario mediante modo monitor.
2. **Módulo de localización del usuario:** Asíncronamente a la detección de señales de los dispositivos se deberá localizar el dispositivo que escanea la señal mediante el uso de acelerómetro, para saber en que punto de la habitación el usuario encontró la señal.
3. **Módulo de procesamiento de información:** Obtenidos los datos de señal obtenidos a partir de *sniffing* de red, y la posición del usuario, esta información se une para obtener los lugares donde la potencia de las señales captadas son más fuertes.
4. **Módulo de despliegue de información:** Una vez obtenida y procesada la información de señal, se despliega al usuario mediante un gráfico de dispersión, donde cada punto muestra la intensidad de señal recibida mediante el uso de colores, similar a un mapa de calor, identificando los lugares donde podrían encontrarse distintos dispositivos espía.

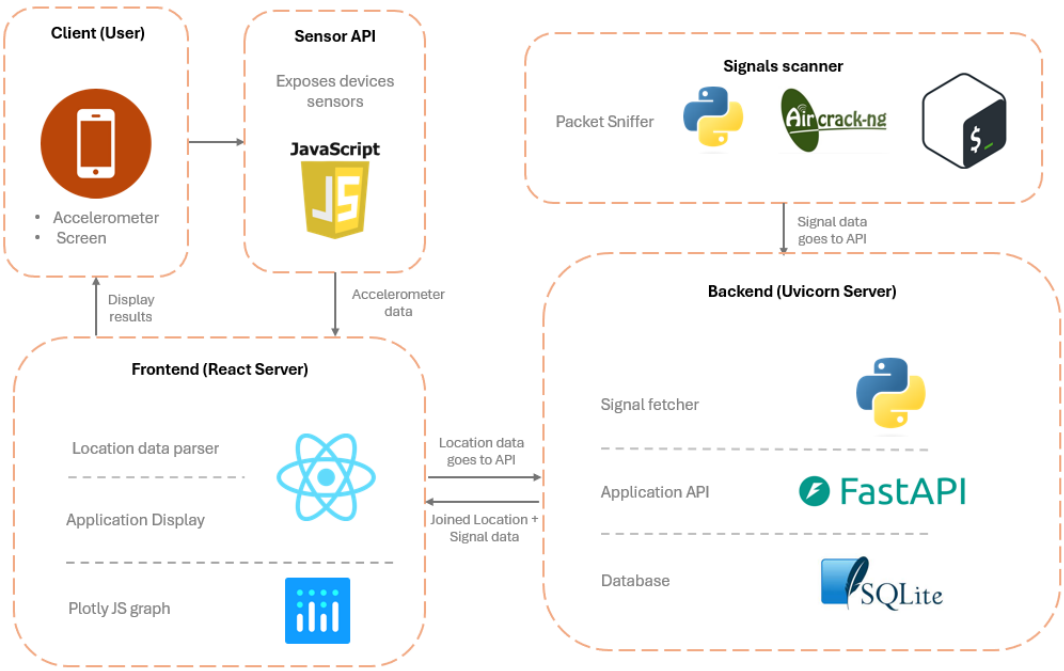

Figura 3.1: Arquitectura inicial propuesta de la aplicación

Esta solución presenta problemas significativos, que no fueron considerados en un principio, llevando a que esta solución se descartase:

- **Imprecisión de la aceleración:** El uso de la aceleración para identificar la posición del usuario no resulta una buena forma de abordar el problema, pues para obtener la posición en base a la aceleración, es necesario integrar dos veces el resultado de la aceleración obtenida, lo cual a su vez significa que cualquier error al captar la aceleración se propaga mucho al realizar la transformación a posición.
- **Gráfico de dispersión es ambiguo:** Al no existir puntos de referencia en el gráfico de despliegue de data, la interpretación de los resultados se hace complicada para el usuario.

3.3. Evaluación de posibles tecnologías

Una de los estudios preliminares principales involucró la decisión de la tecnologías que se usarían para la creación de la aplicación. Para ello se consideró el potencial de lo que podían ofrecer, lo que se detalla a continuación:

Tipo de tecnología	Tecnologías	
Construcción de la Aplicación	***React + FastApi*** Al ser React implementada en JS se puede acceder fácilmente a la API de JS. La combinación FastApi/React es más liviana	***Django*** Ofrece buen manejo de bases de datos y de backend, sin resultar complicada. Posee mucha documentación.
Sistema Operativo	***Debian (Base)*** Más liviano y rápido. Más estable.	***Kali Linux*** Se encuentra configurado para ser usado en hacking ético y posee configuración para ARM.
Gestor de Bases de datos	***PostgreSQL*** Robusto, eficiente y gestiona las cargas muy bien. Bueno para bases de datos masivas.	***SQLite*** Liviano, fácil y directo de levantar y usar. Bases de datos fáciles de manipular.

Tabla 3.1: Tabla comparativa de distintas tecnologías

En la sección 4.2, es posible apreciar las elecciones de tecnologías utilizadas junto a breves explicaciones de las razones por las que se decide utilizar estas tecnologías.

Capítulo 4

Solución

Después de la evaluación descrita en el capítulo anterior, se decidió que la solución implementaría la estructura dada en la figura 4.1.

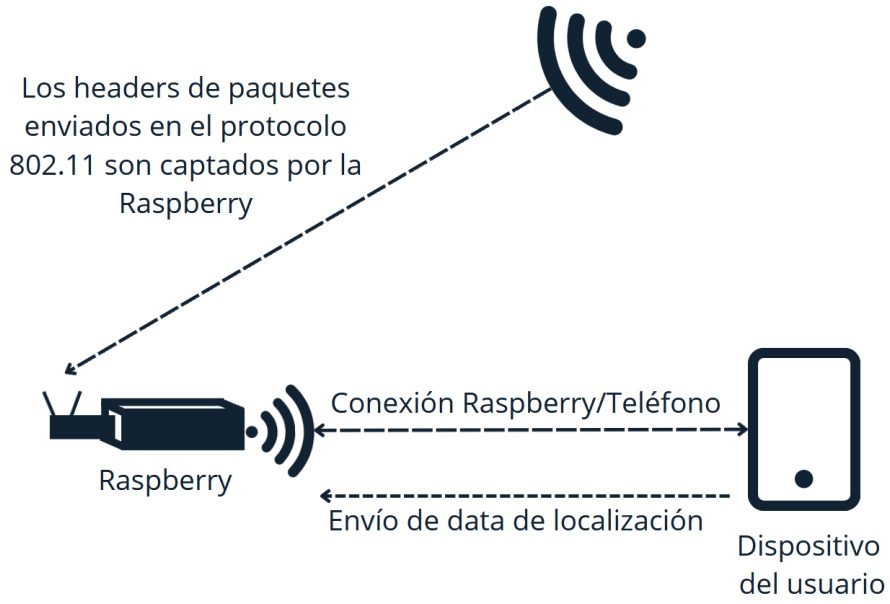

Figura 4.1: Estructura de conexiones

El usuario conecta su dispositivo móvil a una *Raspberry* (con un chip de red que permita modo monitor habilitado), mediante un punto de acceso WiFi. A su vez, la *Raspberry* levanta dos servidores, un frontend y un backend, de forma que el usuario puede acceder a estos localmente. Estos servidores se encargan de gestionar el flujo de datos y capturar paquetes de red.

En el resto de la sección se profundizará en la funcionalidad de cada segmento de la aplicación, y los pasos seguidos para construirla.

4.1. Descripción general de la solución:

Como se mencionó anteriormente, el objetivo de esta aplicación es la detección de dispositivos espía mediante la detección de paquetes en un lugar acotado y cerrado, y la unión de la información de los paquetes detectados junto a data de localización obtenida mediante el uso de acelerómetro de un celular. Para ello se requiere separar las tareas en distintos módulos con labores especializadas.

La aplicación posee las siguientes secciones:

- **Módulo de detección de dispositivos:** Para encontrar los dispositivos presentes en el área se realizan escaneos de red. Estos escaneos de red permiten notar cuales son los distintos dispositivos que se encuentran conectados en las cercanías del usuario. Lo anterior se lleva a cabo por medio de detección de paquetes mediante el uso de modo monitor, de modo que no sea necesario realizar una conexión a un punto de acceso para identificar los distintos dispositivos cercanos.
- **Manipulación de información:** Obtenidos los datos de señal conseguidos a partir de *sniffing de red*, y la posición del usuario, parte de la información se almacena en bases de datos, mientras que el resto de la data se envía en tiempo real mediante el uso de *websockets*.
- **Módulo de despliegue de información:** En primer lugar, se despliegan todas las *MAC Addresses* encontradas junto a su intensidad de señal reportada más alta, donde el usuario puede seleccionar en una tabla cual es el dispositivo que desea buscar. Una vez realizado esto, para que el usuario pueda interpretar la información, se despliega un gráfico de indicador[1], el cual muestra en tiempo real que tan cercana es la señal del dispositivo cuya MAC se está buscando.

4.2. Tecnologías usadas

A continuación se entregan breves descripciones de las tecnologías que fueron usadas en el desarrollo de la aplicación:

- **Kali Linux:** Dado el uso de herramientas utilizadas para el *hacking* ético tales como "*aircrack-ng*" y "*airodump-ng*", se decide utilizar *Kali* como sistema operativo para la *Raspberry* que fue utilizada para el desarrollo de esta aplicación. De esta forma se ahorró tiempo en la configuración de la *Raspberry* para su uso.
- **React:** Para realizar el frontend y manipular fácilmente el flujo de data de la aplicación, se utiliza *React* con el gestor de paquetes *npm*. Los *hooks*[2] de *React* resultaron útiles para manipular los estados de los distintos escaneos, mantener variables entre componentes sin la necesidad de declarar clases y mantener el código legible y ordenado, lo cual consistía en uno de los objetivos del trabajo.

[1] Refiere a los gráficos comúnmente utilizados para velocímetros.
[2] Funciones de React que permiten manipular estados.

- **FastAPI:** Se utilizó este *framework* para realizar una API que se encargase de gestionar *endpoints* y controlar el flujo de los datos que ingresan y salen de la base de datos mediante *requests*. Comprendió una buena elección para la solución implementada pues es rápida, y no posee un nivel de complejidad elevado.
- **Python:** Corresponde a uno de los lenguajes de programación mas utilizados en la industria del desarrollo de software, por lo que si lo que se desea es mantenibilidad del código, *Python* es una buena opción a utilizar. Otra de las razones por las que se decidió utilizar este lenguaje fue el uso del *framework FastAPI*. Así también, para manipular el escaneo de señales de red, y organizar la data se decidió utilizar *Python* pues de esta forma se utilizaba un solo lenguaje de programación para el *backend*.
- **SQLAlchemy:** Una de las librerías más utilizadas en este proyecto fue *SQLAlchemy*, pues consistió en la principal herramienta para manipular, crear y eliminar bases de datos de manera simple.
- **SQLite:** Se decidió utilizar SQLite como gestor de bases de datos debido a que se posee poco espacio de almacenamiento y poca memoria RAM, y a que por otro lado, esta aplicación no será de uso masivo pues el servidor siempre se ejecutará en el dispositivo del usuario.
- **Airodump-ng**: La labor de "*airodump-ng*" (parte de "*aircrack-ng*"), corresponde a escanear la red para obtener la potencia de las señales de los dispositivos cercanos mediante el uso de modo monitor. Se usa "*airodump-ng*" debido a la comodidad que ofrece a la hora de conseguir señales por medio de modo monitor.
- **Shell script:** *Shell script* fue utilizado para automatizar y gestionar tareas que no se podían ejecutar ni gestionar por medio de *scripts* de *Python* o *Javascript*, tales como el escaneo de red.
- **Requests:** Esta librería permite realizar solicitudes GET y POST y PUT, de forma directa. Es por eso que para realizar peticiones HTTP por medio de Python al enviarse la instrucción de escanear las redes y enviar esta data a las bases de datos, se decidió utilizar la librería *Requests*.
- **Uvicorn:** Para desplegar el servidor *backend* de esta aplicación fue necesario utilizar el servidor de aplicaciones web *Uvicorn*, que se caracteriza por funcionar junto al *framework FastAPI*.

La totalidad de las aplicaciones utilizadas en el desarrollo comprenden pilares fundamentales, pues permitieron conseguir la mayoría de los objetivos planteados inicialmente.

4.2.1. Restricciones a considerar

La solución considera ciertas restricciones para su funcionamiento, por lo que se hacen ciertas suposiciones para la solución del problema:

- **El dispositivo IoT que se usa para ciber-acecho utiliza el protocolo WiFi para conectarse a internet:** Los dispositivos IoT pueden comunicarse por distintos canales de comunicación, como Bluetooth, 5G, WiFi, entre otros. En este caso solo se abarca el problema de la detección de dispositivos IoT que se comuniquen mediante WiFi.
- **El atacante no altera la MAC de sus dispositivos:** Las personas que utilizan dispositivos IoT para el espionaje indeseado en algunas ocasiones pueden tener conocimientos como para esconder la MAC de su dispositivo. Para efectos de este trabajo, considerando que es muy poco común que alguien altere la MAC de los dispositivos espías, supondremos que dicha modificación no ocurre. Es por ello que se decide evitar hacer un estudio del comportamiento de las señales de los dispositivos, pues se desea utilizar alguna de las API's de bases de datos disponibles con la información de las MAC de los distintos dispositivos.

4.3. Configuración inicial

En primer lugar resulta necesario configurar el entorno de trabajo y las configuraciones para éste. Se propone crear una aplicación que se encuentre configurada en una imagen de disco para una *Raspberry*, por lo que la aplicación se desarrolló en una *RaspberryPi3b+*. Las configuraciones iniciales involucraron la instalación del sistema operativo *Kali* (que se llevó a cabo con la aplicación *Balena Etcher*, pues *RPi Imager* no realiza correctamente la instalación de *Kali Linux*). Una vez instalado el sistema operativo son instaladas todas las dependencias a utilizar (*python, pip, npm, react, fast-api, sqlalchemy, plotly,* entre otros).

4.4. Estructura de la aplicación

La aplicación sigue el flujo mostrado en la figura 4.2:

Figura 4.2: Flujo de datos

Donde una forma más específica de definir el flujo de datos y tecnologías involucradas se puede ver a continuación:

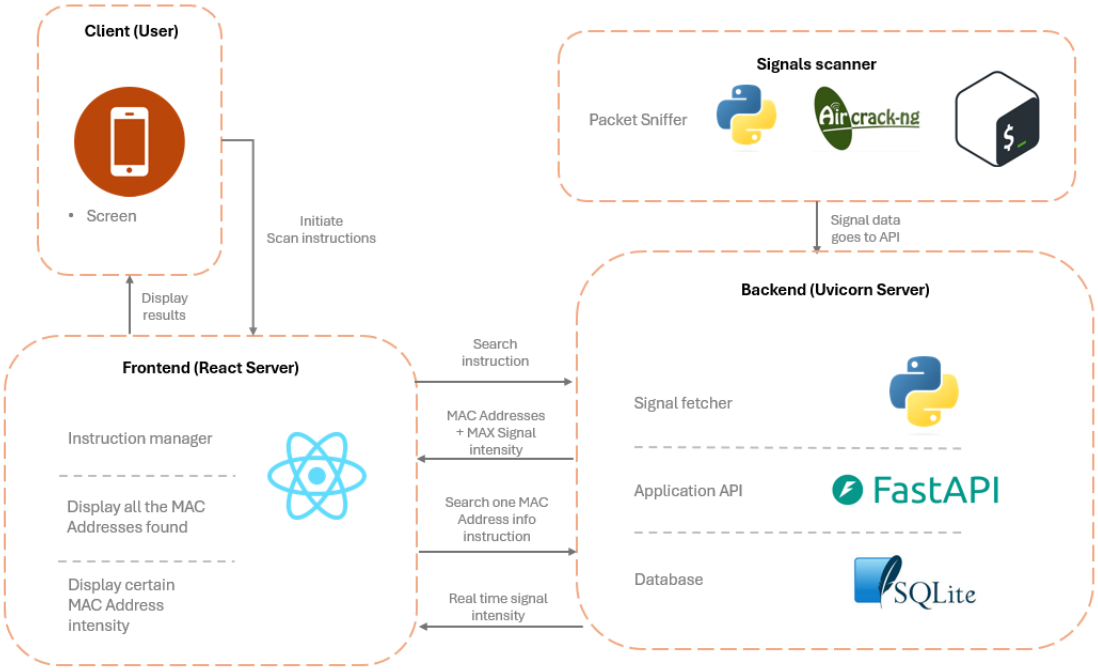

Figura 4.3: Estructura de la aplicación con todas las partes que la conforman

En la Figura 4.3 se muestra la arquitectura general de la aplicación. Esta consta de un *frontend*, implementado con un servidor de *React* al cual los clientes se conectan a través de sus dispositivos móviles, y un *backend* que se implementa en un servidor de *Uvicorn*. Desde su dispositivo móvil, el cliente envía instrucciones y solicitudes al servidor *Uvicorn* mediante *endpoints* para obtener las *MAC Address* de los dispositivos cercanos detectados y las intensidades de señal máxima reportadas.

Paralelamente, cuando el usuario envía la instrucción de comenzar el escaneo, se levanta una instancia de *airodump-ng* (una herramienta para realizar detección de paquetes de red), de la cual un *script* de *Python*, llamado por uno de los *endpoints* generados en el servidor de *FastAPI*, extrae la data y la envía a una base de datos de señales.

Estos datos se almacenan y se acceden a través de un *endpoint* desde el *frontend*, donde finalmente se presentan al usuario en forma de tabla para que pueda seleccionar qué dispositivo desea buscar. Una vez que el usuario elige un dispositivo, se muestra un gráfico indicador que representa la intensidad de la señal. En este gráfico, una señal más alta se representa con una aguja tendiendo hacia la derecha. La información mostrada en el gráfico se obtiene mediante el mismo proceso de escaneo y la conexión a un *websocket* para enviar los datos en tiempo real.

Cuando el usuario finaliza el proceso desde su dispositivo, se terminan los procesos de localización y detección de señales.

4.5. Módulo de detección de dispositivos

El trabajo se enfoca en la detección de dispositivos IoT, abordando el problema mediante escaneo de la red, específicamente en el protocolo 802.11[3]. Se exploran dos modos de operación específicos de los chips de red: el modo "*Managed*" y el modo "*Monitor*", que permiten realizar análisis de red.

El modo "*Managed*" está disponible en todos los dispositivos con conexión a internet y se utiliza para conectar un dispositivo a un punto de acceso[4]. En este modo, solo se detectan los paquetes de los dispositivos conectados al mismo punto de acceso durante el escaneo. Por otro lado, el modo "*Monitor*" permite capturar paquetes cercanos sin necesidad de estar conectado a un punto de acceso, pero a diferencia del modo "*Managed*", en este modo no es posible ver el contenido completo de los paquetes, solo sus cabeceras.

Dado que se quiere hacer un escaneo general de la red sin tener conocimientos sobre cuales son las conexiones de los distintos dispositivos, resulta necesario hacer un escaneo de la red sin un punto de acceso. Para ello, se toma una de las ideas planteadas en el artículo de "*Lumos*"[6] que consiste en utilizar modo monitor para detectar paquetes que se están transmitiendo en los alrededores. Para lograr esto, fue necesario utilizar el analizador de paquetes de red "*airodump-ng*"[5].

[3]Protocolo de comunicación WiFi.

[4]Se refiere como punto de acceso a dispositivos que permiten la interconexión entre computadores, tales como enrutadores.

[5]Herramienta de la librería "*aircrack-ng*" que entrega datos de señales captadas mediante modo monitor.

4.5.1. Configuración del dispositivo para el escaneo de red

Para el correcto funcionamiento del módulo de detección de dispositivos se tuvieron que cumplir ciertos requisitos. En primer lugar fue necesario contar con un chip de red que soportase modo monitor y de esta manera escanear la red sin un punto de acceso (para este caso se utiliza un chip Realtek RTL882bu). Ya habiendo conseguido el chip debieron instalarse los drivers para que el chip pudiese funcionar. Adicionalmente fue necesario contar con ciertos paquetes tales como "*airodump-ng*" para poder realizar ejecuciones en modo monitor.

4.5.2. Estructura del módulo de escaneo de red

Este módulo consta básicamente de cuatro archivos, donde `scan_manager.py` se encarga de gestionar la mayoría de las cosas que ocurren para realizar el escaneo de red, como se muestra en la figura 4.4.

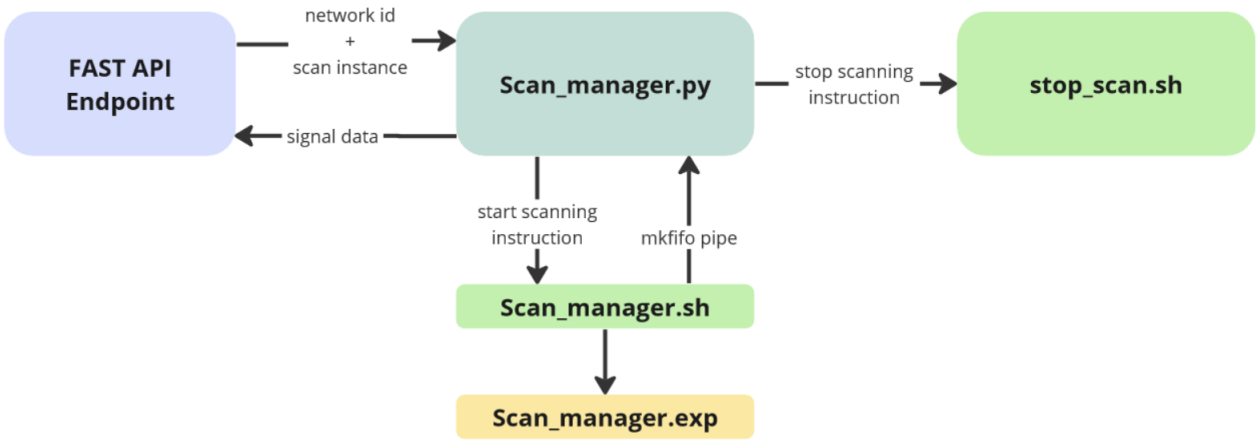

Figura 4.4: Estructura módulo de escaneo de red

La tarea comienza cuando se recibe una señal desde el *frontend* en un *endpoint* de *FastAPI*. Este *endpoint* envía la instrucción junto con un identificador (ID) que indica el tipo de escaneo que se va a realizar: 0 para el escaneo destinado a identificar direcciones *MAC* presentes y 1 para escanear las intensidades de señal de un dispositivo específico. Esta señal se envía a '`scan_manager.py`' para iniciar el proceso de escaneo. La función de inicio de escaneo llama a '`scan_manager.sh`', que a su vez inicia una instancia de '`airodump-ng`' (utilizando '`scan_manager.exp`') en paralelo con un capturador de señales. Luego, la data capturada se envía a la base de datos correspondiente, por medio de *endpoints* de *FastAPI*.

Cuando se desea finalizar el escaneo, el frontend envía una señal indicando que el proceso debe detenerse. Esta instrucción se recibe en '`scan_manager.py`', que a su vez llama a '`stop_scan.sh`', un '`shell_script`' encargado de finalizar las ejecuciones y realizar procedimientos acordes al ID indicado para el escaneo realizado.

4.5.3. Funcionamiento:

Como se mencionó anteriormente, `scan_manager.py` cumple el rol de manejar la data que entra y sale del endpoint de FastAPI. Para ello implementa tres funciones importantes:

- **Comenzar escaneo:** Se llama al *shell-script*[6] `scan_manager.sh`, el cual comienza en segundo plano con la ayuda del comando *tmux*[7], una instancia de `scan_manager.exp`. El objetivo de este archivo *expect*[8] es crear un entorno de *airodump-ng* con instrucciones automatizadas, pues debido a que *airodump-ng* es un comando que funciona con una consola interactiva, se requiere simular una entrada estándar. Adicionalmente, este archivo *expect* cumple con la labor de redirigir todo lo que entregue airodump a un `mkfifo pipe`[9].
- **Obtener señales:** Paralelamente a la inicialización de la instancia de *airodump-ng*, `scan_manager.sh` inicia en segundo plano, un *getter* que se encarga de leer el `pipe mkfifo` y hacer *parse*[10] a la data que encuentra en el `pipe`. Finalmente la data obtenida en el `pipe`, ya formateada, es enviada por un *HTTP request* a la *API*.
- **Finalizar escaneo:** Cuando se desee finalizar el escaneo, se enviará una petición desde el *frontend* que llamará a `stop_scan.sh`. Este *shell-script* se encargara de terminar el escaneo y la obtención de datos, y luego, limpia el entorno utilizado.

[6]Guión de *shell* utilizado para automatizar tareas.

[7]Generador de sesiones en una misma terminal.

[8]Librería de Linux que permite ingresar *inputs* en una terminal, mediante un *script*.

[9]Estructura de tipo FIFO que envía data desde la terminal, y luego escribe la información obtenida donde se le solicite.

[10]Identificar la data en base a una estructura dada.

4.5.4. Tipos de datos recolectados

Airodump-ng recolecta diversos datos en un escaneo de red, tales como BSSID, CH, PWR, Lost, STATION, Packets, etc. Para efectos del desarrollo de esta aplicación son de interés los siguientes campos:

Campo	Uso
STATION	MAC Address de un cliente detectado, permite saber quien es el manufacturador del elemento que se detectó, debido al OUI (Organizational Unique Identifier, que corresponde a los primeros 24 bits de la MAC Adress que indican quien fabrica el chip de red detectado) que se encuentra presente en la MAC.
PWR	Generalmente corresponde al RSSI. Muestra la intensidad de la señal del STATION encontrado. Esto permite saber que tan lejos se encuentra el STATION. Los valores obtenidos en PWR tienen los siguientes significados: • PWR $= -1$: La intensidad de señal no pudo ser reconocida debido a que se encuentra fuera de rango • PWR ≥ -40: Señal fuerte • $-55 <$ PWR < -40: Señal medianamente fuerte. • PWR $= 55$: Señal promedio • $-70 <$ PWR < -55: Señal medianamente débil. • PWR ≤ -70: Señal débil
CH	Corresponde al canal donde se está escaneando la red. Actualmente la aplicación está configurada para que se recorran todos los canales

Tabla 4.1: Campos importantes obtenidos con airodump

Cabe destacar que las comunicaciones en redes de tipo 802.11 se suelen realizar en distintos "canales de comunicación", lo que requiere que el proceso de recolección salte constantemente entre canales para buscar dispositivos emitiendo o recibiendo paquetes en los mismos (a pesar de que esto lamentablemente sacrifique un poco de precisión).

4.6. Manejo de información

Como se mencionó anteriormente, se llevan a cabo dos instancias de escaneo distintas. La primera instancia está enfocada en la detección de dispositivos cercanos, utilizando el modo monitor en una *Raspberry Pi*. Los datos almacenados para este escaneo incluyen `'network_scan_id'`, `'station'`, `'pwr'` y `'detected_at'`.

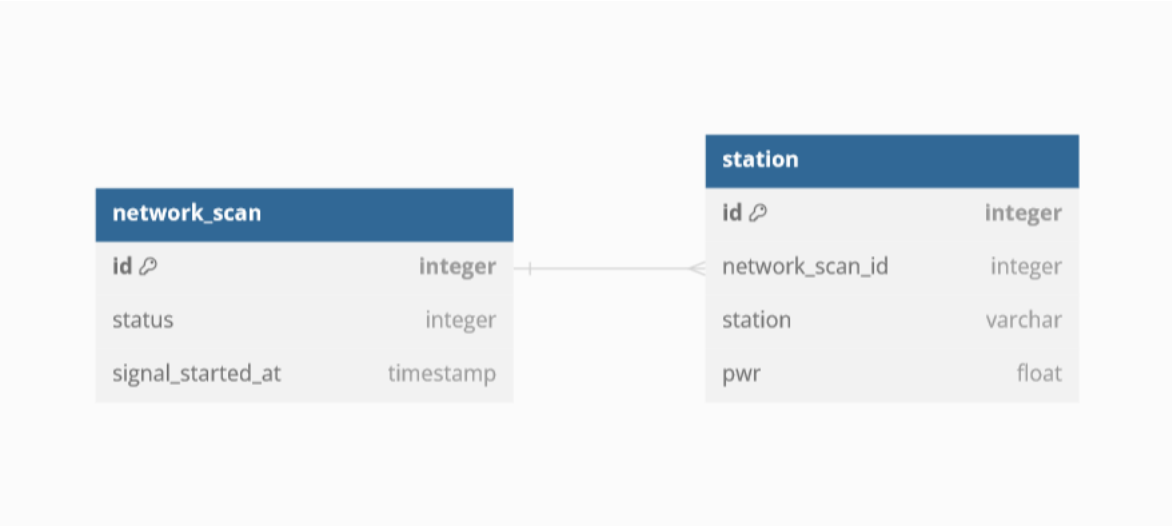

Figura 4.5: *Schema* de bases de datos utilizadas

En este contexto, el '`pwr`' almacenado, representa la mayor intensidad de señal encontrada para cada dispositivo '`station`' detectado. Esta es la data que se almacena en una base de datos. Gráficamente, el flujo de datos para este primer escaneo sigue la forma presentada en la figura 4.6

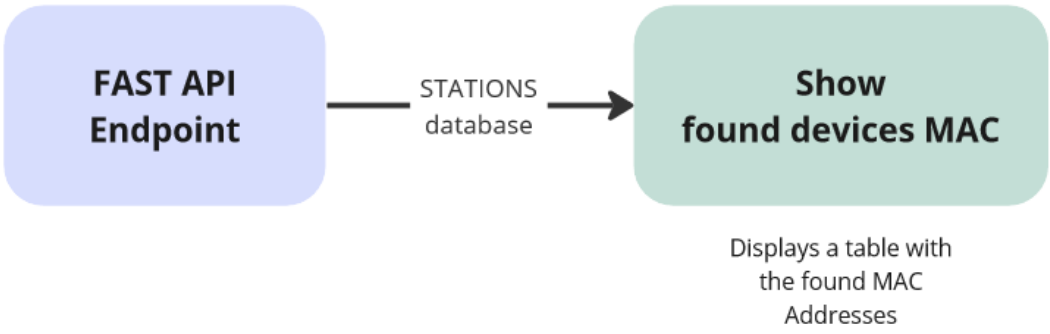

Figura 4.6: Flujo de información para el despliegue de `stations` detectadas

Por otro lado, la segunda instancia de escaneo se centra en encontrar las intensidades de señal (‘`pwr`’) de un dispositivo en particular, para mostrar qué tan cerca se encuentra el dispositivo sospechoso del usuario. La data capturada por `scan_manager.py` en esta segunda instancia debe ser accedida inmediatamente y desplegada en tiempo real, por lo que se utilizan *websockets* para evitar ingresarla a una base de datos, y pasarla directamente al *frontend* mediante *endpoints* de *FastAPI*.

Figura 4.7: Flujo de la información para manejar la data que se despliega al usuario

4.7. Módulo de despliegue de información

Este módulo se encarga del despliegue de la información procesada. Su función principal es tomar los datos procesados y mostrarlos en una tabla, seguida de un gráfico de indicadores que muestra la intensidad de las distintas señales encontradas.

Primer escaneo:

El primer escaneo, como se menciona anteriormente, recopila los datos de una base de datos y los muestra en una tabla. En esta tabla, se enumeran todos los ‘`station`’ junto con sus respectivas intensidades de señal máximas. Esta tabla permite al usuario seleccionar un ‘`station`’ específico para proceder con el segundo escaneo, enfocado en este mismo ‘`station`’ seleccionado.

Los dispositivos desplegados corresponden a los ‘`station`’ que registraron intensidades de señal consideradas “altas”, esto es, los dispositivos que posean PWR es mayor a -60. Se toma esta decisión en base a la información extraída de la Tabla 4.1, donde se determina que las intensidad de señal consideradas “bajas” corresponden a los PWR menores a -55.

Segundo escaneo:

Una vez es recibida la data a partir del *WebSocket*, la intensidad de señal es mostrada en un gráfico de indicadores. Éste gráfico de indicadores oscila entre los valores -110 y -1, y permite a los usuarios de la aplicación determinar que tan intensa es la señal que está siendo captada, donde la agujeta del gráfico se ubica en la zona verde para señales más intensas, y en la zona roja para señales más débiles.

Para la realización del gráfico, se utiliza la librería de npm `react-d3-speedometer`, la cual facilita la creación de gráficos de indicadores de manera sencilla y clara.

4.8. Configuración para el uso de la aplicación

Como uno de los objetivos menores mencionados se estipula la creación de una imagen de la Raspberry para una configuración rápida y directa, lo que se hace mediante la herramienta de Windows, *Win32 Disk Imager*. Esto permite que el usuario solo deba montar esta imagen en una tarjeta SD, para poder comenzar a utilizar la aplicación. Luego, tal como se mencionó anteriormente, se deberán instalar los drivers correspondientes a el chip de red que se utilice para realizar capturas de paquetes.

Una vez montada la imagen en la tarjeta SD, se puede proceder a hacer las configuraciones finales para concretar el escaneo. En primer lugar se deberá levantar un hotspot WiFi siguiendo las instrucciones presentadas en el repositorio del proyecto al pie de la letra, pues de otra forma, la conexion de dispositivos móviles al *hotspot* podría no funcionar. Esto pues se solicitan niveles elevados de autentificación, que varios dispositivos Android y iOS, no pueden entregar. Adicionalmente, con esta configuración se establece la dirección IPV4 a `10.42.0.1`, que corresponde a la dirección fijada por Caddy para levantar el frontend y acceder al backend por medio de un *reverse-proxy*.

Finalmente se deberá clonar el repositorio en el directorio `/home/kali`, y se deberá ejecutar el comando `uvicorn main:app -reload -host 0.0.0.0` para desplegar el backend. Caddy se encarga por su lado de mostrar el frontend sin la necesidad de tomar alguna medida extra.

Capítulo 5

Evaluación y resultados

5.1. Testing

Debido a los problemas presentados en el transcurso de la realización de la aplicación, el *testing* se redujo en general a realizar diversas pruebas de funcionamiento para que la aplicación funcionase sin bugs. Para ello, se establecen variables globales que permiten fijar si la aplicación se encuentra en fase de desarrollo o en fase de producción. Los *test* realizados se ven comprendidos por lo siguiente:

- **Tests unitarios:**
  - **Envío de data *null* a la base de datos:** En caso de que llegase data *null* o fuera de los límites aceptados a la base de datos, el servidor debe responder con un *HTTP status response code* 422 (*Unprocesable Content*), pues es importante asegurarse que no se envie data *null*
  - **Primer *network_ scan* enviado:** *network_ scan* es el encargado de gestionar las sesiones de escaneo, donde cada escaneo tiene su propio `id`. Esta id es obtenida para asignársela a cada elemento que forme parte del escaneo (`stations`). El problema natural a pensar es que sucede cuando se realiza el primer escaneo. En este caso la aplicación gestiona de manera adecuada para que se utilice un `id` 0 y se testea para asegurarse que esto siempre se lleve a cabo.
- **Capacidad de la *Raspberry*** Se deja corriendo el escaneo para ver si los recursos de la Raspberry se ven consumidos eventualmente. Para una Raspberry3b+, cuyas especificaciones consisten en un quad-core de 64 bits con 1,4 GHz de procesador 1GB de memoria RAM y 64GB de almacenamiento en una SD, el programa se encontró corriendo por 20 minutos (el doble de lo esperado en una sesión normal), experimentando solamente retrasos leves en algunos *requests* enviados por el dispositivo móvil.
- **Comportamientos inesperados:** La unica acción que el usuario puede realizar dentro de la aplicación es presionar el botón de iniciar y el botón de terminar scan, por que el único problema por comportamiento inesperado del usuario podría derivar de esa acción. Por ello se prueba este caso, donde se cae en cuenta que como se usa el comando

tmux para poder correr instancias en paralelo, si el servidor no se levanta con permisos de administrador, entonces las pestañas *tmux* no se pueden cerrar y por ende quedan eternamente corriendo. De ello surge una consideración a la hora de correr la aplicación, que es, realizarlo con permisos de administrador.

- **Testeo de compatibilidad con distintos dispositivos:** La aplicación se testea con distintos navegadores y celulares, en primer lugar se experimenta con un *Samsung Galaxy A50, un Xiaomi Redmi* 9 y un *Samsung Note 20 Ultra*, sin presentar ningún problema de compatibilidad en los navegadores *Mozilla*, *Google Chrome* y *Opera*, ni evidenciar problema en ninguno de estos navegadores.
- **Configuración de la aplicación desde cero:** Uno de los objetivos consistió en que el tiempo de configuración de la aplicación fuese acotado. Para ello se toma la imagen de disco, y se escribe en una SD mediante la herramienta *WinDisk Imager*. Esto tardó aproximadamente el tiempo de descarga de la imagen y el tiempo de montar la imagen en la SD a utilizar en la Raspberry. Una vez hecho esto, basta con seguir las instrucciones del `README.md` de la aplicación, para ejecutar esta como se ejecuta normalmente, lo cual en total se vio conformado por aproximadamente 2 horas de configuración.

5.2. Resultados

Se entrega una aplicación capaz de detectar señales de dispositivos IoT cercanos, y mostrar su cercanía mediante las intensidades de señal captadas, para que el usuario pueda saber donde buscar.

El uso de la aplicación se requiere seguir una serie de pasos. En primer lugar, se deberá establecer una conexión al punto de acceso emitido por la *Raspberry*, mediante el dispositivo móvil del usuario. Al encontrarse estos dos dispositivos conectados, es posible acceder a la aplicación levantada por la *Raspberry Pi* localmente. Dada la configuración preestablecida con *Caddy*, para acceder, se deberá ingresar la dirección IP `10.42.0.1` para acceder a esta. Una vez dentro, se advierte que la conexión no es segura, pues no posee certificados. Como la aplicación se despliega localmente esto no supone problemas de seguridad, por lo que se deberá presionar el botón de "*Aceptar el riesgo y continuar*".

Una vez dentro de la aplicación es posible ver una página simple con un botón que da inicio al escaneo de señales. En la figura 5.1 es posible apreciar una visualización de lo mencionado.

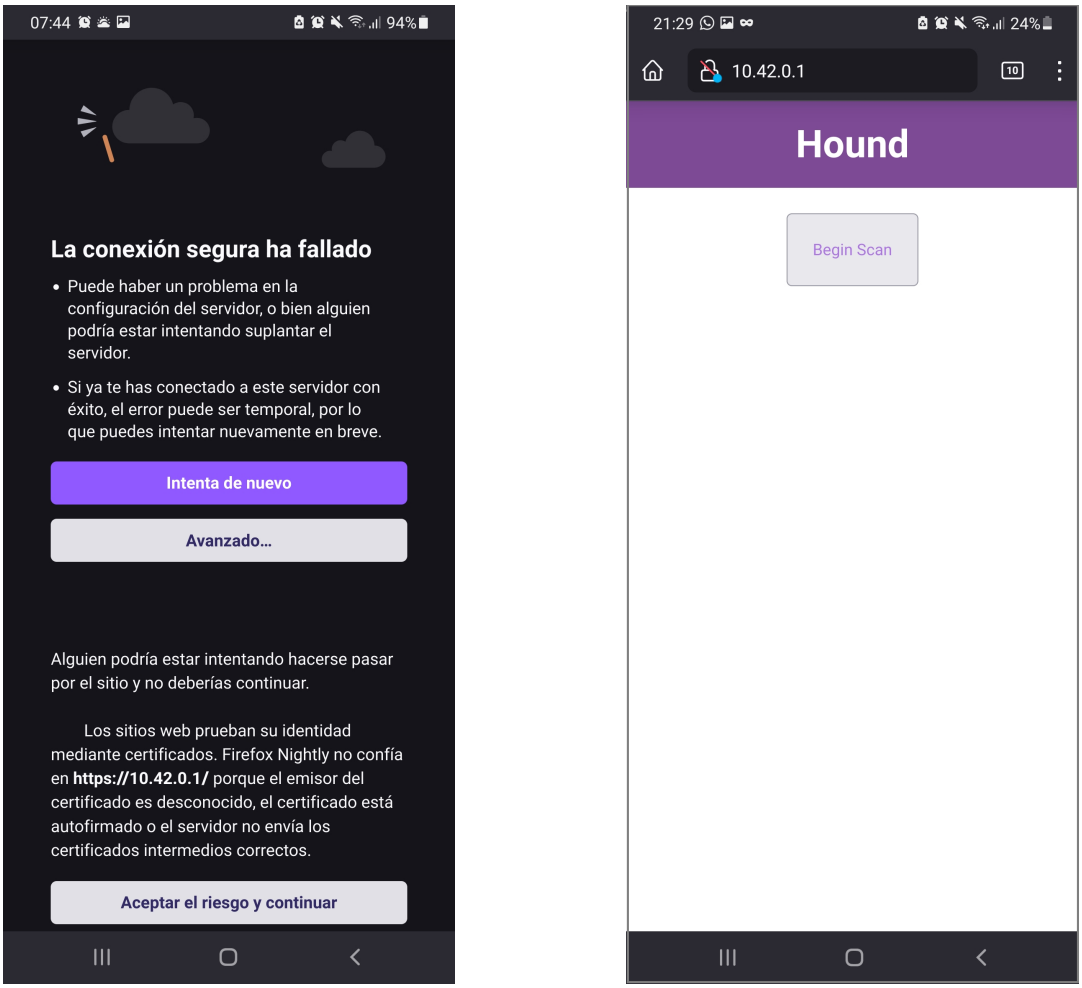

Figura 5.1: Comienzo del escaneo

Cuando el botón *Begin Scan* es presionado, se envía una instrucción para iniciar un escaneo de red, y posteriormente, detectar todas las direcciones MAC de los dispositivos cercanos al usuario. Una vez que el usuario siente que ha recorrido toda la habitación, debe finalizar el escaneo seleccionando el botón *Stop Scan*.

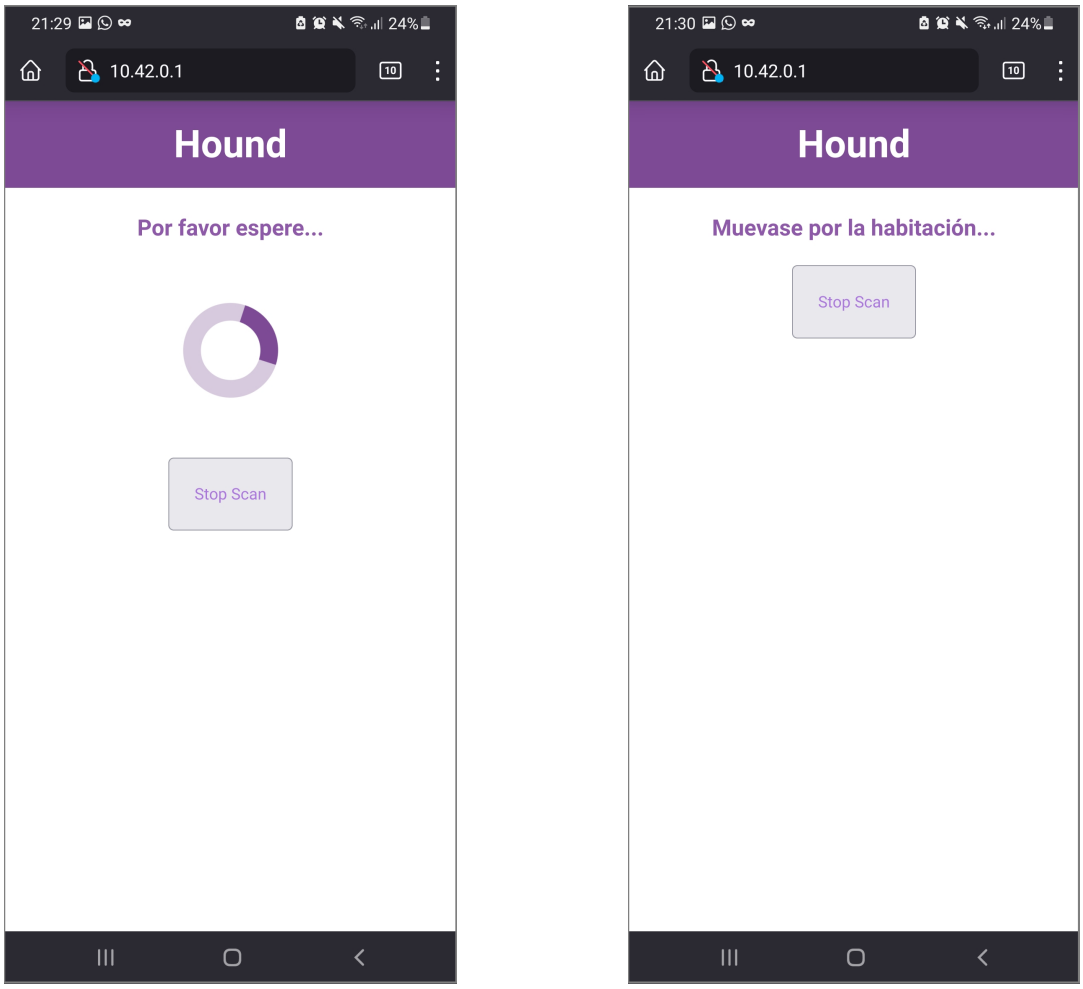

Figura 5.2: Aplicación en dos instantes distintos (1) Espera para configurar módulo de búsqueda de paquetes. (2) Instrucción para recorrer habitación

Una vez hecho esto, se muestra una tabla con todas las direcciones *MAC* y sus respectivas intensidades de señal. El usuario debe seleccionar una *MAC* de la tabla y se iniciará una segunda instancia de escaneo.

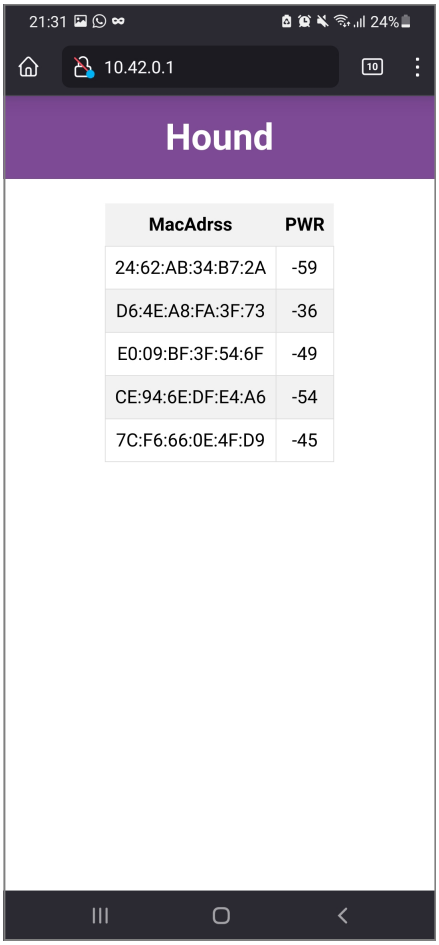

Figura 5.3: Tabla de dispositivos encontrados con sus intensidades de señal máximas

En esta nueva instancia de escaneo, solo se muestran las señales relacionadas al dispositivo cuya dirección MAC fue seleccionada. El gráfico desplegado muestra la intensidad de señal y utiliza colores para indicar qué tan intensa es la señal detectada:

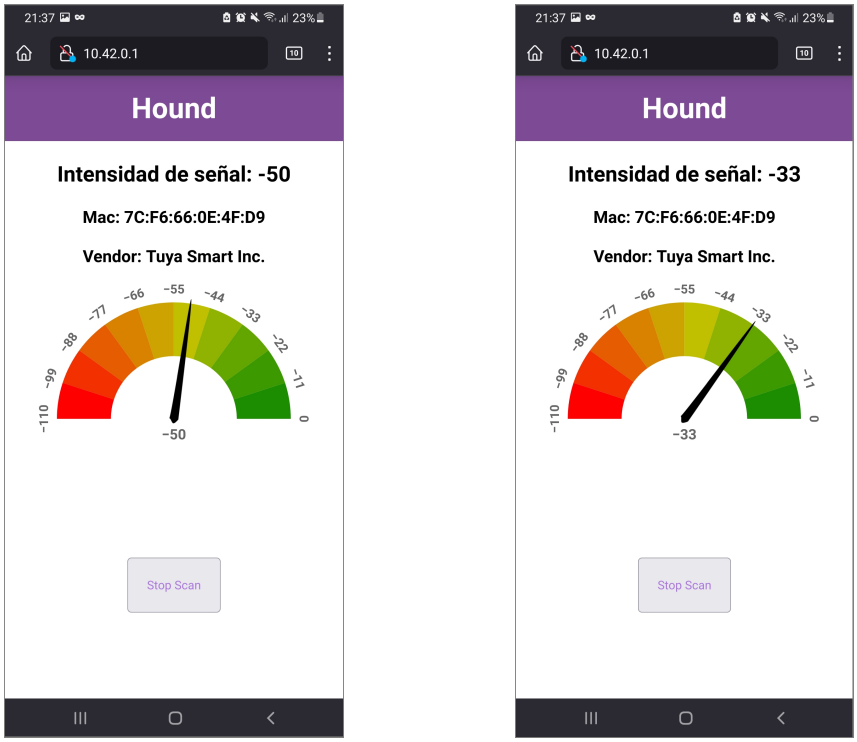

Figura 5.4: Gráfico de indicadores para desplegar MAC en la aplicación

A continuación se presenta un diagrama de flujo, para mostrar el flujo del uso de la aplicación:

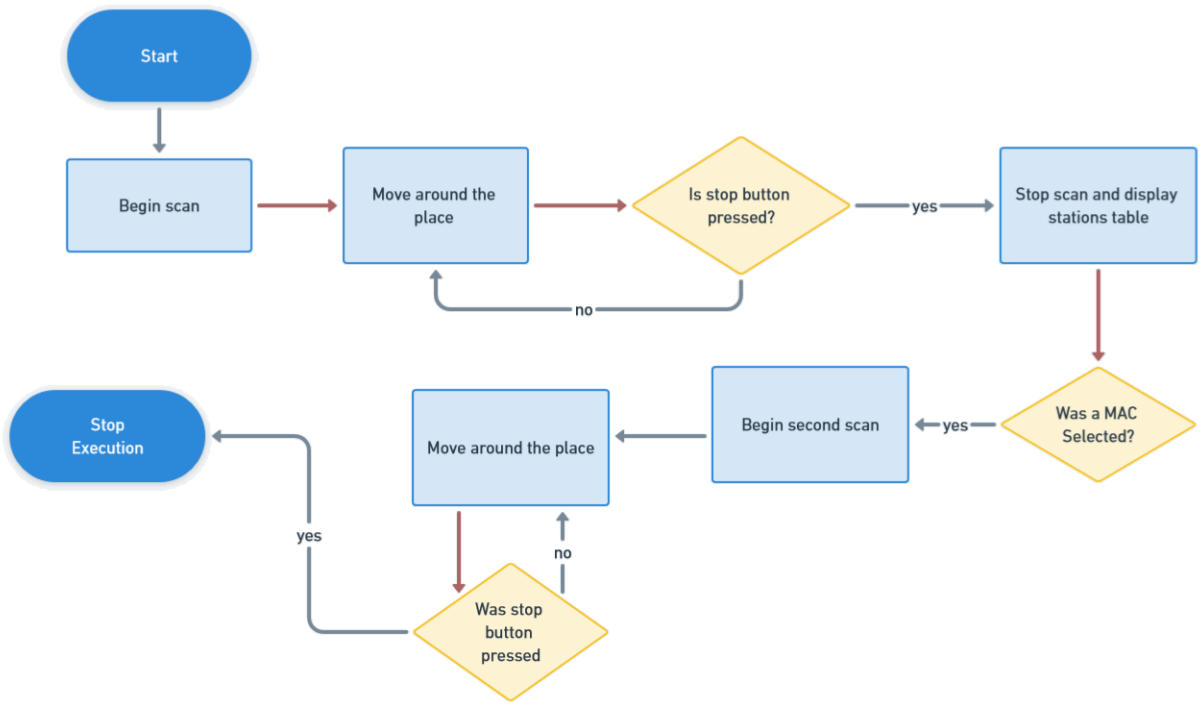

Figura 5.5: Flujo de uso de la aplicación

Capítulo 6

Discusión

6.1. Análisis de resultados

La fiabilidad de los resultados depende de que tan estacionario se deja el dispositivo en cada paso. Dado que el escaneo de señales se realiza haciendo saltos entre los distintos canales, para hacer detecciones precisas, hay que esperar que en cada paso la aplicación recorra todos los canales, lo cual tarda aproximadamente un segundo.

En base a distintas ejecuciones realizadas se notan ciertas tendencias en las señales de los dispositivos. Un dispositivo que se encuentra exactamente al lado de el chip de red de la *Raspberry*, oscila en un rango de intensidad de señal entre -10 y -35 en general. Un dispositivo que se encuentra a un metro aproximadamente de la *Raspberry*, posee una intensidad de señal de -40 aproximadamente. Un dispositivo que posee una intensidad de señal de aproximadamente -55 se encuentra entre uno y tres metros del chip de red.

6.2. Consideraciones del desarrollo y decisiones tomadas

Una gran cantidad de tiempo fue utilizada en el entendimiento del código, e intentar extraer los modelos de la aplicación *Lumos*, la cual no se encontraba documentada ni poseía legibilidad. Finalmente el código base no se encontraba completo, ni se encontró ninguno de los modelos de *machine learning* que se mencionaban en el artículo, a tal punto que no se pudo rescatar nada para el desarrollo de la aplicación. El consumo de tiempo que esto supuso, se podría haber evitado de saber que el código disponible no se encontraba completo. A su vez, el que no existiesen elementos que se pudiesen rescatar de la aplicación *Lumos* supuso que no se podrían tomar como guía sus modelos de desarrollo para la aplicación, ni obtener algoritmos de localización del usuario u obtención de datos de señal, lo que implicó a su vez que el alcance del desarrollo de esta aplicación se vería considerablemente reducido. De todas formas, la lectura del artículo resultó ilustradora para tomar decisiones de herramientas, diseño y procedimientos para la detección de dispositivos IoT, tales como el uso de modo monitor para detectar dispositivos, y el uso de un dispositivo como una Raspberry Pi para

el desarrollo de la aplicación.

Como se mencionó anteriormente, inicialmente se desarrolló la aplicación basada en una estimación de posición mediante la integración de la aceleración obtenida por el acelerómetro del celular del usuario. Si se hubiera tenido en cuenta desde el principio la poca confiabilidad del cálculo de la posición basado en la doble integración de la aceleración, se hubiese tomado otra decisión inicial de desarrollo. Finalmente, se decide desarrollar una aplicación que mostrara e interpretara las intensidades de señal para que el usuario pudiera obtener información a partir de ellas, prescindiendo así del uso del acelerómetro y dependiendo únicamente del RSSI[1].

Asimismo, a pesar de que fue posible mostrar el distribuidor de *chipset* mediante la MAC, no fue posible identificar la naturaleza de estos mismos debido a que no se encuentra disponible una base de datos con las funcionalidades de cada *chipset*.

Un último punto importante a considerar es que mientras la aplicación se encontraba en desarrollo, `create-react-app` (también conocida como CRA) fue deprecada, por lo que en el futuro no se encontrará mantenida. Sin embargo, este problema no es irremediable ni altamente complejo de solucionar, pues existen herramientas que permiten migrar las aplicaciones hechas en CRA a otros *frameworks* tales como *Vue* o *NextJS*.

6.3. Impacto del trabajo realizado

Actualmente, en el mercado no existen aplicaciones que permitan visualizar y analizar señales con el objetivo de localizar dispositivos físicamente, basándose en ellas. Las herramientas disponibles suelen tener un alcance de localización bastante limitado. Sin embargo, esta aplicación puede detectar dispositivos espías que se encuentran a distancias más largas. Poder interpretar los valores de señal sin necesidad de estudiar el significado de los valores de PWR, sumado a la capacidad de encontrar la ubicación aproximada de un dispositivo espía dentro de una habitación, resulta valioso para la tarea de detección de estos dispositivos.

Dada la poca disponibilidad de información respecto a la detección y localización de dispositivos IoT, es posible asegurar que la aplicación ofrece un producto funcional que abarca un problema cuya solución no es directa. Por esto, además, definitivamente supone un avance para la investigación. Por otro lado, de esta forma se genera visibilidad sobre la problemática del *cyberstalking* y las medidas que es posible tomar frente a la sospecha de que una persona esté siendo víctima de esta misma.

[1]Indicador de intensidad de la señal recibida

Capítulo 7

Conclusión

En el trabajo realizado en esta memoria, se desarrolló un MVP (Producto mínimo viable), de una aplicación que permite detectar localización de usuarios y señales de WiFi a través de distintos canales, e integrar estos datos de forma que se junte la data obtenida, y luego desplegar la data para que el usuario pueda comprenderla. La aplicación se encuentra disponible en el repositorio de github de esta memoria.[1].

Para utilizar esta aplicación, el usuario deberá recorrer la habitación en la que se encuentra y observar un gráfico de indicadores que le muestra los lugares donde la intensidad de la señal del dispositivo se considera alta.

El análisis inicial se centró en estudiar detenidamente la aplicación *Lumos* para recopilar información sobre los elementos que podrían ser útiles para el proyecto. A pesar de haber realizado un estudio exhaustivo de *Lumos*, no fue posible reutilizar ningún elemento específico de esta aplicación. Sin embargo, se adoptaron algunas ideas de su solución, como el uso de modo monitor.

En cuanto al tiempo de configuración de la aplicación, el objetivo era lograr que tardase un tiempo aceptable (menor a un día). Dado que la configuración tomó aproximadamente dos horas y que todo el proceso de configuración se encuentra documentado en el repositorio del proyecto, se considera que se cumplió el objetivo establecido de hacer una aplicación más fácil de levantar que *Lumos*.

En términos de compatibilidad, la aplicación se puede ejecutar en diversos dispositivos de distintas marcas sin la necesidad de instalar aplicaciones adicionales. Por lo tanto, se concluye que la solución es compatible con la mayoría de los dispositivos utilizados.

Después de encuestar a un grupo pequeño de personas, se llegó a un consenso de que la aplicación tiene un diseño directo y fácil de usar, lo que la hace amigable para el usuario.

[1]https://github.com/hackerlab-uchile/Hound.

De esta forma, se considera que la aplicación es funcional como un producto mínimo viable, ya que muestra la intensidad de las señales en dispositivos cercanos, lo que permite a los usuarios buscar dispositivos potenciales con una precisión confiable, que era el objetivo principal de la aplicación.

7.1. Trabajo futuro

El siguiente paso para el desarrollo de la aplicación implica realizar mejoras. Una posible mejora y sugerencia para trabajos futuros es mostrar rangos de distancia aproximados basados en las intensidades de señal recibidas. Esto podría mejorar la utilidad de la aplicación al proporcionar a los usuarios una estimación de la distancia a la que se encuentran los dispositivos detectados. Otra mejora propuesta para la aplicación consiste en desarrollar algoritmos de inteligencia artificial que permitan obtener la naturaleza de los chips producidos por distintos fabricantes haciendo *scraping*[2] de data de sus sitios web.

Adicionalmente, se propone automatizar el inicio de la aplicación apenas se encienda la *RaspberryPi* para facilitar el uso de la aplicación.

Por otro lado, dado que `create-react-app` se vió deprecada se considera como un trabajo futuro importante para mantener la mantenibilidad migrando la aplicación a *algún* otro *framework* como *Vite* o *NextJS*.

Finalmente, un caso de estudio interesante que se sugiere abordar, surge de la detección de paquetes enviados por dispositivos que se comuniquen por otros protocolos distintos al 802.11, tales como *Bluetooth* o redes 5G.

[2]Extracción de datos leyendo diversos sitios

Bibliografía

[1] Karen M Abrams and Gail Erlick Robinson. Occupational effects of stalking, 2022.

[2] David M. Adamson, Annie Brothers, Sasha Romanosky, Marjory S. Blumenthal, Douglas C. Ligor, Karlyn D. Stanley, Peter Schirmer, and Julia Vidal Verstegui. *Cyberstalking: A Growing Challenge for the U.S. Legal System*. RAND Corporation, Santa Monica, CA, 2023.

[3] Emily J. Hanson and Kristin Finklea. Stalking concerns raised by bluetooth tracking technologies: In brief, 2022.

[4] Karen Haslam and Editor. How to find, block, and disable an unknown airtag moving with you, May 2023.

[5] Helena Pozniak. Inside the fight to rid the world of abusive stalkerware, Oct 2020.

[6] Rahul Anand Sharma, Elahe Soltanaghaei, Anthony Rowe, and Vyas Sekar. Lumos: Identifying and localizing diverse hidden IoT devices in an unfamiliar environment. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 1095–1112, Boston, MA, August 2022. USENIX Association.

[7] Sharon G. Smith, Kathleen C. Basile, and Marcie jo Kresnow. 2016/2017 report on stalking — updated release. In *The National Intimate Partner and Sexual Violence Survey*, Atlanta, Georgia, April 2022. National Center for Injury Prevention and Control, Centers for Disease Control and Prevention.

[8] P. Suresh, J. V. Daniel, V. Parthasarathy, and R. H. Aswathy. A state of the art review on the internet of things (iot) history technology and fields of deployment. *Proc. of Int'l Conference on Science Engineering and Management Research*, 11 2014.

[9] Jennifer Valentino-devries. Hundreds of apps can empower stalkers to track their victims, May 2018.

ANEXOS

Anexo A

Configuración de la aplicación

A.I. Configuración de Caddy en RPi

Para el despliegue de la aplicación sea mediante una conexión segura mediante el uso de Caddy, es posible usar Caddyfile (y es lo que se utiliza en esta ocasión). El CaddyFile utilizado en este caso se configura de la siguiente manera:

```
10.42.0.1{
    # Sirve el frontend estatico en la direccion 10.42.0.1
    root * /home/kali/Hound/frontend/build

    # Redirecciona el puerto del backend a la direccion 10.42.0.1/api
    # por medio de un reverse proxy
    file_server
    handle_path /api/* {
        reverse_proxy localhost:8000
    }
}
```

A.II. Configuración del hotspot WiFi

Esto se hace de la siguiente forma en una terminal de Linux:

```
nmcli con add type wifi ifname wlan1 mode ap con-name <Name_of_conn> ssid
   <name_of_hotspot>
nmcli con modify <Name_of_connection> 802-11-wireless.band bg
nmcli con modify <Name_of_connection> 802-11-wireless.channel 1
nmcli con modify <Name_of_connection> 802-11-wireless-security.key-mgmt
  wpa-psk
nmcli con modify <Name_of_connection> 802-11-wireless-security.psk  <
  password>
nmcli con modify <Name_of_connection> ipv4.addr 10.42.0.1
nmcli con modify <Name_of_connection> ipv4.method shared
nmcli con up <Name_of_connection>
```

A.III. Levantamiento del backend

Para ello se debe utilizar el siguiente comando en una terminal de Linux.

```
uvicorn main:app --reload --host 0.0.0.0
```

Anexo B

Scripts importantes utilizados

Scripts para paralelización de tareas y automatización

A continuación se muestra `scan_manager.sh`

```
#!/bin/sh
echo 'Starting network scan...'

mkfifo signalpipe


## calling the airodump instance and the function to get the data from the
    pipe
uuid="(uuidgen)" && tmux new -d -s "$uuid"
tmux splitw -h -t "${uuid}:0.0"
tmux send-keys -t "${uuid}.0" "expect scan_manager.exp" ENTER
tmux send-keys -t "${uuid}.1" "python -c 'import scan_manager;
   scan_manager.get_scannings()'" ENTER
# tmux a -t "$uuid"
```

A su vez se utiliza `scan_manager.exp` para ingresar inputs de forma que `airodump-ng` entregue data de los STATIONS en vez de la data que entrega por defecto.

```
#!/usr/bin/expect -f


# Run airodump-ng and wait for it to start
spawn bash -c "airodump-ng wlan1 | tee signalpipe monitor.txt"
expect "airodump-ng"


# 'a' to toggle the scanning instance on airodump where they are set on
   the following order: AP+STA; AP+STA+ACK; AP only; STA only
send "a"
sleep 1
send "a"
sleep 1
send "a"
```

```
15
16 set pipe [open "signalpipe" r]
17
18 interact
19
20 }
```

Por último, para entregar la instruccion a la terminal de que los `panes` deben ser cerrados, se hace lo siguiente (en un archivo llamado `stop_scan.exp`)

```
1  #!/bin/sh
2
3  echo 'Stopping scan...'
4
5  tmux send-keys -t "$(uuid).0" C-c
6  sleep 1
7  tmux send-keys -t "$(uuid).0" "exit" ENTER
8  sleep 1
9  tmux send-keys -t "$(uuid).1" C-c
10 sleep 1
11 tmux send-keys -t "$(uuid).1" "exit" ENTER
12
13
14 echo 'Scan stopped'
```

B.I. Parsing de data de señales

La data de señales sale de el pipe mkfifo de una forma muy poco estructurada, por lo que la forma de formatear esta data consiste en encontrar elementos que posean la estructura de

```
1          while i<(len(line)-1) :
2          global is_empty
3          if (line[i] == "(" ):
4              bssid = "(not associated)"
5              i += 15
6
7          if (line[i] == ":" and bssid == ""):
8              temp = line[i-2: i+15]
9              if (not (" " in temp)):
10                 bssid = temp
11                 i += 14
12
13         if (line[i] == ":" and station == "" and bssid != ""):
14             temp = line[i-2: i+15]
15             if ( not (" " in temp)):
16                 station = temp
17                 i += 14
18
19         if (line[i] == "-" and (pwr == "")):
20             if not (" " in line[i+1] ):
21                 pwr = line[i: i+3]
22                 i += 2
```

B.II. Código para hacer base de datos de MAC

Dado que fue necesario hacer una base de datos de MAC y que se obtenía solamente un CSV, fue necesario transformarlo a una base de datos:

```
import pandas as pd
from sqlalchemy import create_engine
import sqlite3 as lite

file = 'mac-vendors-export.csv'
db = 'sqlite:///mac_vendors_formatted.db'

df = pd.read_csv(file)
engine = create_engine(db)
df.to_sql('mac_vendors.db', engine, if_exists='replace', index=False)
```

Donde se hicieron funciones complementarias en el *backend* para poder extraer correctamente la data de la base de datos y mandarla al *frontend*.

Anexo C

Conexiones

C.I. Funcionamiento de conexiones simultáneas

Se muestran las respuestas realizadas por el celular y la obtención de señales mediante modo monitor de forma simultánea

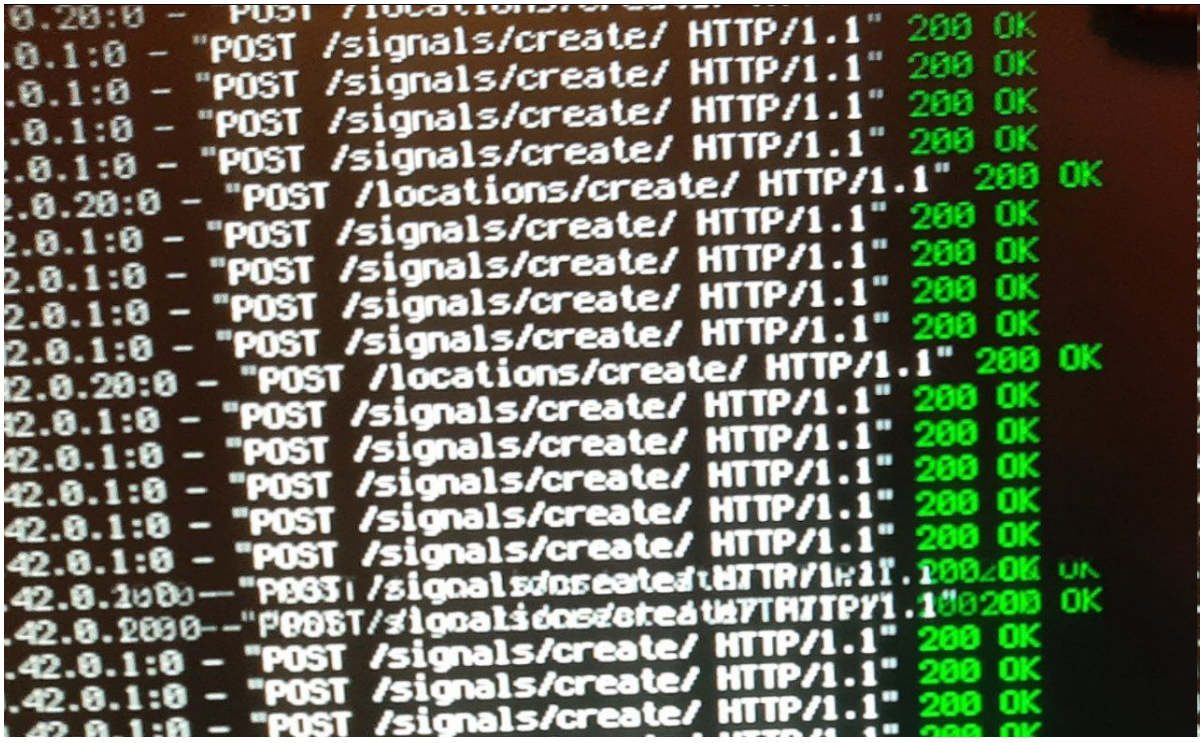

Figura C.1: Envío de data exitoso

C.II. Fetch de paquetes de red

Activando los panes de *tmux* es posible monitorear el envío de data de paquetes de señal.

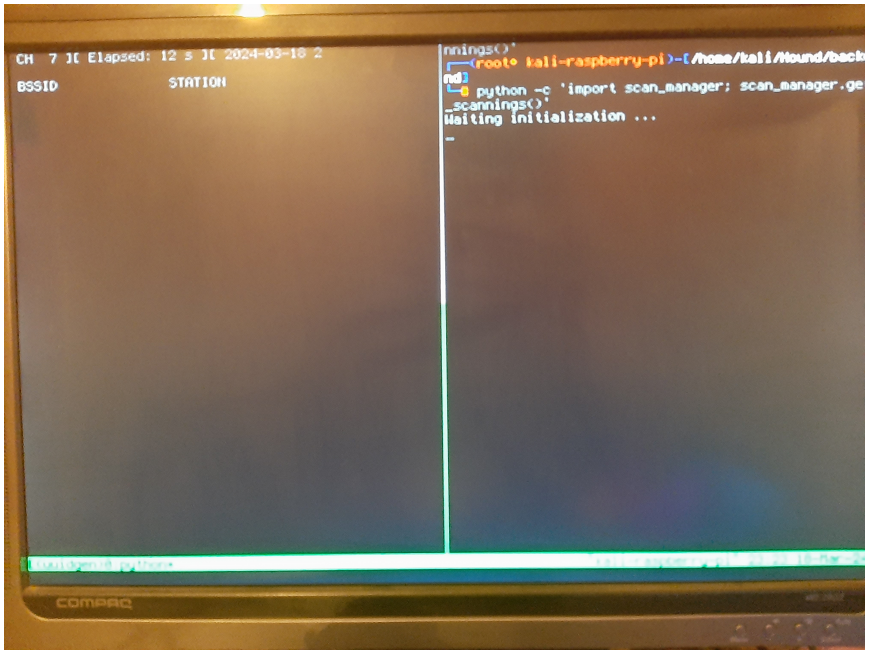

Figura C.2: Inicialización de detección de paquetes de red

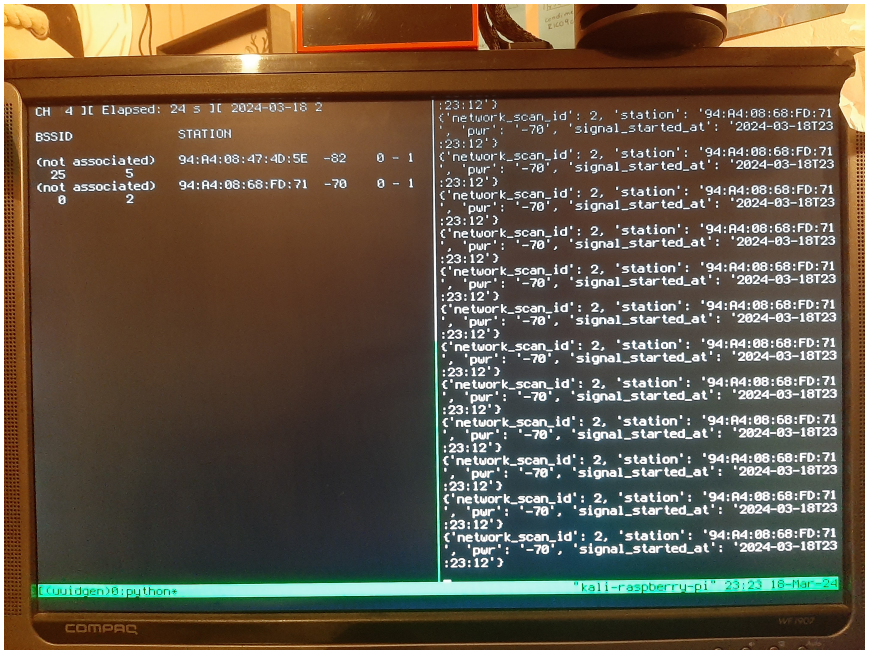

Figura C.3: Proceso de detección de paquetes de red

Anexo D

Mockups

Se desarrollan una serie de *mockups* para aclarar el diseño que debía poseer la aplicación.

Figura D.1: Mockups: (1) Botón para iniciar el escaneo de señales (2) Movimiento de usuario por la habitación

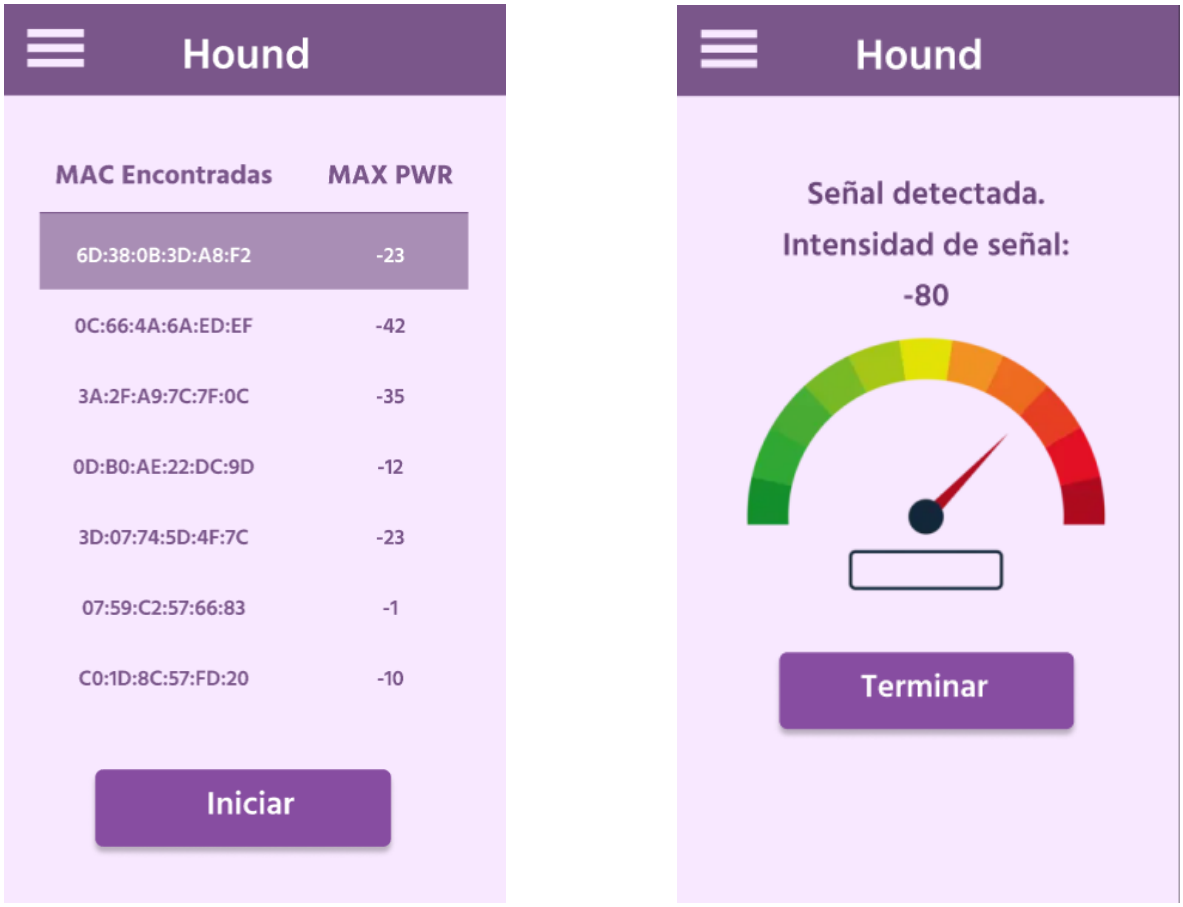

Figura D.2: Mockups: (1) Despliegue de MACS (2)Muestra de señal encontrada