
**Inteligencia de Señales desde diversas fuentes de
radiofrecuencia
Multisource RF SIGINT**



**Trabajo de Fin de Grado
Curso 2023–2024**

Autor
Marcos Alonso Campillo

Directores
José Luis Vázquez Poletti
Juan Carlos Fabero Jiménez

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

Inteligencia de Señales desde diversas fuentes de radiofrecuencia Multisource RF SIGINT

Trabajo de Fin de Grado en Ingeniería Informática

Autor
Marcos Alonso Campillo

Directores
José Luis Vázquez Poletti
Juan Carlos Fabero Jiménez

Convocatoria: *Junio 2024*

Grado en Ingeniería Informática
Facultad de Informática
Universidad Complutense de Madrid

25 de mayo de 2024

Dedicatoria

*A mi familia, amigos y pareja por su apoyo
incondicional.*

Resumen

Inteligencia de Señales desde diversas fuentes de radiofrecuencia

Se presenta una solución innovadora en el campo del Open Source Intelligence (OSINT), centrada en el sistema APRS de reporte automático de paquetes. La herramienta que se ha desarrollado consiste en un sistema diseñado para recopilar, procesar y analizar datos en tiempo real provenientes de emisores de paquetes APRS.

La utilidad del sistema de reporte APRS se extiende a contextos tan diversos que van desde la monitorización de estaciones meteorológicas hasta la gestión de recursos en casos de emergencias y el seguimiento de vehículos. Sin embargo, la complejidad de los datos generados por los emisores APRS y la falta de herramientas especializadas para su análisis dificultan la extracción de información relevante y la identificación de patrones significativos.

La solución que se propone aborda esta necesidad proporcionando una plataforma para la recopilación automatizada de los datos APRS, procesándolos para extraer información relevante y analizándolos para identificar patrones con el objetivo de ayudar a los usuarios a obtener información útil de los datos extraídos.

Palabras clave

OSINT, APRS, Análisis, Datos, Procesamiento.

Abstract

Multisource RF SIGINT

An innovative solution is presented in the field of Open Source Intelligence (OSINT), focusing on the Automatic Packet Reporting System (APRS). The developed tool consists of a system designed to collect, process, and analyze real-time data from APRS packet transmitters.

The utility of the APRS reporting system extends to diverse contexts such as weather station monitoring, emergency resource management, and vehicle tracking. However, the complexity of data generated by APRS transmitters and the lack of specialized tools for analysis hinder the extraction of relevant information and the identification of significant patterns.

The proposed solution tackles this need by providing a platform for the automated collection of APRS data, processing it to extract relevant information, and analyzing it to identify patterns. The goal is to assist users in obtaining useful information from the collected data.

Keywords

OSINT, APRS, Analysis, Data, Processing.

Índice

1. Introducción	1
1.1. ¿Qué es APRS?	1
1.2. Características principales de APRS	1
1.3. Usos principales de APRS	2
1.4. Historia y desarrollo de APRS	3
1.5. Propuesta y objetivos	3
1.5.1. Objetivos	3
1.5.2. Beneficios	4
1.6. Requisitos	4
1.7. Plan de trabajo	5
2. Contexto del problema y estado del arte	7
2.1. Contexto del problema	7
2.1.1. Barreras de entrada	7
2.1.2. <i>Hardware</i>	7
2.1.3. OSINT y APRS	8
2.2. Estado del Arte	9
2.2.1. aprs.fi	9
2.2.2. aprs.to	9
2.2.3. Comparación	10
3. Descripción del Trabajo	11
3.1. APRSINT	11
3.2. Arquitectura de la aplicación	12
3.3. Tecnologías utilizadas y marco teórico	12
3.3.1. Raspberry Pi	12
3.3.2. Disco duro SSD	13
3.3.3. Python	13
3.3.4. Dash	13
3.3.5. Cosmograph JS	13
3.3.6. PostgreSQL	14
3.3.7. SQLAlchemy	15
3.3.8. Aprslib	15
3.3.9. AWS	16
3.3.10. Supervisord	16

3.3.11. Pandas	17
3.3.12. Conda	17
3.3.13. Click	17
3.3.14. Apache Airflow	18
3.3.15. Configuración del proyecto	19
3.4. Adquisición de datos	19
3.4.1. RTL-SDR	19
3.4.2. APRS-IS	20
3.4.3. La trama APRS	21
3.4.4. Recepción de paquetes APRS	23
3.4.5. Procesado de paquetes APRS	23
3.4.6. Almacenamiento de paquetes en la base de datos	25
3.4.7. Base de datos	26
3.4.8. Flujo de un paquete APRS	27
3.5. Visualización y presentación de datos	27
3.5.1. Dash	27
3.5.2. Diseño	29
3.5.3. Home - Primera pantalla	29
3.5.4. Station - Segunda pantalla	32
3.5.5. Graph - Tercera pantalla	33
3.6. Orquestación	35
3.6.1. Supervisord	35
3.6.2. Apache Airflow	36
3.7. Alojamiento	37
3.7.1. Servidor Web	38
3.7.2. Protección ante ataques	40
3.8. Casos de uso	41
4. Conclusiones y trabajo futuro	43
4.1. Conclusiones	43
4.1.1. Éxitos del proyecto	43
4.1.2. Desafíos superados	44
4.1.3. Aprendizajes del proyecto	44
4.1.4. Limitaciones del proyecto	44
4.2. Trabajo futuro	45
Introduction	47
4.3. What is APRS?	47
4.4. Main APRS features	47
4.5. Main APRS applications	48
4.6. History and development of APRS	48
4.7. Proposal and Objectives	49
4.7.1. Objectives	49
4.7.2. Benefits	50
4.8. Requirements	50
4.9. Work plan	50

Conclusions and Future Work	53
4.10. Conclusions	53
4.10.1. Project successes	53
4.10.2. Challenges overcome	53
4.10.3. Project learnings	54
4.10.4. Project limitations	54
4.11. Future work	54
Bibliografia	57

Índice de figuras

1.1. Logo de APRS.	1
1.2. Mapa de frecuencias APRS en el mundo.	2
1.3. Bob Bruninga (creador del sistema APRS).	3
1.4. Plan de trabajo propuesto.	5
2.1. Ejemplo de un RTL-SDR de tipo USB.	8
2.2. Interfaz de aprs.fi	9
2.3. Interfaz de aprs.to	10
3.1. Módulos de la aplicación	11
3.2. Ejemplo de uso de Click.	18
3.3. Infraestructura de la red APRS - APRS-IS.	21
3.4. Formato de trama AX.25.	21
3.5. Procesado de paquetes APRS en AWS Lambda.	24
3.6. Estructura de la base de datos.	26
3.7. Flujo de datos en APRSINT.	28
3.8. Primer diseño de la aplicación web en Figma.	29
3.9. Resultados de la búsqueda de «emergency hospital».	31
3.10. Grafo de estaciones APRS.	34
3.11. Interfaz de Apache Airflow.	37
3.12. Arquitectura del servidor web de APRSINT.	39
4.1. APRS logo.	47
4.2. APRS emission frequency around the globe.	48
4.3. Bob Bruninga (creator of the APRS system).	49
4.4. Proposed work plan.	51

Índice de tablas

3.1. Esquema de orquestación con Apache Airflow.	36
--	----

Capítulo 1

Introducción

“People think of education as something that they can finish.”
— Isaac Asimov

1.1. ¿Qué es APRS?

El APRS o Sistema Automático de Reporte de Paquetes (APRS, por sus siglas en inglés: *Automatic Packet Reporting System*) es un sistema de comunicaciones digitales en tiempo real que permite el intercambio de información entre estaciones de radioaficionados.

De manera general, APRS se utiliza para el rastreo de vehículos, la transmisión de mensajes de texto, la diseminación de información meteorológica y la comunicación en situaciones de emergencia aunque debido a la flexibilidad del protocolo puede ser usado en muchas otras situaciones.



Figura 1.1: Logo de APRS.

1.2. Características principales de APRS

APRS posee varias características que lo hacen útil y versátil en el ámbito de la comunicación de radioaficionados, entre las cuales se encuentran las siguientes:

- **Frecuencia de operación:** APRS opera en la banda de VHF, específicamente en la frecuencia de 144.80 MHz en Europa, aunque en otras regiones del mundo se utilizan frecuencias diferentes, tal como se muestra en la figura Figura 1.2.

- **Modo de transmisión:** APRS utiliza como modo de transmisión paquetes individuales que han de seguir un formato establecido, esto facilita enormemente la adopción e integración de esta tecnología.
- **Actualización de datos en APRS:** En el sistema APRS, un paquete de información se transmite múltiples veces, disminuyendo gradualmente la frecuencia de envío de estas transmisiones a medida que el tiempo avanza. Este método tiene como objetivo maximizar la tasa de recepción de los paquetes [5].

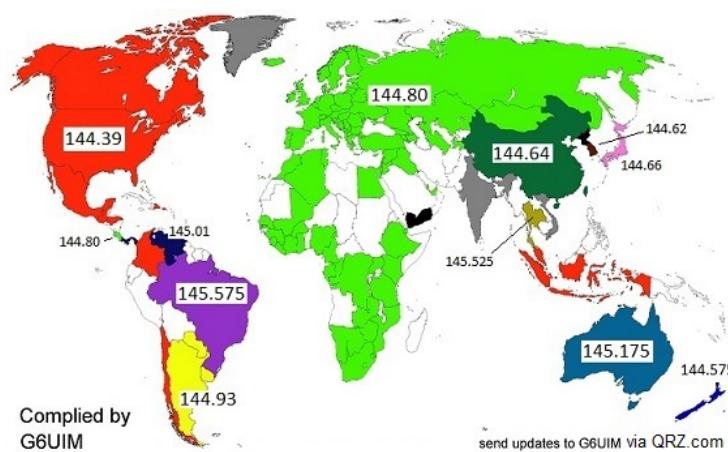


Figura 1.2: Mapa de frecuencias APRS en el mundo.

1.3. Usos principales de APRS

- **Reportes GPS:** APRS permite a los usuarios enviar su ubicación geográfica en forma de coordenadas, obtenida a través de sistemas GPS, lo que facilita el seguimiento de vehículos o personas en tiempo real.
- **Datos meteorológicos:** Las estaciones meteorológicas suelen utilizar mensajes APRS para reportar diferentes datos como temperatura, humedad o presión barométrica, incluyendo información actualizada y útil para diversas aplicaciones.
- **Integración con internet:** Mediante el sistema APRS-IS, los mensajes APRS son accesibles desde internet a través de distintos nodos, ampliando el alcance y la extensión de este sistema.
- **Uso en emergencias:** En situaciones de emergencia, APRS es una herramienta vital para la comunicación de *broadcast* y el seguimiento de recursos y personal.

1.4. Historia y desarrollo de APRS

El sistema APRS nació en la década de los 80 de la mano de Bob Bruninga, un ingeniero que trabajaba en la Academia Naval de los Estados Unidos. Bruninga creó la primera implementación de APRS en un ordenador Apple II en 1982 con el objetivo de mapear informes de posición de la Marina en alta frecuencia. [2]

El primer uso real del APRS fue en 1984, cuando Bruninga desarrolló una versión más avanzada en un VIC-20 para reportar la posición y el estado de los caballos de una carrera de resistencia.

Durante los siguientes años, Bruninga continuó perfeccionando el sistema, al que posteriormente bautizó como Sistema de Tráfico de Emergencia Sin Conexión (CETS, por sus siglas en inglés).

Tras una serie de ejercicios de la Agencia Federal de Gestión de Emergencias (FEMA) usando CETS, el sistema fue trasladado al IBM PC. Durante la década de los 90, CETS (ya conocido como el Sistema de Reporte Automático de Posición o APRS) continuó evolucionando.

A medida que la tecnología GPS se volvía más ampliamente disponible, el término “Posición” se reemplazó por “Paquete” para representar mejor las capacidades más generales del sistema y enfatizar sus usos más allá del mero reporte de posición.



Figura 1.3: Bob Bruninga (creador del sistema APRS).

1.5. Propuesta y objetivos

Se propone una solución completa que incluya la adquisición, procesamiento, visualización y análisis de datos APRS.

1.5.1. Objetivos

El objetivo del proyecto es crear una solución que mejore la experiencia del usuario y enriquezca la información APRS disponible a través de la integración de datos de diversas fuentes abiertas y disponibles.

- **Mejorar la experiencia de usuario:** La solución tendrá una interfaz intuitiva, rápida y útil que facilite la navegación y la interacción con los datos.
- **Análisis avanzado:** La solución contará con herramientas avanzadas para la visualización detallada del tráfico APRS, filtrado preciso y análisis de datos, con el objetivo de extraer inteligencia de los mensajes recibidos.
- **Complementación:** La solución no tiene como objetivo sustituir a aprs.fi, aprs.to u otras plataformas similares, sino ofrecer una alternativa con funcionalidades complementarias que añadan información a los usuarios.
- **Enriquecimiento de información:** Se recopilarán datos provenientes de diversas fuentes abiertas y disponibles para ofrecer una visión más completa del tráfico APRS y sus usuarios, mejorando así la calidad de la información disponible para los usuarios.

1.5.2. Beneficios

- **Mayor comprensión del tráfico APRS:** Los usuarios podrán obtener información más detallada y procesable a partir de los mensajes transmitidos, lo que les permitirá una mayor comprensión del tráfico APRS.
- **Toma de decisiones más efectiva:** El análisis de datos integrado ayudará a los usuarios a tomar decisiones más informadas haciendo uso de la información recibida, mejorando así la eficacia de sus acciones.
- **Flexibilidad:** La opción de auto-alojamiento permitirá a los usuarios tener un mayor control sobre sus datos y privacidad, proporcionando así una mayor flexibilidad en cuanto a su gestión.

1.6. Requisitos

La solución propuesta debe cumplir con una serie de requisitos imprescindibles para asegurar su utilidad y accesibilidad:

- **Bajo costo:** Debe ser una solución económica para garantizar su accesibilidad a una amplia gama de usuarios, incluidos aquellos con presupuestos limitados.
- **Flexibilidad de alojamiento:** Se debe ofrecer la capacidad de auto-alojamiento, los usuarios deben poder optar por alojarla en sus propios servidores o usar la solución en la nube según sus preferencias y requisitos específicos.
- **Extensible:** La solución debe ser extensible, lo que significa que debe permitir la integración de nuevas funcionalidades y la expansión de su capacidad según las necesidades cambiantes de los usuarios.
- **Mejor filtrado:** Se debe ofrecer un filtrado más preciso y flexible en comparación con las plataformas existentes, lo que permitirá a los usuarios obtener

información relevante y útil de los datos APRS para complementar la ya provista por las alternativas.

- **Análisis de datos integrado:** La solución debe incluir un análisis de datos integrado para ayudar a los usuarios a comprender mejor la información recibida a través del APRS y obtener perspectivas y conclusiones valiosas a partir de los mensajes transmitidos.

1.7. Plan de trabajo

Para la realización de este proyecto se ha propuesto el siguiente plan de trabajo, que se divide en las siguientes fases:

- **Fase 1:** Investigación y Análisis de Requisitos.
- **Fase 2:** Diseño y Planificación.
- **Fase 3:** Implementación y Desarrollo.
- **Fase 4:** Realización de la memoria.

En la figura Figura 1.4 se muestra el diagrama de Gantt con las fechas de inicio y fin de cada una de las actividades que se han llevado a cabo a lo largo del proyecto.

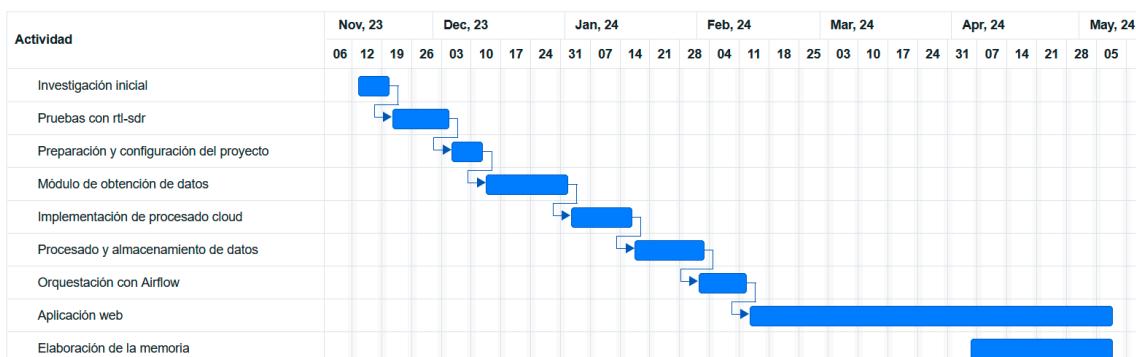


Figura 1.4: Plan de trabajo propuesto.

También se han realizado reuniones periódicas con los directores de TFG para revisar el progreso y discutir los problemas encontrados durante el desarrollo.

Capítulo 2

Contexto del problema y estado del arte

2.1. Contexto del problema

El *Automatic Packet Reporting System* (APRS) como sistema de comunicación digital se basa en la gran comunidad de radioaficionados e incluso tiene aplicaciones industriales como la transmisión de información en tiempo real de vehículos logísticos o estaciones de meteorología. A pesar de su versatilidad y flexibilidad, el APRS presenta algunas barreras de entrada para usuarios que no pertenecen a la comunidad radioaficionada o no tienen sólidos conocimientos de este ámbito. En este capítulo, se analizarán estas barreras y se discutirán los requisitos necesarios.

2.1.1. Barreras de entrada

El APRS requiere conocimientos básicos sobre radiocomunicación y la obtención de una licencia de radioaficionado en caso de querer emitir. La obtención de la licencia a pesar de no implicar un desembolso económico importante, si que requiere un estudio y comprensión de los sistemas y legislación pertinente.

Adicionalmente, el APRS utiliza una estructura de mensaje específico y un conjunto de protocolos que pueden ser difíciles de entender para usuarios no familiarizados con este tipo de tecnología. La falta de documentación y antigüedad de esta hacen que desarrollar soluciones para este sistema sea complicado.

2.1.2. *Hardware*

Para utilizar APRS se necesita *hardware* especializado que incluya un transceptor de radio, un *Terminal Node Controller* (TNC) y un dispositivo GPS (opcionalmente). Existen otras opciones más baratas como los dispositivos *software defined radio* o RTL-SDR Figura 2.1 que se conectan directamente a un ordenador. Los TNC son

dispositivos que convierten las señales digitales emitidas por un ordenador en señales de radio y viceversa. El dispositivo GPS proporciona información de ubicación que se puede transmitir junto con otros datos en el cuerpo del mensaje.

El precio del *hardware* APRS puede variar dependiendo de la calidad y las características del equipo. Sin embargo, a pesar de no contar con precios prohibitivamente altos, la inversión que supone conlleva el hecho de que solamente los usuarios con un interés previo en la radioafición o la tecnología en general estén dispuestos a adquirirlo.



Figura 2.1: Ejemplo de un RTL-SDR de tipo USB.

2.1.3. OSINT y APRS

El APRS sirve como una herramienta de comunicación para los radioaficionados y de reporte de telemetría para las industrias. Pero se puede convertir en una valiosa fuente de información para la obtención de inteligencia (OSINT). Esto se debe a su capacidad para proporcionar una gran cantidad de datos en tiempo real sobre ubicación y estado de los activos así como una amplia gama de información adicional.

En el ámbito del OSINT, el APRS ofrece una serie de aplicaciones prácticas. Por ejemplo, los datos APRS son actualmente utilizados para el rastreo de vehículos en tiempo real, lo que resulta muy útil para empresas de logística y transporte. Además, la red APRS se utiliza para monitorear la actividad de estaciones meteorológicas, proporcionando datos en tiempo real sobre condiciones climáticas locales. También se utiliza en situaciones de emergencia, como pueden ser desastres naturales o incidentes críticos, el APRS desempeña un papel crucial al permitir la transmisión rápida de información sobre ubicaciones de refugios, recursos disponibles y necesidades de ayuda, ya que no depende necesariamente de infraestructura de una entidad como si lo hace la red telefónica.

Aunque el APRS tiene un gran potencial para aplicaciones en el ámbito del OSINT, su uso en esta área es prácticamente inexistente. Este hecho puede explicarse en parte por las barreras de entrada mencionadas anteriormente, que incluyen

la necesidad de poseer conocimientos especializados en radiocomunicación, obtener licencias de radioaficionado y adquirir *hardware* especializado.

2.2. Estado del Arte

Existen varios sitios web que ofrecen servicios relacionados con APRS, entre los cuales destacan **aprs.fi** y **aprs.to**, probablemente las dos webs más grandes.

2.2.1. aprs.fi

aprs.fi es una de las webs más utilizadas para visualizar datos APRS en tiempo real y acceder a un histórico extenso de información. Sin embargo, su interfaz puede considerarse un tanto desactualizada (ver Figura 2.2), lo que puede dificultar la navegación y la búsqueda de información específica. Aunque ofrece una gran cantidad de datos y un histórico significativo, la plataforma carece de capacidades avanzadas de filtrado de estaciones y mensajes. Además, para acceder a funciones más complejas y realizar extracciones de datos, es necesario crear una cuenta.

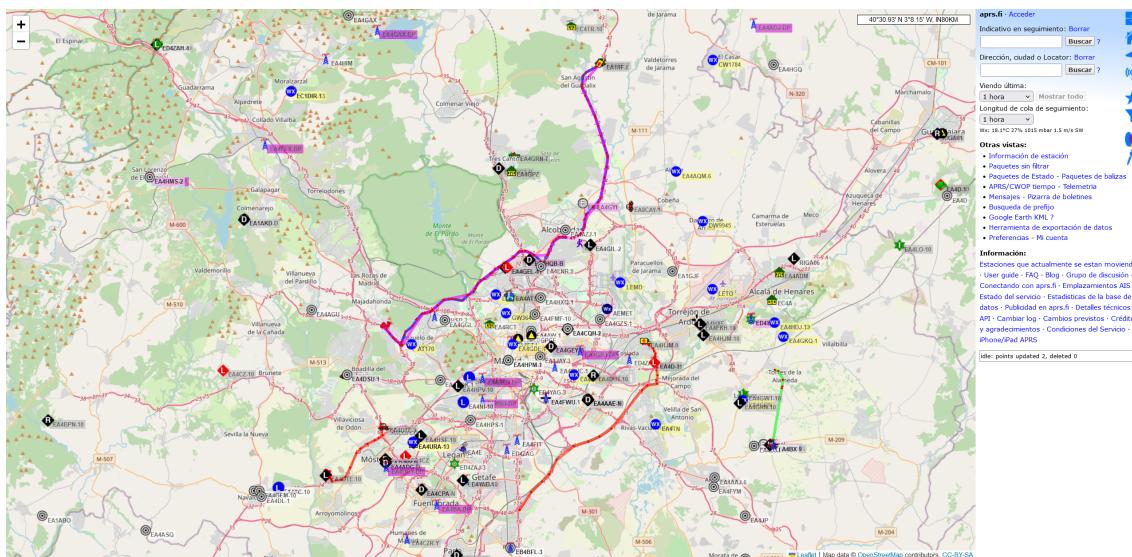


Figura 2.2: Interfaz de aprs.fi

2.2.2. aprs.to

Por otro lado, aprs.to ofrece una interfaz más moderna y cuidada (ver Figura 2.3), lo que facilita la navegación y la búsqueda de información. Además, permite realizar búsquedas básicas y aplicar filtros para refinar los resultados. Sin embargo, al igual que aprs.fi, también requiere que los usuarios se creen una cuenta para acceder a ciertas funcionalidades avanzadas.

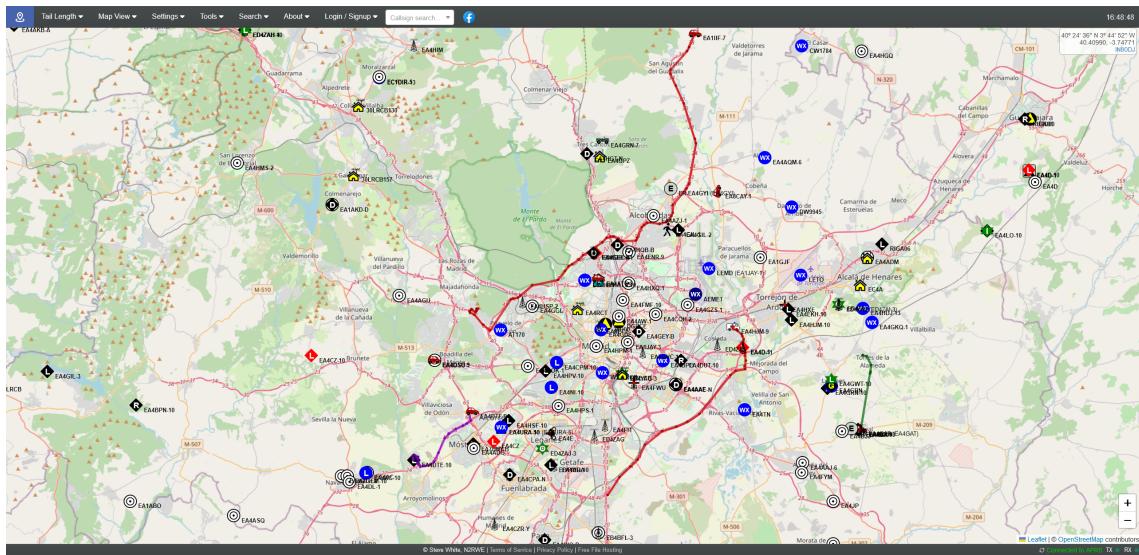


Figura 2.3: Interfaz de aprs.to

2.2.3. Comparación

En resumen, aprs.fi, la web de visualización de APRS por excelencia, ofrece una gran base de datos y una gran cantidad de información histórica de datos APRS, pero su interfaz puede resultar algo desactualizada y carece de capacidades avanzadas de filtrado.

Por otro lado, aprs.to presenta una interfaz más moderna y permite realizar búsquedas básicas y aplicar filtros, aunque también requiere crear una cuenta para acceder a ciertas funciones. Ambas plataformas tienen puntos fuertes y débiles y por esa razón lo mejor es usarlas en conjunto.

Capítulo 3

Descripción del Trabajo

En este capítulo se describen los módulos que conforman la aplicación así como la arquitectura y las tecnologías utilizadas para su implementación.

3.1. APRSINT

La solución propuesta se ha llamado **APRSINT**, que es una combinación de los acrónimos APRS y OSINT. APRSINT está dividida en tres módulos principales:

- **Adquisición de datos:** Este módulo se encargará de recopilar y almacenar los mensajes APRS de diversas fuentes, así como de integrar datos de otras fuentes abiertas y disponibles.
- **Procesamiento y análisis:** Este módulo se encargará de procesar y analizar los datos recopilados, ofreciendo herramientas avanzadas de visualización, filtrado y análisis de datos.
- **Visualización y presentación:** Este módulo se encargará de presentar los datos procesados y analizados de forma clara y comprensible, facilitando la interpretación y la toma de decisiones.

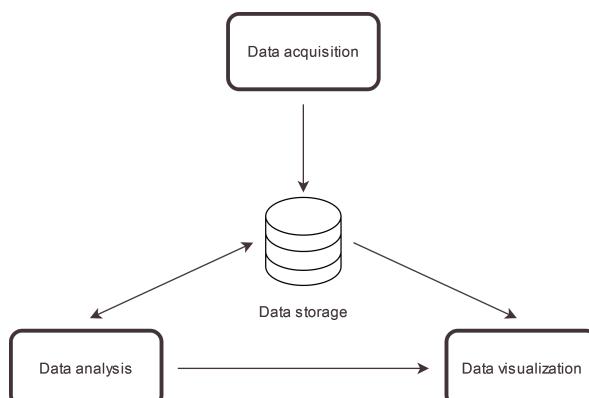


Figura 3.1: Módulos de la aplicación (elaboración propia).

3.2. Arquitectura de la aplicación

Se pueden categorizar los tres módulos (**Adquisición de datos, Procesamiento y análisis, Visualización y presentación**) de la aplicación mencionados previamente en dos categorías principales:

- **Adquisición de paquetes APRS:** En este módulo se encuentran los componentes encargados de la adquisición de paquetes APRS. Estos componentes se encargan de recibir los paquetes APRS, procesarlos y almacenarlos en la base de datos de la aplicación.
- **Aplicación web APRSINT:** En este módulo se encuentran los componentes encargados del análisis, la interpretación y la visualización, de los datos almacenados en la base de datos de la aplicación. Estos componentes se encargan de presentar la información de manera rápida y cómoda para el usuario.

3.3. Tecnologías utilizadas y marco teórico

Como se ha mencionado anteriormente, uno de los objetivos de APRSINT es el de ser una solución accesible y fácil de usar. Para lograr este objetivo, se han seleccionado tecnologías ampliamente utilizadas y bien documentadas. A continuación se describen las tecnologías utilizadas en cada uno de los módulos de la aplicación.

3.3.1. Raspberry Pi

La Raspberry Pi es un *Single board computer (SBC)* u ordenador de placa única desarrollada por la Fundación Raspberry Pi. Las Raspberry Pi son muy populares en el mundo de la informática y la electrónica por su bajo precio, reducido tamaño y su versatilidad.

- **Bajo precio:** La Raspberry es una opción económica para implementar desde prototipos hasta aplicaciones simples, lo que la hace muy accesible para una amplia gama de usuarios.
- **Bajo consumo energético:** Su diseño de bajo consumo energético la hace ideal para aplicaciones que requieren funcionamiento continuo.
- **Versatilidad:** La Raspberry Pi 4 es altamente versátil y puede adaptarse a una variedad de casos de uso, desde servidores ligeros hasta placas de desarrollo para robótica y automatización.

Se ha seleccionado la Raspberry Pi 4B¹ de 8GB de memoria RAM como plataforma de hardware para la implementación de la solución, debido a sus características y capacidades.

¹ Cuando se comenzó el proyecto todavía no se había lanzado la versión 5.

3.3.2. Disco duro SSD

Los discos duros de estado sólido (SSD) son una alternativa a los discos duros tradicionales (HDD) que ofrecen una mayor velocidad de lectura y escritura, menor consumo energético y mayor durabilidad. Se ha seleccionado un disco duro SSD de 250GB que se haya conectado a la Raspberry-pi para almacenar el gran volumen de datos que consume y genera la aplicación a su velocidad y fiabilidad.

3.3.3. Python

Python es un lenguaje de programación interpretado, de alto nivel y de propósito general. Es ampliamente utilizado en el desarrollo de aplicaciones web, científicas y de análisis de datos debido a su simplicidad, flexibilidad y facilidad de uso. Se ha seleccionado Python como lenguaje de programación principal para la implementación de la solución debido a la gran versatilidad que ofrece y sobre todo por la existencia de extensas librerías de visualización y manejo de grandes volúmenes de datos.

3.3.4. Dash

Dash es un *framework* creado por Plotly que permite crear “dashboards”, aplicaciones web interactivas y visualizaciones de datos atractivas utilizando Python como lenguaje de programación. Se ha seleccionado Dash como *framework* para la implementación de la interfaz web de la aplicación debido a su facilidad de uso y a la gran cantidad de funcionalidades que ofrece. Dash está escrito encima de Plotly.js, React y Flask lo que permite una gran capacidad de personalización.

- **Interactividad:** Dash permite crear aplicaciones web altamente interactivas, lo que facilita la exploración de datos y la toma de decisiones.
- **Flexibilidad:** Su arquitectura modular y su amplia gama de componentes permiten la creación de aplicaciones web personalizadas y adaptadas a las necesidades específicas del usuario.
- **Integración con Plotly:** Al estar desarrollado por Plotly, Dash ofrece una integración perfecta con las capacidades de visualización de datos de Plotly, lo que permite crear gráficos y visualizaciones muy atractivas.

Para crear la aplicación web se consideraron algunas alternativas como Django, Streamlit y PowerBI, sin embargo, Dash fue la opción escogida debido a su extensa capacidad de personalización de la que carecían las demás opciones.

3.3.5. Cosmograph JS

Cosmograph es una librería de JavaScript enfocada en la visualización de grandes grafos y redes complejas en aplicaciones web. Permite representar de manera inter-

activa relaciones entre entidades, facilitando la comprensión y el análisis de datos estructurados.

- **Visualización de grafos:** Cosmograph ofrece herramientas avanzadas para la representación visual de grafos como líneas temporales, histogramas y búsquedas de nodos, una mayor comprensión y capacidad de análisis de la información.
- **Rendimiento:** Cosmograph a diferencia de otras librerías más populares como Sigma JS transfiere todos los cálculos de posiciones de los nodos y aristas así como la representación gráfica de este a la GPU. Esto permite la visualización de grafos con miles de nodos y aristas sin afectar el rendimiento de la aplicación.
- **Personalización:** Ofrece opciones de personalización para adaptar la apariencia y el comportamiento de los nodos y aristas según las necesidades específicas del usuario.

Después de una gran cantidad de pruebas y tras considerar muchas alternativas como Sigma-JS, CytoScape y networkx, se acabó eligiendo Cosmograph sobre todo por su rendimiento.

3.3.6. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos relacional de código abierto y potente, conocido por su fiabilidad, robustez y capacidad para manejar grandes volúmenes de datos. Ofrece una amplia gama de características avanzadas que lo hacen adecuado para aplicaciones web y científicas que requieren almacenamiento y manipulación de datos complejos.

- **Fiabilidad y robustez:** PostgreSQL es conocido por su alta fiabilidad y capacidad para manejar grandes cargas de trabajo sin disminuir el rendimiento.
- **Escalabilidad:** Es altamente escalable y puede manejar grandes volúmenes de datos y transacciones concurrentes sin problemas.
- **Funcionalidades avanzadas:** Ofrece una amplia gama de funcionalidades avanzadas, como soporte para transacciones ACID, vistas materializadas, procedimientos almacenados y Full Text Search (búsqueda indexada).
- **Almacenamiento de datos semiestructurados:** PostgreSQL es capaz de almacenar y manipular datos no estructurados como JSON de manera eficiente, lo que lo hace adecuado para aplicaciones que requieren almacenamiento de datos semiestructurados.
- **Rendimiento:** PostgreSQL ofrece un muy buen rendimiento, especialmente en entornos de alta concurrencia y cargas de trabajo intensivas.

La elección de PostgreSQL como sistema de gestión de bases de datos se debió a su robustez, facilidad de uso y a la gran cantidad de funcionalidades que ofrece.

3.3.7. SQLAlchemy

SQLAlchemy es una biblioteca de Python que facilita la interacción con bases de datos relacionales utilizando un enfoque orientado a objetos. Permite trabajar con diferentes motores de bases de datos, como PostgreSQL, MySQL, SQLite, entre otros, de una manera consistente y eficiente.

- **Abstracción de la base de datos:** SQLAlchemy proporciona una capa de abstracción sobre la base de datos, esto permite interactuar con la base de datos utilizando objetos de Python en lugar de consultas SQL directas.
- **Compatibilidad con múltiples motores:** Es compatible con muchos motores de bases de datos, lo que brinda flexibilidad para trabajar con diferentes sistemas según las necesidades del proyecto. En este caso se ha utilizado el motor *Psycopg3* para la conexión con PostgreSQL.
- **ORM (mapeo objeto-relacional):** Ofrece un ORM potente y flexible que mapea objetos Python a tablas de bases de datos, facilitando el manejo de relaciones entre objetos y la persistencia de datos.
- **Seguridad:** SQLAlchemy proporciona herramientas para prevenir ataques de inyección SQL y otros problemas de seguridad comunes a las bases de datos.

La elección de SQLAlchemy se ha basado en su capacidad para mejorar el proceso de interacción con la base de datos. Proporciona una estrecha integración entre la base de datos y la aplicación, permitiendo una flexibilidad muy grande a la hora de hacer consultas o inserciones y sobre todo creando una capa de seguridad para evitar ataques.

3.3.8. Aprslib

Aprslib es una biblioteca de Python que facilita la interacción con el sistema y la red APRS. Permite recibir, enviar y decodificar paquetes APRS a través de la red APRS-IS.

- **Recepción de paquetes:** Aprslib permite recibir paquetes APRS de la red APRS-IS de manera sencilla mediante la conexión con los servidores centrales de APRS-IS.
- **Decodificación de paquetes:** Facilita la decodificación de paquetes APRS, permitiendo extraer información útil como la posición, velocidad y rumbo de los objetos rastreados.
- **Envío de paquetes:** Aprslib permite enviar paquetes APRS a través de la red APRS-IS, lo que facilita la integración con el sistema APRS.

Se ha elegido la librería Aprslib por su facilidad de uso, su documentación y su capacidad para decodificar los paquetes APRS.

3.3.9. AWS

Amazon Web Services (AWS) es una plataforma de servicios en la nube ofrecida por Amazon. Proporciona servicios de infraestructura informática, almacenamiento, bases de datos, análisis e inteligencia artificial, entre otros.

- **Fiabilidad:** La infraestructura global de AWS está diseñada para ser altamente disponible y resistente a fallos, lo que garantiza la continuidad del servicio y la seguridad de los datos.
- **Variedad de servicios:** AWS ofrece una amplia gama de servicios, desde almacenamiento y bases de datos hasta aprendizaje automático y análisis de datos, lo que permite a las empresas construir y desplegar una amplia variedad de aplicaciones y soluciones.
- **Flexibilidad:** AWS proporciona opciones flexibles de implementación, incluyendo la capacidad de utilizar infraestructura física, virtual o basada en contenedores, según las necesidades del proyecto.
- **Seguridad:** AWS cuenta con medidas de seguridad muy robustas para proteger los datos y las aplicaciones, incluyendo controles de acceso, cifrado de datos y protección contra amenazas.

La elección de AWS así como los detalles de la implementación en la nube se describirán en la siguiente sección.

3.3.10. Supervisord

Supervisord es un sistema de control de procesos para sistemas operativos tipo Unix, diseñado para iniciar, detener y gestionar procesos de manera sencilla y robusta. Permite supervisar y mantener en funcionamiento aplicaciones y servicios, reiniciándolos automáticamente en caso de fallos o reinicios del sistema.

- **Gestión de procesos:** Supervisord facilita la gestión de procesos al permitir iniciar, detener, reiniciar y supervisar procesos de manera centralizada.
- **Monitorización:** Supervisord proporciona información detallada sobre el estado de los procesos, incluyendo registros de eventos y estadísticas de rendimiento.
- **Reinicio automático:** En caso de fallos, Supervisord puede reiniciar automáticamente los procesos afectados, minimizando el tiempo de inactividad y manteniendo la disponibilidad del servicio.

Se ha seleccionado Supervisord como gestor de procesos gracias a su facilidad, flexibilidad (permitiendo ejecutar *scripts* de Python directamente) y robustez.

3.3.11. Pandas

Pandas es la librería de facto de Python para gestionar y analizar datos estructurados. Al estar construida sobre Numpy (escrito en C), ofrece un rendimiento considerablemente superior a la implementación manual de estructuras de datos. Además de posibilitar la lectura y escritura de datos en diversos formatos, facilita la manipulación, limpieza y visualización de datos, convirtiéndola en una herramienta fundamental para el análisis de datos en Python.

3.3.12. Conda

Conda es un gestor de paquetes, entornos y canales de código abierto que facilita la instalación y gestión de paquetes y dependencias en Python. Conda permite crear entornos virtuales aislados para proyectos específicos, lo que facilita la gestión de dependencias y la compatibilidad entre versiones de paquetes. Se ha seleccionado Conda como gestor de paquetes para la instalación de las librerías y dependencias necesarias para la aplicación.

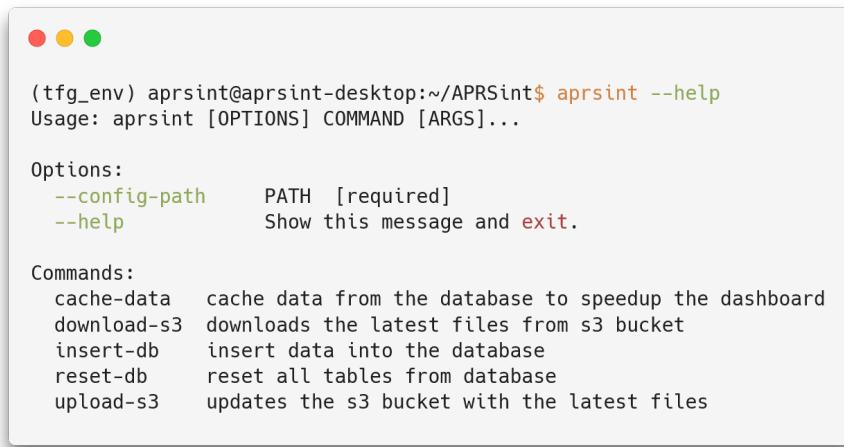
- **Gestión de paquetes:** Conda facilita la instalación, actualización y eliminación de paquetes de *software*. Esta característica es particularmente útil cuando se trabaja en proyectos que requieren bibliotecas específicas con versiones concretas.
- **Entornos virtuales:** Conda permite la creación de entornos virtuales independientes para cada proyecto. Esto significa que pueden existir diferentes versiones de paquetes instaladas en diferentes entornos sin que entren en conflicto entre sí. Esto es especialmente útil cuando se necesita trabajar en proyectos con diferentes requisitos de dependencias.
- **Portabilidad y reproducibilidad:** Al utilizar Conda para gestionar las dependencias de los proyectos, se puede garantizar que otros puedan reproducir el entorno de desarrollo exactamente como estaba. Esto es fundamental para la colaboración en proyectos de *software* y la investigación reproducible en ciencia de datos.
- **Flexibilidad:** Conda no se limita solo al ecosistema de Python. También puede gestionar paquetes de otros lenguajes de programación, como R, Java y C/C++. Esto lo convierte en una herramienta versátil para proyectos que involucran múltiples tecnologías.

Se ha seleccionado Conda como gestor de paquetes debido a su facilidad de uso, su robustez y la gran cantidad de paquetes y librerías disponibles en su repositorio.

3.3.13. Click

Click es una librería de Python que permite la creación de herramientas de línea de comandos de manera sencilla y eficiente. Click facilita la creación de interfa-

ces de usuario basadas en texto, lo que permite a los usuarios interactuar con las aplicaciones a través de la terminal. Se muestra un ejemplo de uso de Click en la Figura 3.2.



```
(tfg_env) aprsint@aprsint-desktop:~/APRSint$ aprsint --help
Usage: aprsint [OPTIONS] COMMAND [ARGS]...

Options:
  --config-path    PATH  [required]
  --help           Show this message and exit.

Commands:
  cache-data      cache data from the database to speedup the dashboard
  download-s3     downloads the latest files from s3 bucket
  insert-db        insert data into the database
  reset-db         reset all tables from database
  upload-s3        updates the s3 bucket with the latest files
```

Figura 3.2: Ejemplo de uso de Click (elaboración propia).

3.3.14. Apache Airflow

Apache Airflow es una plataforma de orquestación de tareas y flujos de trabajo. Es similar a Cron de Unix, pero permite una mayor personalización y control de las tareas que ejecuta. Las tareas se definen en archivos separados facilitando la compartimentalización y ofreciendo una mayor robustez y escalabilidad.

- **Orquestación de flujos de trabajo:** Airflow facilita la orquestación de flujos de trabajo complejos al permitir definir tareas y sus dependencias como DAG's, lo que proporciona una visión clara de la lógica de ejecución.
- **Escalabilidad:** Airflow es altamente escalable y puede manejar flujos de trabajo de cualquier tamaño, desde tareas simples hasta flujos de trabajo altamente complejos.
- **Monitorización y alertas:** Proporciona una interfaz de usuario web para monitorizar el estado de los flujos de trabajo, así como capacidades de alerta para detectar y responder a fallos o retrasos en la ejecución de tareas.
- **Extensibilidad:** Airflow es altamente extensible y permite integrar fácilmente con otros sistemas y herramientas, lo que permite construir flujos de trabajo personalizados que se adapten a las necesidades específicas del proyecto.

3.3.15. Configuración del proyecto

Para la configuración del proyecto se ha utilizado un entorno virtual de Conda. Se ha creado un entorno virtual con las librerías necesarias para la ejecución de la aplicación. Algunas de las librerías más importantes que se han utilizado son las siguientes:

- **Aprslib:** Para la recepción y decodificación de paquetes APRS.
- **Boto3:** Para la interacción con los servicios de AWS.
- **SQLAlchemy:** Para la interacción con la base de datos PostgreSQL.
- **Pandas:** Para la compresión y descompresión de ficheros.
- **Dash:** Para la creación de la aplicación web interactiva.
- **Click:** Para la creación de herramientas de línea de comandos.

Se ha creado un archivo de configuración en el que se definen las variables de entorno necesarias para la ejecución de la aplicación. Estas variables incluyen la configuración de la base de datos, las credenciales de AWS y la configuración de la aplicación web, entre otras.

Adicionalmente se ha configurado el proyecto con setuptools para facilitar la instalación y distribución de la aplicación. Se ha creado un archivo setup.py en el que se definen las dependencias del proyecto y se especifica la estructura del proyecto. Esto permite instalar la aplicación con un simple comando de pip.

```
pip install -i .
```

3.4. Adquisición de datos

En esta sección se describirá con detalle el módulo de adquisición de datos de la aplicación, incluyendo la arquitectura, los componentes y las tecnologías utilizadas.

3.4.1. RTL-SDR

En la primera fase del proyecto se adquirió un *dongle* RTL-SDR con el fin de recibir los paquetes APRS y procesarlos posteriormente. Estos dongles son relativamente baratos oscilando entre los 30 y 100 euros, se conectan mediante usb al ordenador y disponen de un puerto SMA en la parte superior para la antena. La peculiaridad de estos dispositivos reside en que son controlables mediante *software* y se pueden sintonizar en un rango de frecuencias muy amplio (24MHz a 1.7GHz).

Para utilizar estos dispositivos se necesita *software* especializado como GQRX, CubicSDR o SDRSharp. Es posible también usar RTL-SDR en la terminal creando un micrófono virtual para que el *software* de análisis pueda decodificar los paquetes.

Tras multitud de pruebas fallidas con distintos *software* de visualización y análisis y *software* de audio como Direwolf, se decidió no continuar por ese camino y probar otras alternativas como la que finalmente se ha implementado y se explica en la siguiente sección.

3.4.2. APRS-IS

APRS-IS es un sistema que permite la transmisión de mensajes APRS por Internet. La ventaja principal de este sistema, en comparación con la recopilación de información mediante una antena, es que permite obtener un flujo de datos mucho mayor. Esto se debe a la extensa red de antenas distribuidas globalmente. Además, otra ventaja significativa es la capacidad de recibir datos de cualquier parte del mundo sin necesidad de tener una antena en esa ubicación.

Infraestructura de la red APRS-IS

Componentes principales:

- **Trackers o TNC:** Son los emisores de los paquetes APRS, suelen corresponder a estaciones fijas o móviles que envían por radio los paquetes con información de posición, velocidad, rumbo, etc.
- **Digipeaters:** Son estaciones que reciben los paquetes APRS y los retransmiten, permitiendo que los paquetes lleguen a una mayor distancia. Son análogos a los repetidores de internet.
- **I-Gates:** Son estaciones que reciben los paquetes APRS por radio y los envían a la red APRS-IS, permitiendo que los paquetes sean accesibles a través de internet.
- **Servidores APRS-IS:** Son servidores que reciben los paquetes APRS de los I-Gates y los almacenan en una base de datos, permitiendo que los clientes accedan a los paquetes a través de internet. Los servidores APRS-IS suelen estar distribuidos geográficamente para mejorar la disponibilidad y la redundancia.
- **Clientes APRS-IS:** Pueden ser otros servidores, aplicaciones web o aplicaciones móviles que acceden a los servidores APRS-IS para obtener los paquetes APRS y mostrarlos a los usuarios.

La ruta que sigue un paquete APRS desde un tracker hasta un cliente APRS-IS se visualiza en la Figura 3.3 y comprende los siguientes pasos: primero, un dispositivo **tracker** emite un paquete APRS a través de radiofrecuencia. Este paquete es recibido por un **Digipeater**, que funciona como un repetidor y retransmite la señal. Luego, un **I-Gate** recoge el paquete y lo envía a un servidor **APRS-IS** a través de Internet. Por último, un **cliente** APRS-IS accede a este servidor para recuperar el paquete y presentarlo al usuario para su visualización.

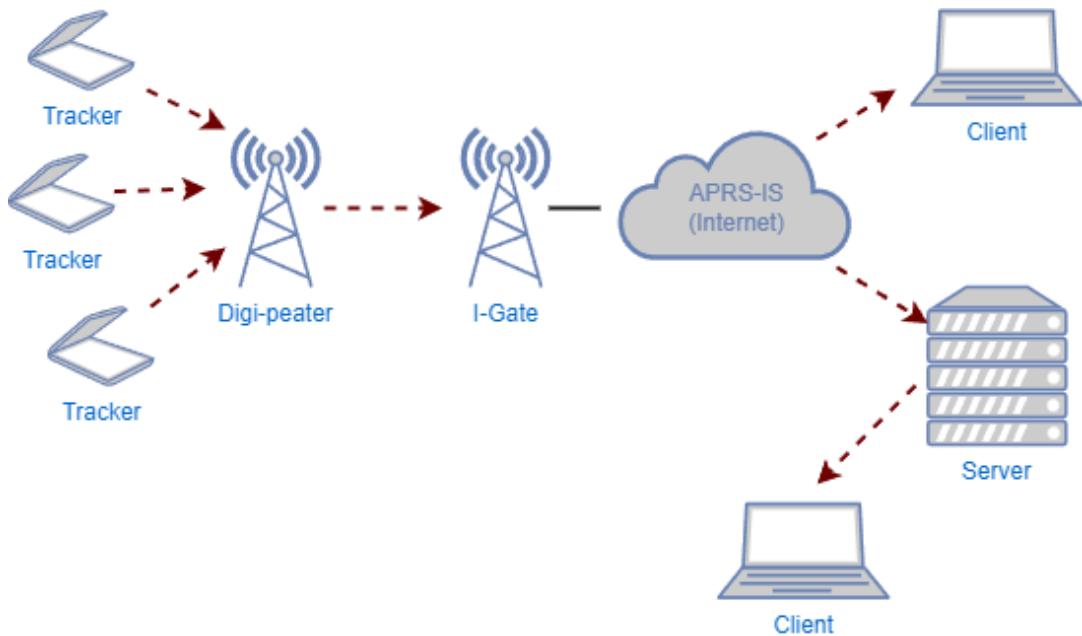


Figura 3.3: Infraestructura de la red APRS - APRS-IS (elaboración propia).

Se ha seleccionado APRS-IS como fuente de datos para la aplicación debido a su facilidad de uso, su disponibilidad y la gran cantidad de información que ofrece desde sus servidores.

3.4.3. La trama APRS

El sistema APRS utiliza tramas AX.25 (Amateur X.25) para la transmisión de datos a través de las ondas de radio. El protocolo X.25 es un protocolo de nivel de enlace de datos que se diseñó y se creó para redes de conmutación de paquetes. El protocolo AX.25 es una versión simplificada del protocolo X.25 que se utiliza en las redes de radioaficionados.

Formato de la Trama APRS

Todos los paquetes APRS se envían utilizando tramas **no numeradas** a diferencia de otros protocolos como TCP que utilizan tramas numeradas. Se muestra un ejemplo de trama APRS en la Figura 3.4.

Formato de trama-UI AX.25									
	Flag	Destino	Origen	Dirección de Digipeater (0-8)	Campo de control (UI)	ID de protocolo	Datos	FCS	Flag
Bytes:	1	7	7	0 - 56	1	1	1 - 256	2	1

Figura 3.4: Formato de trama AX.25 (elaboración propia, [8]).

Una trama AX.25 consta de los siguientes campos:

- **Flag:** Marca el inicio y el final de la trama. Tiene un valor fijo de 0x7E.
- **Destino:** Identifica la estación de destino de la trama. Puede ser una dirección de estación completa o una dirección abreviada.
- **Origen:** Identifica la estación de origen de la trama. Puede ser una dirección de estación completa o una dirección abreviada.
- **Dirección de Digipeater (0-8):** Identifica los repetidores que deben reenviar la trama. Puede haber hasta 8 repetidores en la ruta.
- **Campo de control (UI):** Este campo se establece a 0x03 (trama no numerada).
- **ID de protocolo:** Este campo se establece a 0xF0 (no hay protocolo de capa 3).
- **Datos:** Este campo contiene los datos de la trama. En el caso de las tramas APRS, este campo puede contener información sobre la posición, la velocidad, el estado y otros datos de la estación.
- **FCS:** Frame Check Sequence (FCS): Es un código de verificación de redundancia cíclico (CRC) que se utiliza comprobar la integridad de la trama.

Contenido de la Trama

Dentro del campo de datos de una trama AX.25, se pueden incluir diversos datos relevantes para APRS. Esto puede incluir información de posición (latitud y longitud), velocidad, altitud, rumbo, comentarios del operador, mensajes de texto, etc ... [3][9].

NOCALL > APRS, OH7RDA,OH7RDB : !4903N07201W-test A=001234

Origen	Destino	Digipeaters y I-Gates	Mensaje
---------------	----------------	------------------------------	----------------

En este ejemplo, el campo de datos contiene información sobre posición, altitud y un comentario. La posición está representada en formato APRS, que es una forma abreviada de representar la latitud y longitud en grados decimales. La altitud se representa en pies y el comentario es un mensaje de texto opcional, en este caso es “test”.

Uso en APRS

En el contexto de APRS, las tramas AX.25 se utilizan para transmitir actualizaciones de posición de estaciones móviles, estaciones meteorológicas, balizas y otros objetos de interés. Estas tramas son recibidas por otras estaciones en la red APRS, lo que permite el seguimiento en tiempo real de la ubicación y otros datos de interés.

3.4.4. Recepción de paquetes APRS

Para la recepción de paquetes APRS se ha utilizado la librería Aprslib de Python. Esta librería permite conectarse a un servidor APRS-IS y recibir los paquetes APRS en tiempo real. Se ha creado un sistema que utilizando esta librería se conecta a un servidor APRS-IS, recibe los paquetes APRS y los guarda en un *buffer* en memoria. Cuando el *buffer* se llena con aproximadamente 10.000 mensajes en alrededor de 15 segundos, se procede a escribir el *buffer* en un fichero comprimido en el disco duro SSD.

Un detalle interesante es que los mensajes APRS no se decodifican en este punto, de hecho se guardan sin ningún tipo de procesamiento en ficheros binarios comprimidos en formato gzip para no ocupar mucho espacio en disco. Esto se hace así para evitar la sobrecarga de procesamiento en la Raspberry Pi y para poder procesar los mensajes en un servidor más potente en la nube.

Es importante mencionar que el sistema de recepción de paquetes APRS se ejecuta en segundo plano en modo demonio utilizando el gestor de procesos Supervisord, lo que garantiza que el sistema se mantenga en funcionamiento incluso en caso de fallos o reinicios del sistema.

3.4.5. Procesado de paquetes APRS

En esta sección se describirá con detalle el módulo de procesado de paquetes APRS de la aplicación, incluyendo la arquitectura, los componentes y las tecnologías utilizadas.

Subida de ficheros a AWS S3

Una vez los ficheros se han guardado en el disco duro SSD, se procede a subirlos a un *bucket* S3 en AWS. Para ello se ha utilizado la librería Boto3 de Python, que permite interactuar con los servicios de AWS desde Python. Se ha creado un *script* que se ejecuta una vez al día mediante la orquestación de Apache Airflow y que sube los ficheros comprimidos al *bucket* S3. Una vez subidos los ficheros, se eliminan del disco duro SSD para liberar espacio. Cuando un fichero se sube a S3 se desencadena un evento.

Procesado de paquetes APRS en AWS Lambda

Una vez los ficheros se han subido a S3, se desencadena un evento que ejecuta una función lambda Figura 3.5 en AWS. Esta función lambda se encarga de descomprimir el fichero, procesar los mensajes APRS y convertirlos en formato JSON.

Por defecto Lambda no puede hacer uso de librerías de terceros por lo que se ha tenido que encapsular la librería Aprslib en un fichero .zip, subirlo a Lambda y

establecerlo como una capa de Lambda para poder hacer uso de ella.

Para la descompresión de los ficheros se ha utilizado la librería gzip de Python, que permite leer y escribir ficheros comprimidos en formato gzip. Para el procesado de los mensajes APRS se ha utilizado la librería Aprslib de Python como ya hemos mencionado, que permite decodificar los mensajes APRS y extraer información útil como la posición, velocidad y rumbo de los objetos rastreados en formato clave-valor.

Esta función lambda se ejecuta de manera asíncrona y paralela, lo que permite procesar grandes volúmenes de mensajes APRS de manera eficiente y escalable. Una vez procesados los mensajes APRS, se eliminan los ficheros comprimidos del S3 de entrada (aprsinput) para liberar espacio y evitar duplicados.

Finalmente los mensajes procesados se guardan en un *bucket* S3 de salida (aprsoutput) en formato JSON para su posterior descarga. La descarga de los ficheros JSON ocurre una vez al día mediante la orquestación de Apache Airflow y la librería Boto3. Estos ficheros (uno por fichero de entrada) se descargan a otra carpeta temporal en la Raspberry Pi.

```

# Importar librerias necesarias
import json
import boto3
import gzip as gz
from io import BytesIO
from aprslib import parse

def lambda_handler(event, context):
    # Recibir el archivo desde s3
    s3 = boto3.client("s3")
    bucket = event["Records"][0]["s3"]["bucket"]["name"]
    key = event["Records"][0]["s3"]["object"]["key"]
    response = s3.get_object(Bucket=bucket, Key=key)
    gz_file = response["Body"].read()
    packets = []
    error = 0
    success = 0
    #Descomprimir y convertir a JSON
    try:
        with gz.GzipFile(fileobj=BytesIO(gz_file), mode="rb") as f:
            for line in f.readlines():
                try:
                    packets.append(parse(line))
                    success += 1
                except Exception:
                    error += 1
    s3.put_object(
        Body=json.dumps(packets).encode("utf-8"),
        Bucket="aprsoutput",
        Key=(key.split("."))[0] + ".json",
    )
    try:
        s3.delete_object(Bucket="aprsinput", Key=key)
    except Exception as e:
        print(f"Cannot delete object with error {e}")
    except Exception as e:
        return {"statusCode": 400, "body": f"Error: {e}"}
    return {"statusCode": 200, "body": f"Success: {success}, Error: {error}"}

```

Figura 3.5: Procesado de paquetes APRS en AWS Lambda (elaboración propia).

3.4.6. Almacenamiento de paquetes en la base de datos

Una vez los ficheros JSON ya procesados se han descargado al directorio temporal en la Raspberry Pi, se procede a almacenarlos en la base de datos PostgreSQL. Para ello se ha utilizado la librería SQLAlchemy de Python, que permite interactuar con bases de datos relacionales de manera sencilla y eficiente. Se ha creado un *script* que se ejecuta una vez al día mediante la orquestación de Apache Airflow y que lee los ficheros JSON, procesa los mensajes APRS y los almacena en la base de datos.

Este paso esconde un gran problema que hubo de ser solucionado. El *script* en un principio seguía los siguientes pasos.

1. **Lectura de ficheros JSON:** El *script* lee los ficheros JSON de la carpeta local /tmp descargados del *bucket* S3.
2. **Extracción de la estación:** El *script* extrae la estación origen y destino de cada mensaje y en caso de no existir en la base de datos las crea.
3. **Extracción de las posiciones:** Se extraen las posiciones (si las hay) de cada mensaje y el timestamp si lo hay y se almacenan en la base de datos.
4. **Identificación del país:** Mediante la librería Geopy y ficheros de formas de países se identifica el país de la estación.
5. **Extracción de los mensajes:** Se extraen los mensajes de cada línea del JSON y se almacenan en la base de datos.

Al utilizar este enfoque, se detectó un gran problema de rendimiento, ya que cada archivo JSON contenía alrededor de 10.000 mensajes y en la creación de los objetos, inserciones en la BD y detección del país se tardaba alrededor de 40 segundos por fichero JSON. Esto aunque no parezca demasiado tiempo, si se tiene en cuenta que la cantidad de mensajes APRS que se reciben en 40 segundos es alrededor de 45.000, se puede ver que el sistema no era escalable.

Optimizaciones

Para solucionar este problema se realizaron las siguientes optimizaciones:

- **Uso de sets:** Se ha cambiado el uso de consultas a la BD por sets en la creación de las estaciones para acelerar la comprobación la existencia de una estación en la base de datos.
- **Inserción por lotes:** Se ha modificado la interfaz con SQLAlchemy para permitir inserciones en lotes de 250 mensajes.
- **Detección de países:** Se ha cambiado la librería Geopy por GeoPandas que permite la una detección de países más rápida utilizando paralelización.

Estas optimizaciones han acelerado el tiempo de procesado de un fichero JSON de 40 a 2 segundos, lo que ha permitido que el sistema sea escalable y pueda procesar grandes volúmenes de mensajes de manera eficiente.

3.4.7. Base de datos

Como se ha mencionado previamente, se ha elegido PostgreSQL como sistema de gestión de bases de datos para la aplicación. Esta elección se ha basado en la robustez, la fiabilidad y la capacidad de manejar grandes volúmenes de datos que ofrece PostgreSQL incluso en sistemas con recursos reducidos. La base de datos se ha diseñado siguiendo un modelo relacional Figura 3.6 que permite almacenar y relacionar la información de los mensajes APRS de manera eficiente, el modelo cuenta con las siguientes tablas.

- **stations:** Almacena la información de las estaciones APRS, incluyendo el identificador de la estación (CALLSIGN), el ssid y su símbolo en formato carácter.
- **station_locations:** Almacena la información de las posiciones ¹ de las estaciones APRS, incluyendo el país, la latitud, la longitud y la fecha y hora en la que se han recibido.
- **messages:** Almacena la información de los mensajes APRS, incluyendo el contenido del mensaje, el tipo de mensaje, el timestamp y el mensaje original retransmitido.
- **qrz_profiles:** Sirve como una caché que almacena la información de los perfiles de los usuarios de QRZ, incluyendo el identificador (CALLSIGN), el nombre, la dirección, la ciudad, el estado, el código postal, el país, la latitud, la longitud y la fecha de nacimiento entre muchos otros.

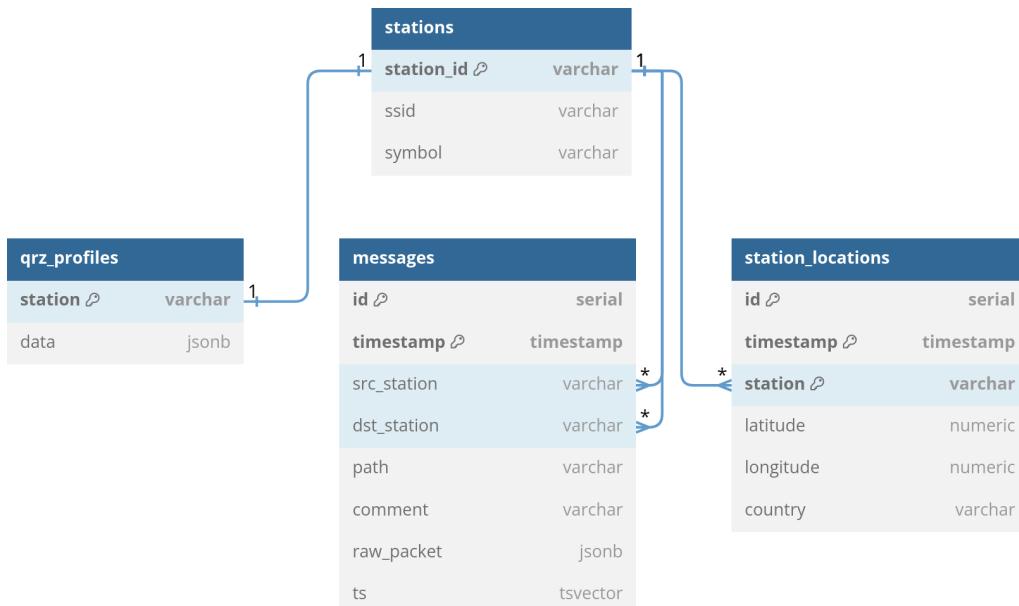


Figura 3.6: Estructura de la base de datos (elaboración propia).

¹ Esta tabla tiene un campo id en la clave primaria por si una estación ha emitido un mensaje sin timestamp

3.4.8. Flujo de un paquete APRS

En esta sección se describe con detalle el flujo de un paquete APRS a través de la aplicación, desde la recepción hasta el almacenamiento en la base de datos.

1. **Recepción:** El paquete APRS se recibe a través de la red APRS-IS y se almacena en un *buffer* en memoria.
2. **Almacenamiento en disco:** Cuando el *buffer* se llena con aproximadamente 10.000 mensajes, se escribe el *buffer* en un fichero comprimido en el disco duro SSD.
3. **Subida a S3:** El fichero comprimido se sube a un *bucket* S3 en AWS para su procesamiento posterior.
4. **Procesado en Lambda:** Una función lambda en AWS descomprime el fichero, procesa los mensajes APRS y los convierte en formato JSON.
5. **Almacenamiento en S3:** Los ficheros JSON procesados se guardan en un *bucket* S3 de salida para su posterior descarga.
6. **Descarga a la Raspberry Pi:** Una vez al día, los ficheros JSON se descargan a una carpeta temporal en la Raspberry Pi.
7. **Almacenamiento en la base de datos:** Un *script* en la Raspberry Pi lee los ficheros JSON, procesa los mensajes APRS y los almacena en la base de datos PostgreSQL.

3.5. Visualización y presentación de datos

En esta sección se describe con detalle el módulo de visualización y presentación de datos de la aplicación, incluyendo la arquitectura, los componentes y las tecnologías utilizadas.

3.5.1. Dash

Dash es un *framework* de Python creado por Plotly para la creación de aplicaciones web interactivas y visualizaciones de datos. Dash permite crear aplicaciones web interactivas y visualizaciones de datos atractivas utilizando Python como lenguaje de programación. Dash está escrito encima de Plotly.js, React y Flask lo que permite una gran capacidad de personalización.

Dash ofrece una versión de pago llamada Dash Enterprise que ofrece una gran cantidad de funcionalidades para elaborar aplicaciones web de manera más rápida y sencilla. Sin embargo, para este proyecto se ha utilizado la versión gratuita de Dash, con algunas librerías adicionales entre las que se encuentran:

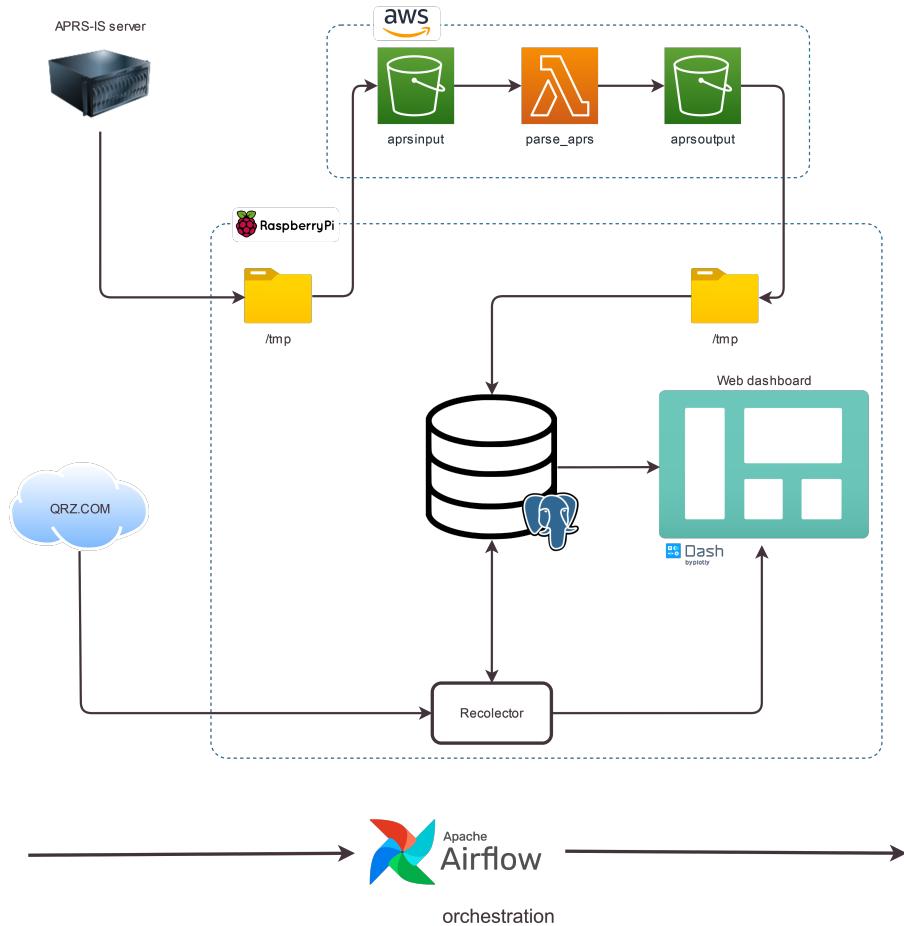


Figura 3.7: Flujo de datos en APRSINT (elaboración propia).

- **Dash core components:** Contiene multitud de componentes interactivos como gráficos, tablas, *sliders*, *dropdowns*, entre otros, que permiten a los usuarios interactuar con los datos de manera intuitiva.
- **Dash html components:** Contiene los *wrappers* de las etiquetas HTML como *divs*, *spans*, *inputs*, entre otros, que permiten personalizar la apariencia y el diseño de la aplicación web.
- **Dash mantine components:** Ofrece una amplia gama de componentes de Mantine como *navbars*, tarjetas, modales, entre otros, que permiten crear aplicaciones web atractivas y responsivas.
- **Dash express:** Añade componentes extra como un panel de filtrado y una barra de navegación ².
- **Plotly express:** Es el principal competidor de Matplotlib en el mundo de la visualización de datos en Python. Ofrece una gran cantidad de gráficos y visualizaciones de datos interactivos.

²Se ha modificado esta librería para adaptarla a la estructura de la aplicación.

3.5.2. Diseño

El diseño de la aplicación web se ha basado en la simplicidad, la usabilidad y la accesibilidad. El primer diseño del proyecto se realizó en la aplicación de prototipado Figma como se muestra en la Figura 3.8.

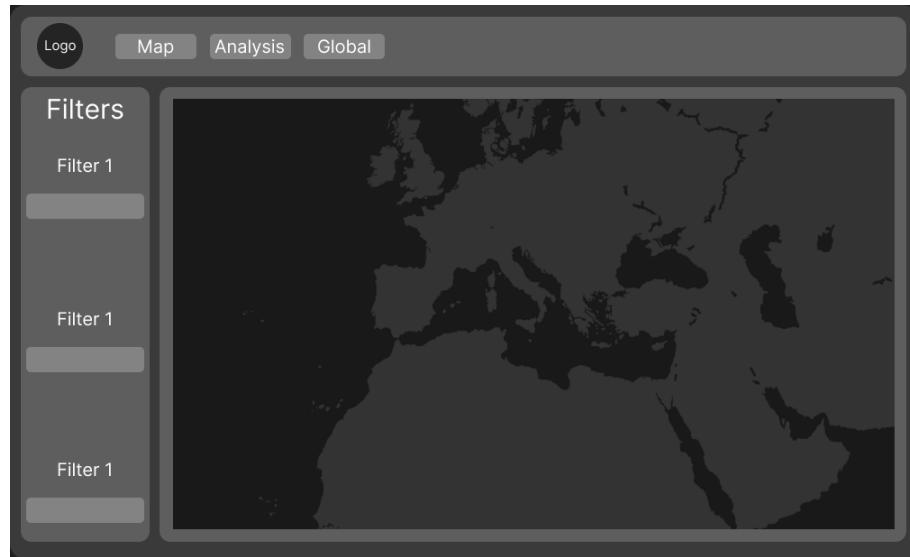


Figura 3.8: Primer diseño de la aplicación web en Figma (elaboración propia).

Este diseño se ha ido modificando y adaptando a lo largo del desarrollo del proyecto para mejorar la usabilidad y la experiencia del usuario. Este diseño muestra la pantalla principal de la aplicación, que incluye un mapa con las estaciones APRS, y un panel de control con filtros y opciones de visualización.

3.5.3. Home - Primera pantalla

En esta sección se describe la primera pantalla de la aplicación web, que muestra un mapa con las estaciones APRS y un panel de control con filtros y opciones de visualización.

Obtención de datos

Siguiendo la estrategia de otros sistemas de *dashboarding* como PowerBI o Tableau y con el objetivo de mejorar la eficiencia y la escalabilidad del sistema, se ha optado por eliminar en lo posible las consultas a la base de datos en tiempo real. Para ello se ha creado un sistema de caché que almacena los datos a representar en esta pantalla en un archivo de rápida lectura que es actualizado diariamente mediante Apache Airflow. Este sistema de caché permite reducir la carga en la base de datos y mejorar el rendimiento de la aplicación web.

El sistema de caché se ha implementado utilizando la librería Pandas de Python, usando un fichero .feather para almacenar los datos en formato binario y comprimido y reducir significativamente³ el tiempo de carga de la página.

Mapa de estaciones APRS

El mapa de estaciones APRS muestra la posición de cada una de las estaciones APRS, representadas por un marcador en el mapa. Es un mapa interactivo que permite hacer zoom, desplazarse y al posicionar el cursor encima de una estación aparecerá un recuadro ofreciendo información adicional.

Esta página hace uso de la librería Plotly para renderizar el mapa y del servicio web de mapas Mapbox para obtener los mapas y estilos de mapa.

Filtros

El panel de control de la aplicación web incluye una serie de filtros y opciones de visualización que permiten al usuario personalizar la información mostrada en el mapa. La ventaja de estos filtros es que son aditivos y se pueden combinar para obtener con exactitud la información requerida. Los filtros disponibles son:

- **Filtro por rango de fechas:** Permite filtrar las estaciones APRS por rango de fechas, mostrando solo las estaciones de las que se ha recibido algún mensaje en el rango de fechas seleccionado.
- **Filtro por país:** Permite filtrar las estaciones APRS por país, mostrando solo las estaciones que se encuentran en el país seleccionado.
- **Filtro por contenido del mensaje:** Este es quizás el filtro más interesante, ya que permite filtrar las estaciones por el contenido del mensaje, este filtro será explicado más adelante.
- **Filtro por tipo de estación:** Permite filtrar las estaciones APRS por tipo, mostrando solo las estaciones que corresponden al tipo seleccionado.
- **Filtro por ssid:** Permite filtrar las estaciones por ssid.

Búsqueda difusa en contenido de mensajes

Este filtro es personalmente el más interesante de todos los filtros disponibles. El objetivo es permitir al usuario buscar mensajes emitidos por las estaciones que contengan una palabra o frase específica. Para ello se han probado varias alternativas como el uso de expresiones regulares o una implementación en Python puro. El problema de estas opciones es que son muy lentas para el enorme volumen de datos

³Los ficheros de tipo .feather son hasta 100 veces más rápidos en lectura y escritura que los ficheros .csv[6]

con el que se cuenta y al ser dinámico no era factible realizar esa búsqueda en tiempo real.

Finalmente se ha optado por el uso de PostgreSQL FTS (*Full Text Search*). PostgreSQL FTS es un sistema de búsqueda de texto completo que permite realizar búsquedas de texto en grandes volúmenes de datos de manera eficiente. Las ventajas que ofrece este sistema son:

- **Rendimiento:** Todo el proceso de búsqueda se realiza en la base de datos, lo que permite realizar búsquedas de texto sin transferir todos los mensajes al servidor para su búsqueda.
- **Búsqueda difusa:** Permite realizar búsquedas de texto difuso, lo que significa que se pueden buscar palabras o frases que contengan errores tipográficos o que no coincidan exactamente con el texto buscado.
- **Indexación:** PostgreSQL FTS indexa automáticamente los mensajes de texto, lo que permite realizar búsquedas de texto de manera eficiente incluso en grandes volúmenes de datos.

Se ha creado un índice de texto completo en la columna de mensajes de la tabla de mensajes y se ha utilizado la función `to_tsvector` para convertir los mensajes en un vector de texto completo. Posteriormente se ha utilizado la función `plainto_tsquery` para convertir la palabra o frase de búsqueda en una consulta de texto completo y finalmente se ha utilizado la función `ts_query` para realizar la búsqueda de texto completo en la columna de mensajes. Este filtro permite al usuario buscar mensajes emitidos por las estaciones que contengan una palabra o frase específica, lo que facilita la identificación y el análisis de los mensajes relevantes [1].

Cuando un usuario realiza una búsqueda de texto completo, se muestra un modal con la lista de todos los mensajes que contienen la palabra o frase de búsqueda, ordenados por relevancia haciendo uso de la función `ts_rank`. Otra funcionalidad interesante es que la palabra o palabras buscadas se resaltan en el mensaje para facilitar su identificación como se muestra en la Figura 3.9.

Station	Comment
K4SRC	Jay Hospital Emergency
K4SRC	Santa Rosa Hospital Emergency
K4SRC	Jay Hospital Emergency

Figura 3.9: Resultados de la búsqueda de «emergency hospital» (elaboración propia).

En el modal se muestra una tabla de dos columnas, la columna izquierda contiene un botón con el nombre de la estación y la columna derecha el contenido del mensaje.

3.5.4. Station - Segunda pantalla

Esta es la página que hace única a APRSINT. En esta página se muestra la información detallada de una estación APRS en concreto. Se puede llegar a esta pantalla desde la página principal haciendo clic en un marcador de una estación en el mapa, haciendo clic en una estación en la búsqueda en los mensajes o haciendo clic en un nodo del grafo de la página 3. La pantalla station está dividida en tres secciones.

Información de QRZ

QRZ es una base de datos de radioaficionados que contiene información detallada sobre los radioaficionados de todo el mundo. Para acceder a la información disponible en la web de QRZ⁴ es necesario tener una cuenta y estar registrado. QRZ cuenta con un servicio de API que permite acceder a la información de los radioaficionados de manera programática.

Para obtener la información de los radioaficionados se ha utilizado la librería requests de Python, que permite obtener la información necesaria de QRZ.

Con el fin de acelerar las consultas, se ha implementado una tabla en la base de datos para almacenar en caché las solicitudes, lo que disminuye la cantidad de consultas a una estación determinada si esta ya ha sido solicitada con anterioridad.

Este módulo es de los más complejos de la aplicación, pero a su vez de los más útiles, ya que permite al usuario obtener información muy detallada de una estación APRS en concreto y de la persona que está detrás. Algunos de los datos que se pueden obtener son:

- Nombre de la persona registrada
- Fechas de registro y última actualización en la web
- Alias conocidos de la estación
- Dirección de la persona registrada (Como link a google maps)
- Fecha de nacimiento de la persona registrada

Información de posiciones

Esta sección se compone de dos partes. En la parte superior se muestra un mapa con todas las posiciones reportadas por la estación, se muestra también la caja delimitadora de todas las posiciones.

En la parte inferior se muestran estadísticas de las posiciones reportadas por la estación como la frecuencia media de emisión, la primera y última emisión recibida

⁴<https://qrz.com>

y por cada mensaje emitido por la estación se muestra el número de repeticiones del mensaje y las URL (si existen) encontradas en los mensajes.

Información de mensajes

Finalmente encontramos la sección de mensajes. En esta sección se muestra una tabla en la que se muestra la información de los mensajes emitidos por la estación. La tabla se compone de las siguientes columnas:

- La fecha de emisión del mensaje.
- La latitud y longitud de la estación en el momento de la emisión.
- El país en el que se encontraba la estación en el momento de la emisión.
- La estación destino del mensaje.
- El *path* es decir los *Digipeaters* que han retransmitido el mensaje.
- El contenido del mensaje.

La tabla es interactiva y permite al usuario filtrar los mensajes por el intervalo de fechas que pretenda.

3.5.5. Graph - Tercera pantalla

Esta pantalla es la última de la web, en ella se muestra un grafo dirigido en el que los nodos son las estaciones y las aristas dirigidas, mensajes como se muestra en la Figura 3.10.

El grafo es interactivo y permite hacer zoom, desplazarse y al posicionar el cursor encima de un nodo o arista aparecerá un recuadro ofreciendo información adicional. El color y tamaño de los nodos depende de la cantidad de mensajes que ha emitido o recibido la estación y de misma manera el color y tamaño de las aristas depende de la cantidad de mensajes que se han emitido o recibido desde la estación origen y destino.

Las características que se buscaban en la solución final eran:

- **Interactividad:** Se buscaba un grafo interactivo que permitiera al usuario explorar las relaciones entre las estaciones APRS.
- **Rendimiento:** Quizás la característica más importante, el grafo debía ser capaz de mostrar una gran cantidad de estaciones y conexiones sin afectar al rendimiento de la aplicación.
- **Personalización:** Se buscaba un grafo que permitiera personalizar la apariencia y el comportamiento de los nodos y aristas según las características de cada estación y mensaje.

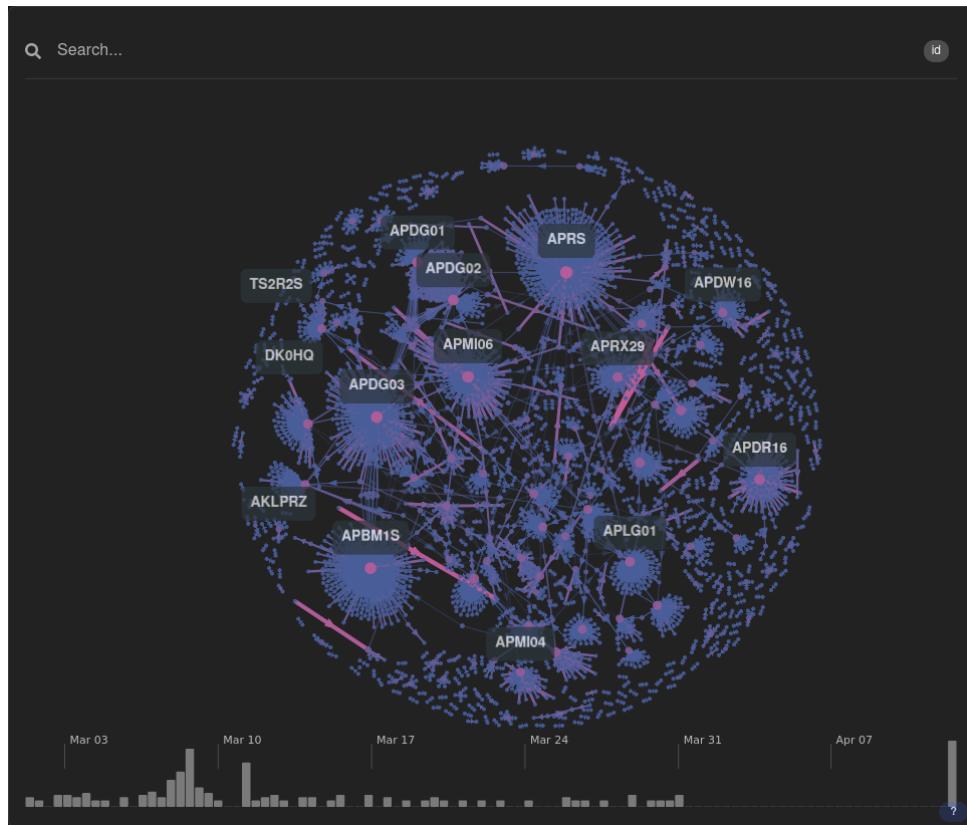


Figura 3.10: Grafo de estaciones APRS (elaboración propia).

Para alcanzar la solución final que se presenta ahora en la página, se han probado muchas opciones que se han ido descartando por diferentes motivos.

Alternativas probadas

En un primer momento se intentó utilizar la librería **Networkx** de Python para la creación del grafo, sin embargo, se descartó debido a su rendimiento y a su falta de interactividad. Posteriormente se intentó utilizar la librería **Cytoscape** que es la solución por defecto que ofrece Dash, esta ofrece una gran cantidad de funcionalidades para la creación de grafos interactivos, sin embargo, se descartó de nuevo debido a su pobre rendimiento.

Una de las alternativas que también se probó fue la librería **Sigma JS** que ofrece una gran cantidad de funcionalidades y personalización para la creación de grafos interactivos. Esta librería cuenta con una funcionalidad interesante *forceAtlas2* que permite calcular mediante una simulación de fuerzas la posición de los nodos y aristas en el grafo. Sin embargo, de nuevo se descartó debido a su rendimiento con una gran cantidad de nodos y aristas.

Finalmente se optó por la librería **Cosmograph JS**. Cosmograph utiliza la librería *Cosmos* de cálculo de posiciones mediante la GPU y por tanto puede manejar un gran número de nodos y aristas.

El Grafo

El problema principal con Cosmograph ha sido la falta de documentación, debido a que es una librería bastante joven, y la dificultad de integrarla con Dash. Esta librería está publicada en NPM (el repositorio de librerías de JavaScript) y se ha tenido que utilizar bundle.js para convertir el código de JavaScript en un solo fichero que se pueda importar en Dash. Ha sido necesario modificar el código fuente de la librería para terminar de integrarla con Dash.

Se han utilizado tres componentes de Cosmograph JS para la creación del grafo, el componente *Cosmograph* que crea el grafo, el componente *CosmographSearch* que permite mediante una barra de búsqueda encontrar una estación y el componente *CosmographTimeline* que permite filtrar el grafo por rango de fechas.

Para mejorar el rendimiento general de la web se ha utilizado la técnica de *lazy-loading*, que consiste en cargar los componentes del grafo solo cuando el usuario presiona el botón *Load graph*. En el momento que el usuario presiona el botón, se activa un *callback* de cliente de Dash. El *callback* primero lee los datos del grafo de un fichero CSV que se actualiza diariamente mediante Apache Airflow. Posteriormente se crean los nodos y las aristas y se inicializa el grafo, la barra de búsqueda y la barra de tiempo. Seguidamente se establecen los parámetros de la simulación de fuerzas y se inicia la simulación. Por último se establecen las propiedades visuales y funcionales de los nodos y aristas.

Cuando el usuario posiciona el cursor sobre un nodo, se muestra un recuadro con el nombre de la estación. Cuando el usuario hace clic en un nodo, se le redirige a la página **station** de la estación seleccionada.

3.6. Orquestación

En esta sección se describe con detalle la metodología de orquestación de tareas y flujos de trabajo de la aplicación que se han ido comentando a lo largo del capítulo.

3.6.1. Supervisord

Supervisord es un sistema de control de procesos para sistemas operativos tipo Unix, diseñado para iniciar, detener y gestionar procesos de manera sencilla y robusta.

Se ha utilizado Supervisord como gestor de procesos para ejecutar en modo demonio como se ha mencionado previamente el sistema de recepción de paquetes APRS. Se ha establecido en la configuración de Supervisord que el sistema de recepción de paquetes se ejecute en el arranque del sistema y siempre después de que la interfaz de red esté disponible. En caso de fallo del sistema de recepción, Supervisord reiniciará el sistema y registrará la causa del fallo.

3.6.2. Apache Airflow

Apache Airflow es una plataforma de orquestación de tareas y flujos de trabajo. Es similar a Cron de Unix, pero permite al usuario una mayor personalización y un mayor control sobre las tareas que controla. La unidad de ejecución en Apache Airflow es el DAG (*Directed Acyclic Graph*). Las tareas se definen en archivos individuales y al igual que en otros sistemas se pueden definir dependencias entre tareas.

Apache Airflow se ha utilizado para orquestar todas las tareas que permiten que APRSINT funcione correctamente se presenta en la Tabla 3.1 la lista de los DAG que se ejecutan diariamente.

Hora de ejecución	DAG	Descripción
3:00 am	upload_files	Sube todos los archivos al S3 aprsinput
5:00 am	download_files	Descarga todos los archivos a la carpeta local
7:00 am	insert_database	Inserta todos los mensajes en la BBDD
9:00 am	cache_data	Precalcula en archivos los datos para la web

Tabla 3.1: Esquema de orquestación con Apache Airflow.

Por defecto Apache Airflow utiliza una base de datos SQLite para almacenar los metadatos de las tareas y los flujos de trabajo. Sin embargo, para este proyecto se ha optado por utilizar una base de datos PostgreSQL para almacenar los metadatos de Apache Airflow. Esto se ha hecho para mejorar la escalabilidad y sobre todo la fiabilidad del sistema.

Se ha creado también un usuario en la base de datos de PostgreSQL con permisos de lectura y escritura para Apache Airflow. Se ha configurado Apache Airflow para que utilice esta base de datos y este usuario en el archivo de configuración de Apache Airflow.

Otra configuración importante que se ha realizado en Apache Airflow es el cambio del ejecutor por defecto de SequentialExecutor a LocalExecutor. El ejecutor por defecto de Apache Airflow es SequentialExecutor, que ejecuta las tareas de manera secuencial en un solo hilo. Sin embargo, no se recomienda el uso de este ejecutor⁵ por lo que para mejorar el rendimiento y la escalabilidad del sistema se ha optado por utilizar LocalExecutor, que ejecuta las tareas de manera paralela en varios hilos.

Se presenta en la Figura 3.11 la interfaz de Apache Airflow con los DAG's que se ejecutan diariamente.

⁵<https://airflow.apache.org/docs/apache-airflow/stable/core-concepts/executor/sequential.html>

DAGs

All	Active	Paused	Running	Failed	Filter DAGs by tag	Search DAGs	Auto-refresh		
	4	0	0	0			<input checked="" type="checkbox"/>		
	DAG	Owner	Runs	Schedule	Last Run	Next Run	Recent Tasks	Actions	Links
<input checked="" type="checkbox"/>	cache_data	AprSint	36	0 9 * * *	2024-05-09, 09:00:00	2024-05-10, 09:00:00	36	▶ 🔗	...
<input checked="" type="checkbox"/>	db_insert	AprSint	36	0 7 * * *	2024-03-09, 07:00:00	2024-05-10, 07:00:00	36	▶ 🔗	...
<input checked="" type="checkbox"/>	s3_download	AprSint	36	0 5 * * *	2024-05-09, 05:00:00	2024-05-10, 05:00:00	36	▶ 🔗	...
<input checked="" type="checkbox"/>	s3_upload	AprSint	36	0 3 * * *	2024-05-09, 03:00:00	2024-05-10, 03:00:00	36	▶ 🔗	...

Showing 1-4 of 4 DAGs

Version: v2.8.1
Git Version: .release:c0ffa9c5d96625c68ded9562632674ed366b5eb3

Figura 3.11: Interfaz de Apache Airflow (elaboración propia).

3.7. Alojamiento

Para alojar tanto la aplicación web como el sistema de recepción de paquetes y la base de datos de APRSINT se ha optado por utilizar una Raspberry Pi 4 conectada a un internet residencial.

Para evitar los problemas que suponen las direcciones dinámicas se ha decidido por utilizar el servicio de DNS dinámico de no-ip que permite asignar un nombre de dominio a una dirección ip dinámica. Este sistema se ha configurado en el router de modo que cuando la dirección ip cambia, el router notifica a no-ip y este actualiza el registro DNS asociado a la ip antigua por la nueva dirección.

Se ha configurado también la redirección de puertos en el router para permitir el acceso a la Raspberry Pi desde el exterior. Los servidores que se encuentran actualmente en la Raspberry Pi son:

- **Servidor Web:** Se ha configurado un servidor web Nginx para servir la aplicación web de APRSINT.
- **Base de Datos:** Se ha configurado un servidor PostgreSQL para almacenar los metadatos de Apache Airflow y los datos de los mensajes APRS.
- **Apache Airflow:** Se ha configurado un servidor Apache Airflow para orquestar las tareas y flujos de trabajo de APRSINT.
- **Servidor SSH:** Se ha configurado un servidor SSH para permitir el acceso remoto a la Raspberry Pi y realizar todo el desarrollo.

3.7.1. Servidor Web

En esta sección se describen las tecnologías y la arquitectura del servidor web que aloja la aplicación web de APRSINT [4]. La aplicación utiliza el *framework* Flask como *backend* web, junto con Gunicorn como servidor web WSGI y Nginx como reverse proxy.

Flask

Flask es un *framework* web ligero para Python que proporciona herramientas para crear aplicaciones web de forma rápida y sencilla. En esta aplicación, Flask se utiliza como el *framework backend* para la aplicación de Dash. Proporciona las rutas y la lógica de negocio necesarias para la aplicación.

Gunicorn

Gunicorn es un servidor HTTP WSGI (*Web Server Gateway Interface*) para Python. Se utiliza para servir la aplicación Flask de forma eficiente y escalable. Gunicorn gestiona múltiples procesos de forma paralela, lo que mejora el rendimiento de la aplicación.

En APRSINT se ha configurado Gunicorn para ejecutar la aplicación Flask en modo demonio y con 4 procesos de trabajo. Gunicorn crea un *socket* UNIX en el que escucha las solicitudes HTTP y las reenvía a la aplicación Flask para su procesamiento.

Nginx

Nginx es un servidor web de código abierto que en este caso se ha utilizado como proxy inverso. Actúa como intermediario entre los clientes y el servidor Flask / Gunicorn. Nginx se encarga de recibir las solicitudes HTTP, dirigirlas al servidor Gunicorn y devolver las respuestas a los clientes. Nginx maneja tareas como el balanceo de carga, el almacenamiento en caché la compresión de archivos y la seguridad. Se ha configurado Nginx para servir la aplicación web de APRSINT en el puerto 80 (local).

El diagrama de la arquitectura del servidor web se muestra en la Figura 3.12. Como se puede observar, la solicitud HTTP del cliente pasa a través de Nginx, que actúa como un proxy inverso y la reenvía a Gunicorn, donde se ejecuta la aplicación Flask. Una vez que Flask procesa la solicitud, la respuesta se envía de vuelta al cliente a través de Nginx.

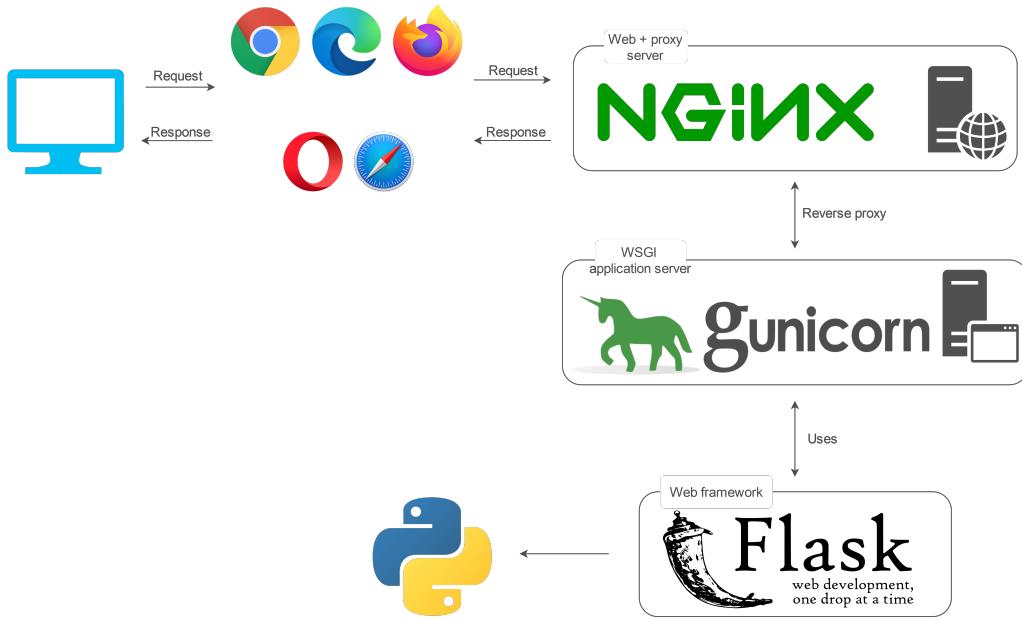


Figura 3.12: Arquitectura del servidor web de APRSINT (elaboración propia).

Flujo de una petición HTTP

A continuación se describe cómo se maneja una solicitud HTTP desde se recibe en Nginx hasta que se envía la respuesta de vuelta al cliente. A continuación se detalla el proceso paso a paso:

1. Un usuario busca en su navegador la aplicación web de APRSINT.
2. La solicitud llega al servidor Nginx, que actúa como el servidor frontal.
3. Nginx examina la solicitud y determina que está destinada a la aplicación Dash, por lo que la reenvía al servidor Gunicorn.
4. Gunicorn recibe la solicitud y la asigna a uno de sus 4 procesos de Flask para su procesamiento.
5. La aplicación Flask ejecuta la lógica correspondiente a la solicitud, que incluyen el acceso a la base de datos, y la generación del contenido de la página.
6. Una vez que la lógica de la aplicación Flask se ha completado, se genera una respuesta.
7. La respuesta se envía de vuelta a Gunicorn, que la reenvía a Nginx.
8. Nginx recibe la respuesta y la devuelve al cliente que realizó la solicitud original.

Este proceso asegura que las solicitudes sean manejadas de manera eficiente y escalable, garantizando una experiencia de navegación más fluida.

3.7.2. Protección ante ataques

En el desarrollo de la aplicación, se ha priorizado la seguridad como un aspecto fundamental. Se han implementado varias medidas [7] para proteger el sistema contra posibles ataques externos e internos. A continuación, se detallan algunas de las estrategias de seguridad utilizadas:

- **Uso del protocolo SSH:** Se ha optado por utilizar exclusivamente el protocolo SSH (*Secure Shell*) para la administración y desarrollo remoto de la Raspberry Pi. El SSH proporciona una conexión segura y cifrada, lo que reduce el riesgo de interceptación de datos sensibles durante la comunicación.
- **Creación de un usuario no root:** Se ha creado un usuario no *root* con privilegios limitados para las operaciones diarias. Esto ayuda a mitigar el impacto de posibles vulnerabilidades en el sistema, ya que el acceso de usuario se limita a las funciones esenciales.
- **Cambio del puerto SSH por defecto:** Como medida adicional de seguridad, se ha cambiado el puerto predeterminado del servicio SSH. Esto dificulta los intentos de acceso no autorizado, ya que los atacantes suelen escanear los puertos estándar en busca de vulnerabilidades.
- **Desactivación del acceso por contraseña:** Se ha desactivado el acceso por contraseña al servidor SSH, lo que significa que los usuarios deben autenticarse mediante claves SSH. Este enfoque reduce el riesgo de ataques de fuerza bruta dirigidos a contraseñas débiles.
- **Instalación de fail2ban:** Se ha instalado y configurado fail2ban, una herramienta de prevención de intrusiones que monitoriza los registros del sistema en busca de intentos de acceso fallidos. Fail2ban bloquea automáticamente las direcciones ip de los atacantes después de un número especificado de intentos fallidos, lo que ayuda a proteger contra ataques de fuerza bruta.
- **Configuración de un firewall UFW:** Se ha configurado el firewall UFW (*Uncomplicated Firewall*) para controlar el tráfico de red entrante y saliente. El firewall UFW bloquea el acceso a los puertos no utilizados y permite especificar reglas de acceso para servicios específicos, lo que ayuda a prevenir intrusiones no autorizadas.
- **Actualizaciones regulares del sistema:** Se ha establecido un *cronjob* para que actualice el sistema operativo y las aplicaciones instaladas regularmente. Las actualizaciones de seguridad y los parches de *software* son esenciales para proteger el sistema contra vulnerabilidades conocidas.

Mediante estas estrategias, se pretende minimizar los riesgos de seguridad y proteger la integridad y la confidencialidad de los datos de la aplicación.

3.8. Casos de uso

En esta sección se describen los casos de uso de la aplicación web de APRSINT, incluyendo los actores, las funcionalidades y los escenarios de uso. Para esto, se han identificado a dos posibles actores que utilizarían la aplicación web de APRSINT:

- **Investigador de OSINT:** Este actor utiliza funciones avanzadas de filtrado para analizar y extraer inteligencia de los datos APRS. El investigador busca obtener información detallada a partir de los mensajes transmitidos, utilizando herramientas para explorar patrones, tendencias y relaciones en los datos.
- **Radioaficionados curiosos:** Son entusiastas de la radioafición que visitan APRSINT principalmente por curiosidad y para explorar el mundo de APRS de una manera más interactiva y visual. Aunque pueden no tener experiencia en análisis avanzado de datos, están interesados en descubrir nuevas estaciones, rutas de seguimiento y eventos en tiempo real utilizando la plataforma.

Se han identificado las siguientes funcionalidades y escenarios de uso para la aplicación web de APRSINT:

- **Visualización de estaciones APRS:** Los usuarios de la aplicación pueden visualizar las estaciones APRS en un mapa interactivo, proporcionando una representación geográfica del panorama APRS.
- **Filtrado de estaciones APRS:** Permite a los usuarios filtrar las estaciones APRS según criterios específicos, como rango de fechas, país, contenido del mensaje, tipo de estación y SSID, para obtener información relevante.
- **Búsqueda de mensajes:** Se pueden buscar mensajes emitidos por las estaciones que contengan una palabra o frase específica, facilitando la localización de información relevante en la red APRS.
- **Visualización de información detallada de una estación APRS:** Esta funcionalidad permite acceder a información detallada de una estación APRS en particular, incluyendo datos de identificación, posiciones reportadas y mensajes emitidos, para un análisis más profundo.
- **Visualización del grafo de estaciones APRS:** Los usuarios pueden visualizar un grafo que representa las conexiones entre las estaciones APRS, proporcionando una visualización clara de la estructura de la red APRS.
- **Exploración del grafo de estaciones APRS:** Permite la exploración del grafo de estaciones APRS interactuando con los nodos y enlaces, obteniendo información adicional sobre las relaciones entre las estaciones.
- **Descarga de datos:** Facilita la descarga de los datos de las estaciones APRS en formato CSV, facilitando su análisis fuera de la plataforma APRSINT.

Estas son las funcionalidades y escenarios de uso que se han diseñado para satisfacer las necesidades de los dos actores y proporcionar una experiencia completa en APRSINT.

Capítulo 4

Conclusiones y trabajo futuro

4.1. Conclusiones

En este trabajo se ha presentado un sistema completo centrado en la adquisición, procesamiento, análisis y visualización de datos APRS. A lo largo del proyecto, se ha seguido una metodología de trabajo dinámica y flexible que ha permitido adaptarse a los problemas encontrados durante el desarrollo.

Considero que los objetivos planteados al inicio del proyecto se han cumplido con éxito. La solución propuesta se ha desarrollado siguiendo un enfoque modular y escalable, lo que permite una fácil extensión y adaptación a futuras necesidades. Además, se ha priorizado la usabilidad y la experiencia del usuario.

4.1.1. Éxitos del proyecto

Durante el desarrollo de este proyecto, se han alcanzado varios éxitos significativos. En primer lugar, se ha logrado implementar un sistema completo que cumple con los requisitos establecidos al inicio del proyecto. Esto incluye la adquisición eficiente de datos APRS, un procesamiento y análisis detallado y una visualización clara de la información. Además, el volumen de datos que maneja APRSINT está a la par con otras soluciones comerciales, lo que demuestra la robustez de la implementación a pesar de ser Open-Source.

Además, el enfoque modular y escalable adoptado en el desarrollo de la solución ha demostrado ser efectivo, ya que a lo largo del desarrollo ha facilitado la incorporación de nuevas funcionalidades y la adaptación sencilla a los cambios que han ido ocurriendo a lo largo del proyecto.

4.1.2. Desafíos superados

A pesar de los éxitos alcanzados, también se han enfrentado varios desafíos y obstáculos durante el desarrollo de este proyecto.

Uno de los principales desafíos encontrados fue el intento inicial de utilizar dispositivos RTL-SDR para la adquisición de datos APRS, lo que resultó en problemas técnicos y de estabilidad¹. Este desafío se ha superado utilizando la red APRS-IS para la adquisición de datos lo que ha resultado ser mucho más eficiente y confiable y sobre todo, mucho más versátil, ya que se pueden obtener datos de cualquier parte del mundo sin necesidad de tener un dispositivo físico en la zona de interés.

Otro de los desafíos que ha habido que superar ha sido el de la limitación del *hardware*. Como se ha mencionado anteriormente, la Raspberry Pi utilizada no podía soportar el procesamiento masivo de datos APRS además de la carga que tenía por los otros módulos de la aplicación. Para superar esta limitación, se implementó una solución que transfiere el procesamiento al cloud de AWS. Esto ha acelerado el procesamiento y ha aligerado la carga de cómputo de la Raspberry Pi.

Quizá el desafío más importante del proyecto ha sido la gestión de recursos y tiempo, ya que este ha sido un proyecto unipersonal ambicioso. La óptima gestión y planificación del tiempo han resultado ser fundamentales para superar este desafío y completar el proyecto con éxito.

4.1.3. Aprendizajes del proyecto

El proyecto ha sido una experiencia de aprendizaje valiosa en multitud de aspectos. En primer lugar, he adquirido una comprensión más profunda de los sistemas de comunicación de radioaficionados y de la tecnología APRS. Además he trabajado con tecnologías y herramientas nuevas para mí, como puede ser el cloud AWS o sistemas de orquestación como Apache Airflow, que me han permitido ampliar mis habilidades técnicas. Finalmente considero que el aprendizaje más importante ha sido el de la gestión de un proyecto de esta envergadura, desde la planificación inicial hasta la implementación y la entrega final.

4.1.4. Limitaciones del proyecto

A pesar de los éxitos y logros alcanzados, también hay algunas limitaciones en la solución propuesta. Una de las limitaciones más importantes es la dependencia de la red APRS-IS para la adquisición de datos, lo que puede limitar la cobertura y la precisión de los datos recopilados. Además, la solución actual no es capaz de manejar grandes volúmenes de datos en tiempo real, lo que puede limitar su utilidad en entornos de alta demanda.

¹En las fechas en las que comenzó el proyecto, el *Digipeater* de la zona estaba apagado y no se recibía nada de información

4.2. Trabajo futuro

En resumen, considero que el proyecto ha sido satisfactorio y que la solución propuesta tiene un gran potencial para mejorar la experiencia de los usuarios de APRS y enriquecer la información disponible. Para el trabajo futuro, se pueden considerar varias áreas de mejora y expansión.

Una posible área de desarrollo futuro es la integración de tecnologías, como el aprendizaje automático e inteligencia artificial, para mejorar la precisión y las capacidades de análisis del sistema, facilitando la detección de patrones y otros elementos interesantes. Además, se puede explorar la posibilidad de ampliar la funcionalidad de la solución para incluir características adicionales solicitadas por los usuarios o identificadas durante el uso continuo del sistema.

Además, el sistema puede ser extendido fácilmente para trabajar con otros protocolos de radio, como CATS² o LoRa, lo que ampliaría su alcance y utilidad en diferentes entornos de comunicación. También se puede mejorar la capacidad de filtrado del sistema, permitiendo a los usuarios definir y aplicar una variedad más amplia de filtros para adaptarse a sus necesidades específicas de análisis de datos.

Otra área de mejora importante es la expansión de las fuentes de datos APRS utilizadas por el sistema. Actualmente, el sistema utiliza los datos provenientes de la red APRS-IS, lo que podría no capturar todos los paquetes APRS transmitidos si no llegan a un I-Gate. Para abordar esto, se pueden explorar otras fuentes de datos, como la integración con dispositivos RTL-SDR, la integración directa con estaciones de radio APRS locales o la colaboración con otros sistemas APRS existentes para mejorar la cobertura y la precisión de los datos recopilados.

Finalmente, se puede considerar escalar el sistema para utilizar *hardware* más capaz, lo que mejoraría su rendimiento y capacidad de procesamiento. Esto implicaría el uso de servidores más potentes o la distribución de la carga de procesamiento en múltiples nodos, lo que permitiría manejar volúmenes de datos más grandes y complejos de manera más eficiente.

²CATS es un proyecto con el objetivo de mejorar el sistema APRS

Introduction

“People think of education as something that they can finish.”
— Isaac Asimov

4.3. What is APRS?

APRS or *Automatic Packet Reporting System* is a real-time digital communications system that allows the exchange of information between amateur radio stations.

In general, APRS is used for vehicle tracking, transmission of text messages, broadcasting of weather information and to aid communication in emergency situations, although due to the flexibility of the protocol it can be used in any other situation.



Figure 4.1: APRS logo.

4.4. Main APRS features

APRS has several features that make it useful and versatile in the field of amateur radio communication, among which are the following:

- **Working frequency:** APRS operates in the VHF band, specifically on the 144.80 MHz frequency in Europe, although other regions of the world use different frequencies as shown in the figure below. Figura 4.2.
- **Transmission mode:** APRS uses as a transmission mode individual packets that have to follow an established format, which greatly facilitates the adoption and integration of this technology.
- **APRS retransmission policy:** In the APRS system, a packet of information is transmitted multiple times, gradually decreasing the emission frequency of

these transmissions as time progresses. This method aims to maximize the packet reception rate.

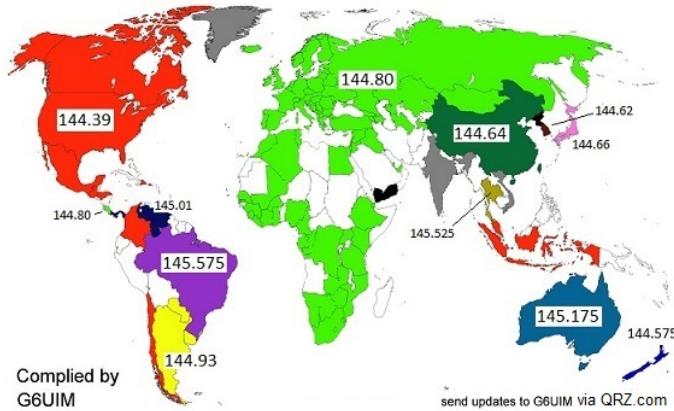


Figure 4.2: APRS emission frequency around the globe.

4.5. Main APRS applications

- **GPS position reports:** APRS allows users to send their geographic location in the form of coordinates, obtained through GPS systems, which facilitates the tracking of vehicles or people in real time.
- **Weather data:** Weather stations often use APRS messages to report different data such as temperature, humidity or barometric pressure, including updated and useful information for various applications.
- **Internet integration:** Through the APRS-IS system, APRS messages are accessible over the Internet through different nodes, extending the reach and scope of this system.
- **Uso en Emergencias:** In emergency situations, APRS is a vital tool for broadcast communication and tracking of resources and personnel.

4.6. History and development of APRS

The APRS system was born in the 1980s by Bob Bruninga, an engineer working at the U.S. Naval Academy. Bruninga created the first implementation of APRS on an Apple II computer in 1982 for the purpose of mapping Navy position reports on the high seas.[2]

The first real use of APRS was in 1984, when Bruninga developed a more advanced version on a VIC-20 to report the position and status of horses in an endurance race.

Over the next few years, Bruninga continued to refine the system, which he later dubbed the CETS (*Connectionless Emergency Traffic System*).

After a series of *Federal Emergency Management Agency* (FEMA) exercises using CETS, the system was moved to the IBM PC. During the 1990s, CETS (now known as the *Automatic Position Reporting System*) continued to evolve.

As GPS technology became more widely available, the term “Positioning” was replaced by “Package” to better represent the broader capabilities of the system and emphasize its uses beyond mere position reporting.



Figure 4.3: Bob Bruninga (creator of the APRS system).

4.7. Proposal and Objectives

A comprehensive solution is proposed to encompass the acquisition, processing, visualization, and analysis of APRS data.

4.7.1. Objectives

The aim of the project is to develop a solution that enhances user experience and enriches available APRS information through the integration of data from various open and accessible sources.

- **Enhanced User Experience:** The solution will have an intuitive, fast, and useful interface that facilitates navigation and interaction with the data.
- **Advanced Analysis:** The solution will have advanced tools for detailed visualization of APRS traffic, precise filtering, and data analysis, aiming to extract intelligence from received messages.
- **Supplementation:** The solution is not aimed at replacing aprs.fi, aprs.to, or other similar platforms but to offer an alternative with complementary functionalities to add valuable insights to the user.
- **Information Enrichment:** Data from various open and available sources will be collected to offer a more comprehensive view of APRS traffic and its users, thus improving the quality of information available to users.

4.7.2. Benefits

- **Better Understanding of APRS Traffic:** Users will be able to obtain more detailed and actionable information from transmitted messages, allowing them to better understand APRS traffic.
- **More Effective Decision Making:** Integrated data analysis will help users make more informed decisions based on the information received, thereby improving the effectiveness of their actions.
- **Flexibility:** The self-hosting option will allow users to have greater control over their data and privacy, providing additional flexibility in its management.

4.8. Requirements

The proposed solution must meet a series of essential requirements to ensure its value and accessibility:

- **Low Cost:** The solution must be economical to ensure accessibility to a wide range of users, including those with limited budgets.
- **Hosting Flexibility:** The solution must offer self-hosting capability; users should be able to choose to host it on their own servers or use the solution in the cloud according to their preferences and specific requirements.
- **Extensibility:** The solution must be extensible, meaning it should allow the integration of new functionalities and expansion of its capacity according to the changing needs of users.
- **Improved Filtering:** More precise and flexible filtering should be provided compared to existing platforms, enabling users to obtain relevant and useful information from APRS data to complement that already provided by alternatives.
- **Integrated Data Analysis:** The solution must include integrated data analysis to help users better understand the information received via APRS and derive valuable insights and conclusions from transmitted messages.

4.9. Work plan

For the execution of this project, the following work plan has been proposed, which is divided into the following phases:

- **Phase 1:** Research and Requirements Analysis.
- **Phase 2:** Design and Planning.
- **Phase 3:** Implementation and Development.

■ Phase 4: Report Writing.

Figure 4.4 shows a Gantt chart with the start and end dates of each of the activities carried out throughout the project.

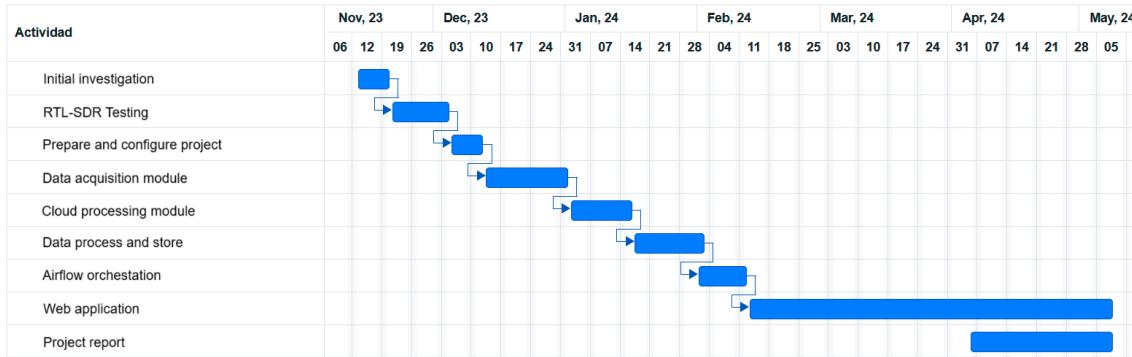


Figure 4.4: Proposed work plan.

Regular meetings have also been held with the TFG supervisors to review progress and discuss any issues encountered during development.

Conclusions and future work

4.10. Conclusions

This work has presented a comprehensive system focused on the acquisition, processing, analysis, and visualization of APRS data. Throughout the project, a dynamic and flexible working methodology has been followed, allowing for adaptation to the challenges encountered during development.

The objectives set at the beginning of the project have been successfully achieved. The proposed solution has been developed following a modular and scalable approach, enabling easy extension and adaptation to future needs. Additionally, usability and user experience have been prioritized.

4.10.1. Project successes

Several significant successes have been achieved during the development of this project. Firstly, a complete system fulfilling the requirements established at the project's outset has been implemented. This includes efficient acquisition of APRS data, detailed processing and analysis, and clear visualization of information. Additionally, the volume of data handled by APRSINT is on par with other commercial solutions, demonstrating the robustness of the implementation despite being Open-Source.

Furthermore, the modular and scalable approach adopted in the solution's development has proven effective. Throughout development, it facilitated the incorporation of new functionalities and straightforward adaptation to changes that occurred during the project.

4.10.2. Challenges overcome

Despite the successes achieved, several challenges and obstacles were encountered during this project's development.

One of the main challenges encountered was the initial attempt to use RTL-

SDR devices for APRS data acquisition, resulting in technical and stability issues³. This challenge was overcome by using the APRS-IS network for data acquisition, which proved to be much more efficient, reliable, and versatile, allowing data to be obtained from anywhere in the world without the need for a physical device in the area of interest.

Another challenge that had to be overcome was hardware limitations. As mentioned earlier, the Raspberry Pi used could not support massive APRS data processing in addition to the workload from other application modules. To overcome this limitation, a solution was implemented to offload processing to the AWS cloud. This accelerated processing and lightened the Raspberry Pi's processing load.

Perhaps the most significant challenge of the project was resource and time management, as this was an ambitious solo project. Good time management and planning proved to be crucial in overcoming this challenge and successfully completing the project.

4.10.3. Project learnings

The project has been a valuable learning experience in many aspects. Firstly, a deeper understanding of amateur radio communication systems and APRS technology has been acquired. Additionally, working with new technologies and tools, such as AWS cloud or orchestration systems like Apache Airflow, has allowed for the expansion of technical skills. Finally, the most important learning has been project management of this magnitude, from initial planning to implementation and final delivery.

4.10.4. Project limitations

Despite the successes and achievements, there are also some limitations in the proposed solution. One of the most significant limitations is the dependency on the APRS-IS network for data acquisition, which may restrict the coverage and accuracy of the collected data. Additionally, the current solution is not capable of handling large volumes of real-time data, which may limit its utility in high-demand environments.

4.11. Future work

In summary, I consider the project a success, and the proposed solution has great potential to enhance the APRS user experience and enrich available information. For future work, several areas of improvement and expansion can be considered.

³At the beginning of the project, the Digipeater in the area was offline, and no information was received

One potential area for future development is the integration of technologies such as machine learning and artificial intelligence to enhance the system's accuracy and analysis capabilities, facilitating pattern detection and other interesting elements. Additionally, exploring the possibility of expanding the solution's functionality to include additional features requested by users or identified during continued system use is worthwhile.

Moreover, the system can be easily extended to work with other radio protocols, such as CATS⁴ or LoRa, which would broaden its scope and utility in different communication environments. Improving the system's filtering capacity, allowing users to define and apply a wider variety of filters to suit their specific data analysis needs, is also an area for enhancement.

Another significant area for improvement is expanding the APRS data sources used by the system. Currently, the system relies mainly on data from the APRS-IS network, which may not capture all transmitted APRS packets if they do not reach an I-Gate. To address this, exploring other data sources, such as integration with rtl-sdr devices, direct integration with local APRS radio stations, or collaboration with other existing APRS systems to enhance data coverage and accuracy, is essential.

Finally, considering scaling the system to use more capable hardware would improve its performance and processing capacity. This could involve using more powerful servers or distributing processing load across multiple nodes, allowing for more efficient handling of larger and more complex data volumes.

These areas of improvement and expansion represent exciting opportunities to continue developing and enhancing the proposed solution, ensuring its relevance and usefulness in an ever-evolving technological environment.

⁴CATS is a project aimed at improving the APRS system

Bibliografía

Citadme diciendo que me han citado mal.
Groucho Marx

- [1] ANGELAKOS, J. The state of (full) text search in postgresql 12. <https://www.postgresql.eu/events/fosdem2020/sessions/session/2890/slides/273/FTS.pdf>, 2020. Acceso: 02-02-2024.
- [2] BOB BRUNINGA. Articles.txt. <http://www.aprs.org/APRS-docs/ARTICLES.TXT>, 1999. Acceso: 05-01-2023.
- [3] HESSU. How aprs paths work. <https://blog.aprs.fi/2020/02/how-aprs-paths-work.html>, 2020. Acceso: 28-04-2024.
- [4] JUSTIN ELLINGWOOD, KATHLEEN JUELL. How to serve flask applications with gunicorn and nginx on ubuntu 18.04. <https://www.digitalocean.com/community/tutorials/how-to-serve-flask-applications-with-gunicorn-and-nginx-on-ubuntu-18-04>, 2021. Acceso: 06-05-2024.
- [5] MIGUEL DARÍO, XE1UD. Que es y cómo funciona el sistema aprs. <https://crecj.org/que-es-y-como-funciona-el-sistema-aprs>, 2024. Acceso: 02-02-2024.
- [6] PARTHIBAN MARIMUTHU. Stop storing data in csvs vs feather. <https://www.analyticsvidhya.com/blog/2022/06/stop-storing-data-in-csvs-vs-feather>, 2022. Acceso: 24-05-2024.
- [7] PATRICK FROMAGET. 17 security tips to protect your raspberry pi like a pro. <https://raspberrytips.com/security-tips-raspberry-pi/>, unknown. Acceso: 17-03-2024.
- [8] THE APRS WORKING GROUP. Automatic position reporting system aprs protocol reference protocol version 1.0. <https://www.ui-view.net/files/APRS101.pdf>, 2000. Acceso: 05-01-2024.
- [9] WIKIPEDIA. Aprs. https://en.wikipedia.org/wiki/Automatic_Packet_Reportin System, 2024. Acceso: 12-11-2023.

Acrónimos

APRS Automatic Packet Reporting System.

SBC Single Board Computer.

APRS-IS Automatic Packet Reporting System Internet Service.

RTL-SDR Realtek Software Defined Radio.

AWS Amazon Web Services.

JSON JavaScript Object Notation.

SSH Secure SHell.

HTTP HyperText Transfer Protocol.

WSGI Web Server Gateway Interface.

OSINT Open Source Intelligence.

TNC Terminal Node Controller.

GPS Global Positioning System.

API Application Programming Interface.

ORM Object-Relational Mapping.

SQL Structured Query Language.

DDNS Dynamic Domain Name System.