



CARRERA DE ESPECIALIZACIÓN EN INTERNET DE LAS COSAS

MEMORIA DEL TRABAJO FINAL

Desarrollo de un dispositivo embebido localizador y botón antipánico

Autor:
Ing. Erik Hromek

Director:
Mg. Ing. Lucas Dórdolo (FIUBA)

Jurados:
Esp. Ing. Pedro Rosito (FIUBA)
Mg. Lic. Leopoldo Zimperz (FIUBA)
Mg. Ing. Gonzalo Sanchez (FIUBA)

*Este trabajo fue realizado en la Ciudad de Quilmes,
entre agosto de 2023 y agosto de 2024.*

Resumen

En la presente memoria se realiza una descripción del diseño y desarrollo de un prototipo de botón antipánico y una plataforma web básica para la recepción de alertas. Este trabajo hace énfasis en desarrollar un dispositivo que pueda ser construido con módulos de bajo costo y adquiribles en el mercado local, con foco en el consumo energético.

Para su desarrollo se aplicaron gran parte de los conocimientos adquiridos en la carrera tales como el desarrollo sobre sistemas embebidos, implementación de protocolos de comunicación y uso de tecnologías web.

Agradecimientos

Índice general

Resumen	I
1. Introducción general	1
1.1. Planteo básico	1
1.1.1. Botones antipánico	1
1.2. Motivación	2
1.3. Objetivos y alcance	3
1.4. Soluciones comerciales similares	4
1.4.1. LK109	4
1.4.2. TK102B	5
2. Introducción específica	7
2.1. Particularidades de los sistemas de botones antipánico	7
2.1.1. Características del hardware	7
2.1.2. Características de los sistemas web	8
2.2. Componentes de hardware utilizados	8
2.2.1. Microcontrolador ESP32	8
2.2.2. Módulo GPS NEO-6M	9
2.2.3. Módulo GSM SIM800L	10
2.2.4. Batería 18650 y chip cargador TP4056	11
2.3. Componentes de software utilizados	11
2.3.1. Django REST framework	11
2.3.2. Angular	12
2.3.3. PostgreSQL	13
2.3.4. Heroku	13
2.4. Herramientas y servicios de software utilizados	13
2.4.1. Docker	13
2.4.2. Visual Studio Code	13
2.4.3. PyGPSClient	14
2.4.4. Moserial	14
2.4.5. Twilio	15
3. Diseño e implementación	17
3.1. Arquitectura del sistema	17
3.2. Desarrollo de módulos de hardware	18
3.2.1. Tareas del dispositivo embebido	20
3.2.2. Optimizaciones de energía aplicadas	23
3.3. Desarrollo de módulo de backend	24
3.3.1. Servicios desarrollados	28
3.4. Desarrollo de módulo de frontend	30
3.4.1. Particularidades del desarrollo	31
3.5. Integración	33

4. Ensayos y resultados	37
4.1. Pruebas funcionales del hardware	37
5. Conclusiones	39
5.1. Conclusiones generales	39
5.2. Próximos pasos	39
Bibliografía	41

Índice de figuras

1.1. Diagrama en bloques del dispositivo y el sistema web	1
1.2. Imagen del botón LK109	4
1.3. Imagen del botón TK102B	5
2.1. Microcontrolador ESP32-S	9
2.2. Módulo GPS NEO-6M con una antena externa	10
2.3. Módulo SIM800L	11
2.4. Batería 18650	11
2.5. Circuito Mp4056	11
2.6. Aplicación PyGPSClient	14
2.7. Aplicación Moserial	15
3.1. Arquitectura general del sistema	17
3.2. Esquema de componentes de hardware del ESP32	19
3.3. Diagramas de flujo de tarea inicial, y bucles del módulo GSM y GPS	21
3.4. Diagramas de flujo de las tareas de <i>setup</i> de energía	21
3.5. Diagramas de flujo de las tareas de control y almacenamiento . .	22
3.6. Diagrama de flujo del <i>callback</i> por pulsación	23
3.7. Estructura de archivos de módulo de <i>Users</i>	25
3.8. Estructura de archivos de módulo de <i>Alerts</i>	25
3.9. Diagrama en bloques del <i>backend</i>	26
3.10. Serializador de la entidad <i>Beneficiary</i>	27
3.11. Serializador de la entidad <i>Beneficiary</i>	27
3.12. Modelo de datos	28
3.13. Configuración de <i>middlewares</i> para la autenticación de la API . .	29
3.14. Estructura del <i>frontend</i>	30
3.15. Estructura del <i>frontend</i>	32
3.16. Comparativa entre <i>snake case</i> y <i>camel case</i>	32
3.17. Esquema de integración entre botón antipánico y Twilio	33
3.18. Esquema de integración entre Twilio y la API	34
3.19. Esquema de integración entre <i>frontend</i> y la API	35

Índice de tablas

1.1. Resumen de objetivos	3
1.2. Tabla comparativa	6
2.1. Tabla comparativa	9
2.2. Tabla comparativa	10
2.3. Tabla comparativa	12
2.4. Tabla comparativa	13
3.1. Tabla de bibliotecas	19
3.2. Tabla de bibliotecas	25
3.3. <i>Servicios</i>	29
3.4. <i>Servicios</i>	31
3.5. <i>Bibliotecas importadas</i>	33

Capítulo 1

Introducción general

Este capítulo presenta una introducción a los conceptos básicos de la temática del trabajo y explica las principales causas que motivaron el desarrollo del proyecto, junto con los objetivos que se buscaron alcanzar.

1.1. Planteo básico

Existen en la actualidad diferentes soluciones en lo que respecta a la prevención o seguridad ciudadana [1]. Estos sistemas centralizan información en tiempo real de cámaras de seguridad, móviles de las fuerzas de seguridad, entre otros. Además, suelen incorporar alguna funcionalidad para que un ciudadano, ante una situación de emergencia, pueda dar aviso a las autoridades. El mecanismo para notificar puede ser una aplicación móvil, un dispositivo geolocalizador u otra alternativa pero el objetivo es el mismo. En la figura 1.1 se pueden observar los componentes de alto nivel del sistema web y del dispositivo geolocalizador.

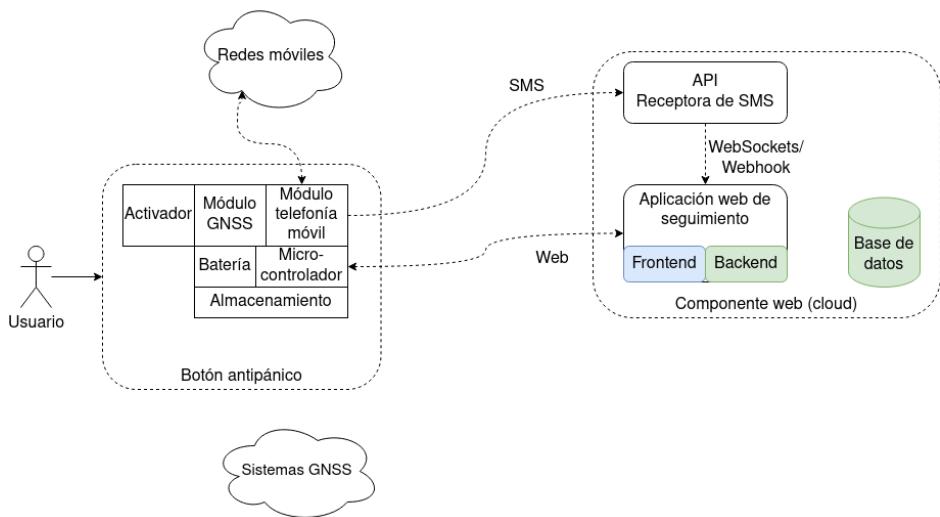


FIGURA 1.1. Diagrama en bloques del dispositivo y el sistema web

1.1.1. Botones antipánico

Un botón antipánico hace referencia a un dispositivo pequeño y portátil, alimentado por batería, que permite a una persona, ante una situación de emergencia, presionar un pulsador con el objetivo de generar una alerta y que esta sea recepcionada en un sistema de monitoreo.

Estos dispositivos generalmente son brindados por las autoridades, aunque también pueden ser provistos por empresas de seguridad privadas. Sus beneficiarios suelen ser:

- Personas en situación de violencia de género.
- Personas en situación de violencia intrafamiliar.
- Establecimientos donde pueden ocurrir situaciones de violencia y es necesario dar rápidamente aviso a las autoridades.
- Agente o sereno cuyo recorrido debe ser registrado.

Entre otros.

Entre sus funcionalidades, suelen poseer:

- Conectividad a la red de telefonía móvil (*Global System for Mobile communications* o GSM).
- Acceso a datos móviles (*General Packet Radio Service* o GPRS).
- Conectividad a sistemas de geoposicionamiento satelital (*Global Navigation Satellite System* o GNSS).
- Botón de disparo de alerta.
- Reporte periódico de ubicación por mensaje de texto (SMS) o vía web.

Un detalle interesante respecto a estos dispositivos es que generalmente se comercializan en dos formatos:

- Dispositivo personal: dispositivo embebido que puede ser llevado como colgante, en la muñeca o en el bolsillo. Tiene una autonomía limitada.
- Dispositivo para móviles: también conocido como AVL o *Automatic Vehicle Location*; además de contar con antenas externas para mayor precisión, cuenta con cables para conexión a la batería de los vehículos, permitiendo tener una mayor autonomía.

En relación a este trabajo, se planteó el desarrollo del prototipo de un botón geolocalizador y una plataforma web a modo de prueba de concepto, para la recepción de alertas.

1.2. Motivación

El autor del trabajo posee experiencia en este tipo de soluciones, principalmente en la adquisición, configuración, implementación y soporte de botones antipánico disponibles en el mercado. Se detectó y se llegó a la conclusión, respecto a los dispositivos, de que estos presentan los siguientes inconvenientes:

- Especificaciones de hardware obsoletas.
- Documentación escasa, desactualizada o incluso incorrecta en algunos casos.
- Poca fiabilidad de dispositivo (dificultad para configurarlos, mala señal en ambientes *indoor*, desconfianza al momento de utilizarlo).
- Inconsistencias entre dispositivos idénticos.

- Poca duración de batería (desde aproximadamente 24 horas hasta 3 o 4 días como máximo, cuando lo deseable para el beneficiario final del botón es de al menos 5 o 6 días).
- Precio exageradamente elevado y muchas dificultades para adquirirlos mediante proveedores, más en períodos de alta inflación.
- Nulas características de seguridad.

Se investigaron y revisaron diferentes alternativas en el mercado, así como también se trabajó con dispositivos utilizados por sistemas de la competencia, llegando a conclusiones similares: no se encuentran dispositivos aceptables cuyo costo sea accesible.

Todas estas cuestiones nombradas anteriormente son las que motivaron la formulación del proyecto, siendo necesario disponer de un dispositivo embebido funcional y un sistema web que permita recepcionar la información del botón.

Otras motivaciones menores del trabajo, pero igualmente muy relacionadas a la especialización, fueron:

- Profundizar en el uso de frameworks para *Single Web Applications* o SPAs.
- Poder aplicar los conocimientos trabajados en relación a tecnologías *cloud* y desarrollo de APIs web.

1.3. Objetivos y alcance

En relación a los objetivos de alto nivel del trabajo, estos fueron:

- Investigación y determinación de un conjunto de módulos de hardware que sean adecuados para la construcción del prototipo.
- Diseño y construcción un prototipo de botón antipánico que permita validar las decisiones tecnológicas.
- Tener fiabilidad superior sobre ciertas características, en comparación con algunos de los dispositivos disponibles en el mercado.
- Desarrollo de un sistema web a modo de prueba de concepto para el uso del dispositivo.

Respecto a los objetivos por componente, estos se presentan en la tabla 1.1.

TABLA 1.1. Resumen de objetivos del trabajo

Componente	Objetivo
Prototipo	Duración de la batería
Prototipo	Calidad de señal de GSM en ambientes cerrados y abiertos
Prototipo	Calidad de señal de GNSS en ambientes cerrados y abiertos
Prototipo	Tiempo de actualización de la ubicación (<i>time-to-first-fix</i>)
Prototipo	Envío de alertas mediante pulsador
Sistema web	Recepción de SMS de alertas
Sistema web	Alta de dispositivos
Sistema web	Visualización de alertas en tiempo real
Sistema web	Despliegue en un entorno <i>cloud</i>

No se incluyeron, dentro del alcance del trabajo, las siguientes características:

- Aplicación web productiva para gestionar las activaciones del botón antipánico. Por productiva se refiere a que pueda ser ofrecida al mercado o utilizable por usuarios encargados del monitoreo.
- Desarrollo de un dispositivo listo para reemplazar dispositivos existentes, es decir que no sea un prototipo.
- Desarrollo de un contenedor físico para el dispositivo.

Además, durante el desarrollo del trabajo, se fueron detectando diferentes cuestiones técnicas a resolver, que modificaron algunos objetivos planteados inicialmente.

1.4. Soluciones comerciales similares

Se presentan a continuación algunos modelos conocidos y utilizados en el mercado local. Estos dos modelos sufren los inconvenientes nombrados en las secciones anteriores: el motivo de fondo es que existen muchos clones de estos modelos, resultando muy difícil determinar si se está trabajando con un dispositivo original o un clon con posibles problemas [2].

1.4.1. LK109

LK109 es el nombre de un dispositivo geolocalizador que trabaja con redes GSM y GPRS, y usa *Global Positioning System* (GPS) como sistema de GNSS. Existe también una versión que trabaja con redes 3G, cuyo costo es superior. En la figura 1.2 se puede observar el dispositivo en cuestión.



FIGURA 1.2. Imagen del botón LK109

Entre sus características más interesantes se encuentran [3]:

- Indicadores de estado: batería, señal de GPS y señal de GSM.
- Posibilidad de reportar la ubicación mediante TCP o SMS.
- Reporte de ubicación en tiempo real, con una periodicidad configurable.
- Posibilidad de monitorear de forma remota el entorno del dispositivo mediante un micrófono.
- Autonomía de 240 horas o 10 días.

1.4.2. TK102B

TK102B es el nombre de otro dispositivo geolocalizador que trabaja con redes GSM y GPRS, y usa *Global Positioning System* (GPS) como sistema de GNSS. En la figura 1.3 puede observarse el diseño de este modelo de dispositivo.

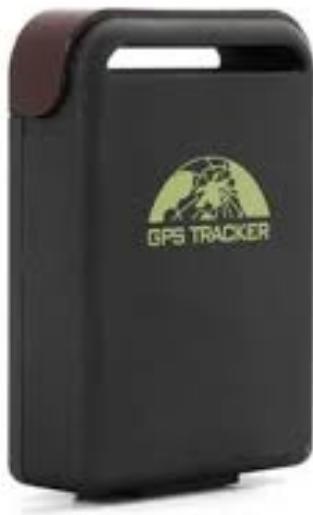


FIGURA 1.3. Imagen del botón TK102B

Entre sus características más relevantes se encuentran [4]:

- Indicadores de estado: batería, señal de GPS y señal de GSM.
- Posibilidad de reportar la ubicación mediante TCP o SMS, incluyendo la dirección aproximada, mediante geodecodificación inversa.
- Reporte de ubicación en tiempo real, con una periodicidad configurable.
- Posibilidad de monitorear de forma remota el entorno del dispositivo mediante un micrófono.
- Autonomía de 80 horas o 3 días aproximadamente.

Se presentan a continuación las principales características de estos dos modelos, contrastadas:

TABLA 1.2. Tabla comparativa de los dos modelos

Característica	LK109	TK102B
Conectividad	GSM, GPRS, SMS	GSM, GPRS, SMS
Reporte periódico	Sí	Sí
Autonomía (horas)	240	80
Indicadores de estado	Sí	Sí
Escucha remota	Sí	Sí
Protocolo de comunicación	h02	gps103

Respecto al ítem de autonomía, no se ha podido comprobar en la práctica que la duración sea la indicada (siempre es mucho menor, entre 12 y 24 horas), a excepción del uso mediante *sleep* o ahorro de batería, que no tiene utilidad para las aplicaciones de seguimiento de una persona. Esto es debido a que, en el modo de ahorro de batería, la ubicación se actualiza posterior al envío de una alerta y no antes, resultando inútil la ubicación reportada.

En relación al ítem de escucha remota, es decir la posibilidad de escuchar lo que ocurre alrededor del dispositivo, no se ha podido verificar para varios dispositivos; de todas maneras no resulta una funcionalidad útil en la mayoría de los casos de uso. Las características que se deben cumplir y validar obligatoriamente son:

- Precisión al momento de obtener la ubicación.
- Autonomía considerable de la batería.
- Rapidez para obtener señal de GSM y GNSS.

Capítulo 2

Introducción específica

En este capítulo se hace una presentación de cuestiones técnicas particulares del trabajo, una descripción de alto nivel de las tecnologías involucradas y las herramientas y servicios externos que formaron parte del desarrollo, así como también los componentes de hardware empleados.

2.1. Particularidades de los sistemas de botones antipánico

Es necesario, en primer lugar, hacer una introducción a las características que deben tener las soluciones de este tipo, junto con sus limitaciones y particularidades técnicas. Cualquier desarrollo de este tipo debe considerar las siguientes características para los módulos de hardware y software.

2.1.1. Características del hardware

Los botones antipánico y dispositivos geolocalizadores suelen ser utilizados en zonas donde la cobertura de telefonía móvil es muy baja. Es decir, se cuenta con conectividad limitada para llamadas y mensajes, y prácticamente no hay cobertura para datos móviles o es de baja tasa de transferencia. La tecnología más presente es la de GPRS o 2G[5], ya que es la red de mayor alcance, pero también la de menor ancho de banda. Esto impone una restricción respecto a la conectividad, por lo que la mayoría de los dispositivos tienen las siguientes características respecto al método de comunicación:

- Comunicación vía SMS, TCP o UDP para configuración y envío de datos.
- Comunicación vía protocolos personalizados[6] para reducir el *overhead* de datos.
- Mecanismos para almacenar datos en caso de perder conectividad.

Debido a que, dentro del alcance del trabajo se apunta a configuración y envío de mensajes de alerta, se consideró innecesario, para estas funciones, requerir otro medio adicional. No se considera apropiado usar un método de comunicación que pueda no ser fiable, como lo son los datos móviles en zonas donde la conectividad es baja.

Por otra parte, otro punto importante respecto a estos dispositivos, es su capacidad de obtener señal aceptable tanto de GSM como GNSS en ambientes *indoor* y *outdoor*. Prácticamente todos los botones antipánico poseen antenas internas y sin alimentación activa, por lo que su capacidad para adquirir un nivel de señal aceptable es limitada; esto introduce varios puntos a favor y en contra:

- No generan un consumo de energía adicional.
- Su tiempo de *startup*, es decir, hasta que adquieren un nivel de señal operativo aceptable, es elevado si están en ambientes cerrados.
- Su precisión suele ser baja en ambientes cerrados, lo cual puede ser un problema en situaciones de emergencia.
- El tiempo de actualización de la posición de la persona puede ser elevado si la antena no logra captar una buena señal.

2.1.2. Características de los sistemas web

Un sistema web, al requerir estar en comunicación con un dispositivo y mostrar su información en tiempo real, debe contar, idealmente, con las siguientes características:

- No requerir ningún software adicional para su acceso y uso más allá de un navegador web y conexión a Internet.
- Poder recibir mensajes de texto, es decir, tener un número telefónico virtual asociado.
- Enviar actualizaciones *push* al navegador del usuario en tiempo real.

Se presentan algunos desafíos respecto al segundo y tercer punto. Para la recepción de SMS, es necesario tener integrada una central de telefonía virtual o implementar un *Short Message Service Center* o SMSC, un servicio específico para el ruteo de mensajes de texto desde y hacia la web. Respecto al tercer punto, resulta obligatoria la implementación de algún mecanismo de actualización en tiempo real desde el servidor hacia el cliente, el navegador web, y no al revés, como puede ser *WebSocket*.

En algunos casos, también debería permitir comunicación bidireccional con los dispositivos, para que un operador pueda configurarlos *on-demand*. De todas maneras, debido a que está fuera del alcance del trabajo, no se planteó el desarrollo de esta característica.

2.2. Componentes de hardware utilizados

2.2.1. Microcontrolador ESP32

Se utilizó un SoC (*System on a Chip*) de la familia ESP32 de Espressif debido a que resulta una opción ideal para el desarrollo de sistemas embebidos y aplicaciones de Internet de las Cosas. Entre las características que resultaron más atractivas para su elección, se destacan[7]:

- Bajo costo y disponibilidad en el mercado local.
- Prestaciones muy interesantes: Wi-Fi, Bluetooth, modos de ahorro de energía, varios periféricos, entre otros.
- Gran soporte de la comunidad y disponibilidad de bibliotecas, junto a su documentación.
- Ya utilizado en la carrera de especialización.

En la figura 2.1 puede observarse un ESP32-S (variante comercial utilizada).



FIGURA 2.1. Microcontrolador ESP32-S

Se presentan en la tabla 2.1 algunas alternativas comparadas. De todas maneras, esta comparación no debe ser interpretada linealmente ya que hay placas como Ai Thinker A9G y TTGO T-Call V1.3 que poseen ya módulos integrados, pero todos fueron analizados para el trabajo.

TABLA 2.1. Tabla comparativa de placas de desarrollo

Característica	ESP32	A9G	LoPy 4	TTGO T-Call
Disponibilidad	Baja	No	No	Baja
Comunidad	Muy grande	Grande	Poca	Sí
Costo	Muy bajo	Moderado	Muy elevado	Muy elevado
Curva aprendizaje	Muy baja	Alta	Baja	Baja

2.2.2. Módulo GPS NEO-6M

Para la implementación del módulo de posicionamiento, se decantó por el uso del módulo de GPS NEO-6M desarrollado por la compañía u-blox. Este módulo GPS posee un rango operativo de entre 2.7 V y 3.6 V, siendo muy similar al microcontrolador elegido anteriormente. Tiene modos de ahorro de energía configurables e interfaces de comunicación UART, SPI y I²C[8]. Se analizaron otras alternativas, encontrando mucha similitud, pero hallando mayor documentación y casos de uso por parte de la comunidad para el NEO-6M. En la figura 2.2 se puede observar una variante de fabricación del NEO-6M.



FIGURA 2.2. Módulo GPS NEO-6M con una antena externa

Se muestra en la tabla 2.2 una comparativa respecto a otro módulo similar.

TABLA 2.2. Tabla comparativa de placas de desarrollo

Característica	Quectel L86	NEO-6M
Disponibilidad	Moderada	Baja
Costo	Moderado	Elevado
Sistemas GNSS	GPS	GPS, GLONASS, Galileo

2.2.3. Módulo GSM SIM800L

Para el desarrollo de la conectividad a las redes de telefonía móvil se utilizó el módulo SIM800L, un chip fabricado por la compañía Simcom. Funciona con un rango de voltaje entre 3.4 V y 4.4 V, y permite la utilización del módulo mediante comandos AT. Estas instrucciones consisten en cadenas de caracteres cortas utilizadas dentro de la industria desde la década de 1980 hasta el día de hoy para la comunicación con módulos, módems, entre otros dispositivos[9]. Posee mecanismos para optimizar el consumo de energía pero tiene picos elevados de consumo durante la transmisión. Respecto a la compatibilidad de generaciones de tecnologías de telefonía móvil, solo soporta hasta 2G[10]. De todas maneras, para los fines del trabajo, resultó suficiente. En la figura 2.3 se puede visualizar el módulo SIM800L con una antena helicoidal, aunque pueden utilizarse otros tipos de antenas con mayor ganancia de potencia.

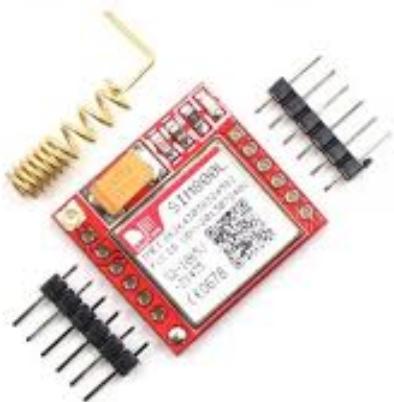


FIGURA 2.3. Módulo SIM800L

2.2.4. Batería 18650 y chip cargador TP4056

Para la alimentación del microcontrolador y sus módulos, se decidió utilizar una batería recargable del tipo 18650 de Li-ion de 2600 mAh. Este tipo de baterías tiene la ventaja de trabajar con un rango de voltaje apropiado para microcontroladores y módulos, pudiendo además tener una elevada corriente máxima de descarga (hasta 2 A aproximadamente). En la figura 2.4 se puede apreciar el formato de estas, muy similar al de las pilas domésticas. Debido a que requieren un circuito integrado para la alimentación y el control ante sobrecargas, se utilizó un módulo Tp4056 que permite utilizar una entrada micro USB de 5 V, lo cual posibilita, por ejemplo, el uso de un cargador común de teléfono celular. Este circuito se puede visualizar en la figura 2.5.



FIGURA 2.4. Batería 18650



FIGURA 2.5. Circuito Mp4056

2.3. Componentes de software utilizados

2.3.1. Django REST framework

Django REST framework es un conjunto de bibliotecas y herramientas del lenguaje Python para el desarrollo de aplicaciones web y APIs REST. Es el framework *de facto* para el desarrollo web en Python. Entre sus características más interesantes, se encuentran[11]:

- Soporte incluído para serialización de datos.
- Código modular y reusable mediante *views* y *viewsets*, que son clases de objetos para el armado de interfaces REST sobre el modelo de datos.

- Mapeo de bases de datos relacionados a objetos mediante un *Object-relational mapping* (ORM) propio.
- Documentación automática para las API por defecto.
- Estable desde hace más de una década.
- Combina la rapidez para la codificación y flexibilidad de un lenguaje de tipado dinámico junto a un conjunto de características y estructuras para escalar el desarrollo y que siga siendo mantenible.
- Soporte para múltiples mecanismos de autenticación ya incluídos.
- Bajo uso de recursos.

En la tabla 2.3 se puede ver una comparativa con uno de los frameworks web más utilizados de la actualidad[12], Express. Se eligió Django para el desarrollo de la API de la aplicación porque se deseó trabajar con un lenguaje como Python, pero con un framework fuertemente estructurado.

TABLA 2.3. Tabla comparativa de frameworks web

Característica	Django	Express
Lenguaje	Python	Javascript
Curva de aprendizaje	Moderada	Muy baja
Arquitectura	Definida	Flexible
Enfoque	Estructurado	Libre/Flexible
Performance	Muy alta	Extremadamente alta
Baterías incluídas ¹	Sí	No

2.3.2. Angular

Angular es un framework de código abierto desarrollado originalmente por Google para la construcción de aplicaciones web del tipo SPA[13]. Utiliza el lenguaje Typescript que es un *superset* de Javascript, dotándolo de tipado estático y características no presentes en Javascript. Entre sus características más potentes se pueden destacar[14]:

- Framework completo con una estructura definida.
- Arquitectura orientada en componentes.
- *Binding* de variables bidireccional.
- Disponibilidad de muchas bibliotecas y herramientas.

Para el desarrollo de las interfaces de usuario, se planteó el uso de *Angular Material*, que es una biblioteca de componentes de interfaces disponible para Angular. Su principal ventaja es la disponibilidad de componentes estándar ya armados y diseño común[15].

Para la incorporación de mapas en la aplicación, se definió el uso de Leaflet, una biblioteca para el desarrollo de mapas en aplicaciones web. Su incorporación a Angular puede hacerse mediante el paquete *ngx-leaflet*[16] junto con el proveedor gratuito y *open source* de mapas OpenStreetMap.

2.3.3. PostgreSQL

PostgreSQL es un *relational database management system* o RDBMS de código abierto que hace uso de SQL y es famoso por ser *ACID-compliant*, su gran amplitud de tipos de datos, robustez y extensibilidad[17]. Es utilizado ampliamente por la industria del software[18] y se encuentra disponible como una opción por los mayores proveedores de *cloud* de la actualidad. Prácticamente todos los frameworks web permiten conectarse a este motor de bases de datos.

2.3.4. Heroku

Heroku es un proveedor de infraestructura y servicios en la nube y *Platform as a Service* (PaaS) focalizado en el rápido despliegue y mantenimiento de las aplicaciones web. La principal ventaja de Heroku es que apunta a abstraer al usuario de detalles técnicos y la infraestructura que da soporte a los servicios usados. Se presenta en la tabla 2.4 una comparativa frente a otros proveedores, como Amazon Web Services (AWS) y Google Cloud Platform (GCP).

TABLA 2.4. Tabla comparativa de proveedores *cloud*[19][20][21]

Característica	Heroku	AWS	GCP
Complejidad	Baja	Muy alta	Media-Alta
Costo	Medio	Bajo	Bajo-Medio
Flexibilidad	Baja-Media	Muy alta	Alta
Servicios	Poca oferta	Mucha oferta	Mucha oferta
Regiones	Limitadas	Amplias	Amplias

2.4. Herramientas y servicios de software utilizados

2.4.1. Docker

Docker es una plataforma *open source* para la contenerización de aplicaciones, con el objetivo de[22]:

- Generar versiones de sistemas idénticas entre diferentes ambientes, es decir que un sistema sea portable.
- Aislar los recursos del *host* de los sistemas que corren en él.
- Reducir los tiempos de despliegue, mantenimiento y soporte de aplicaciones.

Docker permite generar imágenes de sistemas siguiendo un conjunto de instrucciones definidas mediante un archivo *Dockerfile*, habilitando generar estas mismas imágenes en otros ambientes.

2.4.2. Visual Studio Code

Visual Studio Code es un editor de texto e IDE (Integrated development environment o Entorno de desarrollo integrado) de código abierto desarrollado originalmente por Microsoft[23]. Su flexibilidad y soporte para una cantidad considerable de extensiones para gran parte de los lenguajes y frameworks de la industria lo posicionaron en la última década como uno de los IDE más utilizados.

Permite desarrollar aplicaciones con Python, Javascript/TypeScript, incorporando incluso frameworks para el desarrollo de embedidos como es el caso de ESP-IDF[24] o PlatformIO[25]. La mayoría de proveedores de servicios en la nube han desarrollado extensiones para agilizar el despliegue de aplicaciones en sus servicios.

2.4.3. PyGPSClient

PyGPSClient es una herramienta de escritorio desarrollada en Python para el *debugging* y análisis de varios módulos GNSS de u-blox. Su principal ventaja frente a la herramienta estándar para las pruebas de estos módulos, u-center, es que está disponible para cualquier sistema que soporte Python, en comparación con la anterior que solo se encuentra para Windows[26]. En la figura 2.6 es posible observar la aplicación en ejecución, mostrando los mensajes recibidos por el módulo GPS directamente en una terminal y la ubicación geolocalizada del módulo.

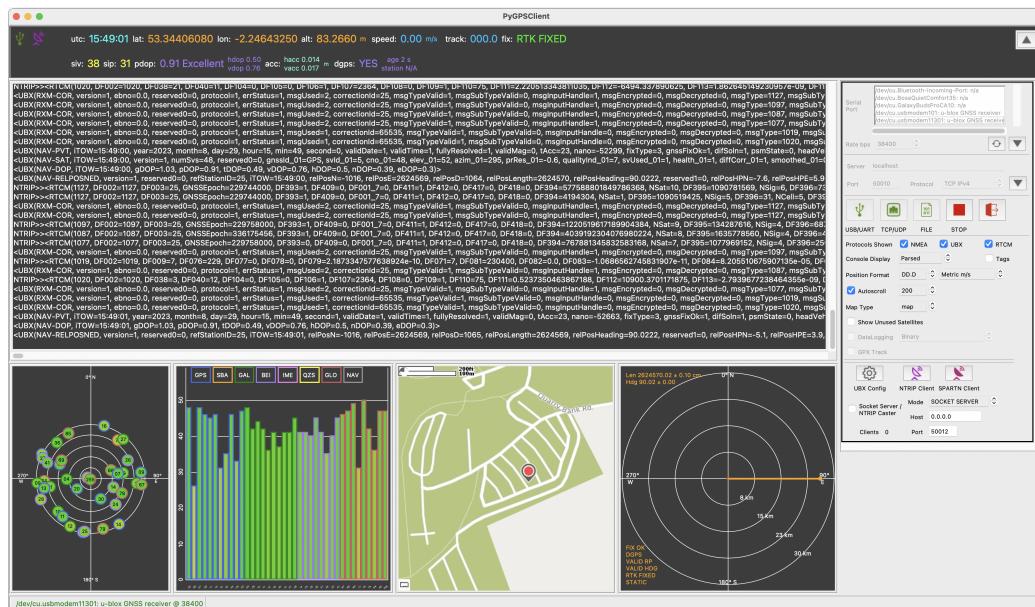


FIGURA 2.6. Aplicación PyGPSClient

2.4.4. Moserial

Moserial es una aplicación de escritorio orientada al *testing* de dispositivos embedidos, consolas y equipos electrónicos que permiten comunicación vía *Universal Asynchronous Receiver-Transmitter* o UART. Se encuentra disponible para el entorno gráfico GNOME, popular en sistemas GNU/Linux. En la figura 2.7 se muestra su uso, donde se pueden apreciar los datos recibidos vía comunicación serial. Resultó de importancia para la comunicación con el módulo de GSM, SIM800L.

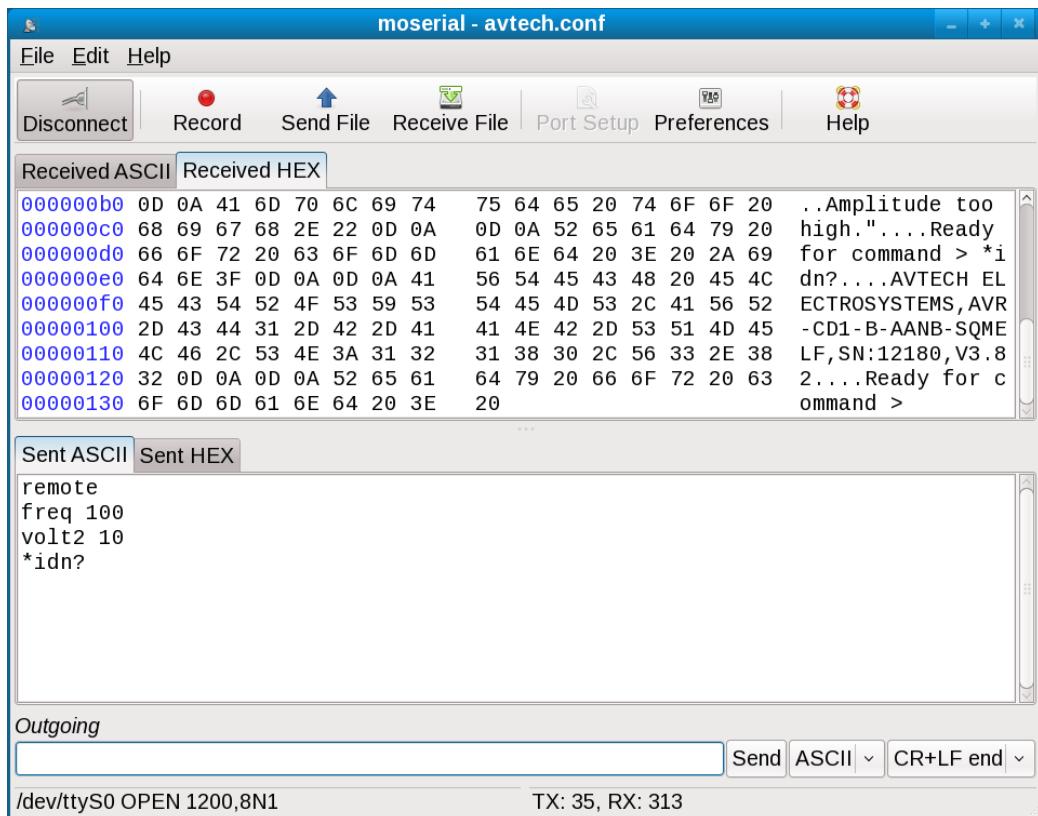


FIGURA 2.7. Aplicación Moserial

2.4.5. Twilio

Twilio es un *Communications Platform as a Service* o CPaaS que ofrece diferentes servicios para el desarrollo de sistemas que requieran funcionalidades relacionadas a telefonía IP[27]. Dentro del trabajo, su principal uso fue para la adquisición de un número de teléfono virtual para la recepción de mensajes de texto provenientes de un dispositivo embebido. Permite acceder a sus servicios mediante varios mecanismos ofrecidos por la plataforma, como APIs web.

Capítulo 3

Diseño e implementación

En este capítulo se presentan las características de diseño y desarrollo de todos los componentes que forman parte del sistema, presentados en el capítulo 2.

3.1. Arquitectura del sistema

En la figura 3.1 se presenta el diagrama general del sistema.

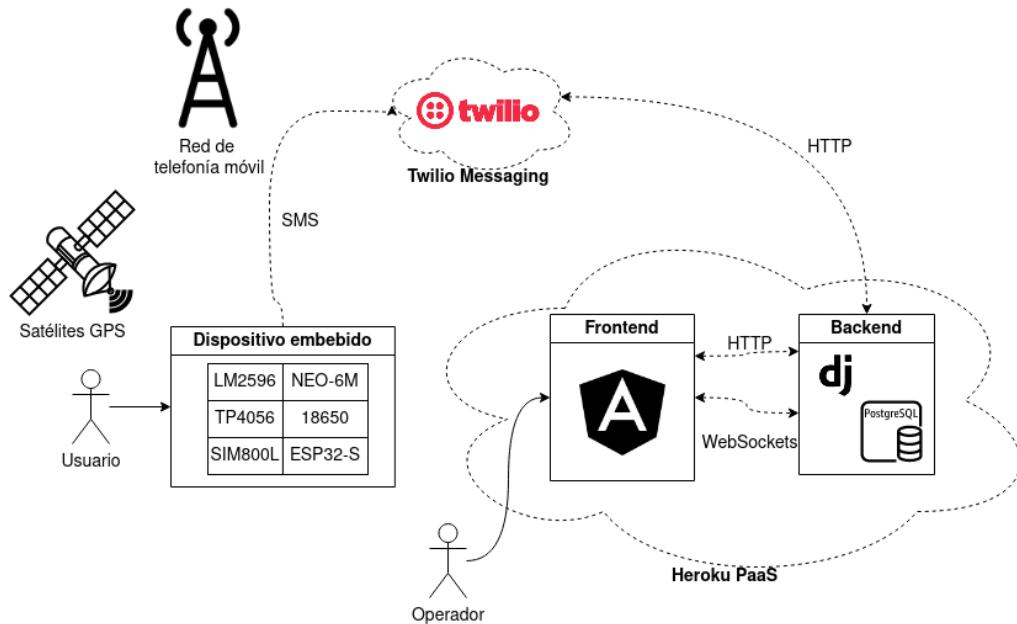


FIGURA 3.1. Arquitectura general del sistema

Se pueden identificar los siguientes componentes:

- En primer lugar, se observa a un usuario del dispositivo embebido, que es el componente que envía alertas mediante SMS hacia el servicio en la nube de Twilio.
- En segundo lugar, se encuentra el servicio de Twilio, que recepciona los mensajes recibidos y los reenvía mediante un *webhook* hacia el servicio web o *backend*. La comunicación entre el dispositivo embebido y Twilio fue posible ya que en este último se contrató un número telefónico para la recepción de los SMS.

- En tercer lugar, el *backend*, que es el componente encargado de la recepción de alertas, procesamiento y almacenamiento de datos y comunicación con el *frontend*. Fue desarrollado con el framework Django, en Python.
- A continuación, se ubica el *frontend*, que es la aplicación web que se encarga de toda la interacción del usuario operador con el sistema, incluyendo carga de datos y gestión de alertas. Consiste en una *Single Page Application* y su desarrollo fue realizado en Angular, cuyo lenguaje es TypeScript[13].
- Por último lugar, se encuentra un componente menor, pero no menos importante que es la base de datos encargada de almacenar toda la información del sistema, como usuarios, alertas, tipos de alertas, entre otros. El motor elegido fue PostgreSQL, descrito en el capítulo anterior.

Además, todos los *endpoints* requieren de autenticación y autorización por parte del operador, a excepción de algunos que deben ser públicos para ser utilizados.

3.2. Desarrollo de módulos de hardware

El firmware del dispositivo embebido fue desarrollado con el *Espressif IoT Development Framework*, abreviado como *ESP-IDF*, el framework oficial de la placa de desarrollo elegida para el proyecto, ESP32-S[24]. Su lenguaje de programación es C y provee un robusto conjunto de bibliotecas para el desarrollo sobre periféricos y funcionalidades del ESP32. Adicionalmente, ESP-IDF funciona como registro de un conjunto de componentes realizados por el fabricante oficial o la comunidad para extender funcionalidades comunes de los sistemas embebidos.

Como placa de desarrollo, el firmware se desarrolló para una placa de desarrollo ESP32-WROOM-32s del fabricante NodeMCU, un kit de desarrollo atractivo por su bajo costo, interoperabilidad, documentación disponible y bajo consumo de energía. La ventaja de usar esta placa es que, además de contar con el framework ESP-IDF y todas las características descritas en el capítulo 2, permite que el firmware desarrollado pueda ser adaptado a placas de características similares, no quedando el firmware atado solamente a una alternativa.

Para el desarrollo, se importaron varias bibliotecas usadas para el desarrollo embebido. Entre estas, resulta importante destacar dos bibliotecas importantes:

- *libnmea*: Biblioteca que facilita la lectura de los datos recibidos del módulo GPS[28]. Esto es posible ya que el NEO-6M utiliza un formato de mensajes estándar denominado NMEA[29]. La ventaja de usar *libnmea* es que convierte los datos recibidos en estructuras de C, pudiendo acceder rápidamente a valores como la latitud, longitud, etc.
- *iot-button*: Biblioteca que permite configuraciones complejas sobre botones o pulsadores[30]. La ventaja de utilizar este componente es que permite rápidamente configurar funciones de *callback* ante eventos como pulsaciones largas, sin requerir que se desarrolle lógica de control, así como también contempla inconvenientes como un *debounce* o falsas pulsaciones al momento de accionar el botón[31].

Estas dos bibliotecas fueron importadas al proyecto como *managed components*, propios del registro de componentes externos de ESP-IDF[24]. En la tabla 3.1 se encuentra detallado el uso que tuvo cada biblioteca importadas para el firmware.

TABLA 3.1. Bibliotecas más relevantes utilizadas

Biblioteca	Uso
<i>iot_button.h</i>	Implementación del botón pulsador y acciones a tomar al momento de ser usado
<i>nmea.h</i>	Lectura de mensajes NMEA recibidos por el módulo GPS
<i>string.c</i>	Manipulación y transformación de cadenas de texto al momento de leer mensajes recibidos de los módulos
<i>ctype.c</i>	Uso de funciones comunes de C para trabajar sobre caracteres de texto
<i>uart.h</i>	Comunicación con los módulos GSM y GPS mediante puerto serie
<i>esp_system.h</i>	Funciones básicas del ESP-IDF
<i>esp_log.h</i>	Visualización de errores y mensajes de <i>debug</i> del firmware
<i>esp_pm.h</i>	Configuraciones de ahorro de energía
<i>esp_sleep.h</i>	
<i>adc_oneshot.h</i>	
<i>adcadc_cali.h</i>	Conversión del módulo analógico-digital y lectura de valores para calcular la carga de la batería
<i>adc_cali_scheme.h</i>	
<i>nvs.h</i>	Almacenamiento <i>flash</i> para guardar datos en la memoria no volátil, como la última posición válida
<i>nvs_flash.h</i>	
<i>freertos/FreeRTOS.h</i>	Configuración del sistema operativo en tiempo real
<i>freertos/task.h</i>	Implementación de procesos/tareas

Para el desarrollo, compilación, pruebas y monitoreo, se trabajó con el entorno de desarrollo integrado Visual Studio Code, en conjunto con la extensión oficial de ESP-IDF, el cual facilita el trabajo con el kit de desarrollo y la incorporación y uso de módulos propios del ecosistema del ESP32. En la figura 3.2 se pueden observar los componentes físicos del dispositivo embebido:

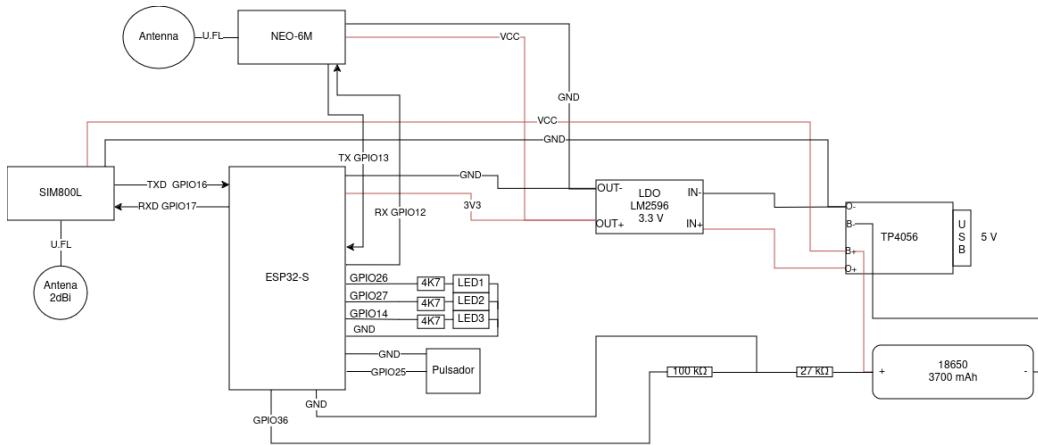


FIGURA 3.2. Esquema de componentes de hardware del ESP32

Adicionalmente a los dispositivos originalmente elegidos, se incorporó un regulador de tensión LM2596 en la entrada de energía de la batería no contemplado originalmente, que permite adecuar el voltaje de entrada de 4.2 V a 3.3 V[32], requeridos por el ESP32 y el NEO-6M. Además, se incorporó una antena activa

para el módulo GPS ya con una antena pasiva la precisión de la localización obtenida era baja o tomaba más tiempo del esperado. Por otra parte, se reemplazó la antena helicoidal del módulo GSM por una con formato PCB y 3dBi de ganancia.

3.2.1. Tareas del dispositivo embebido

Para implementar las funciones del dispositivo embebido, se dividió el desarrollo del firmware en 7 bloques o funciones de código principales. Estas se pueden describir de la siguiente forma:

- Inicio: consiste en la función de arranque del firmware, en donde se realizan acciones como inicialización de periféricos, revisión de errores de arranque, reducción de la velocidad de reloj del procesador al mínimo requerido, asignación inicial de variables, y por último, la configuración de las tareas que corren en bucle. Posterior al inicio, esta función finaliza su ejecución. En la figura 3.3 se pueden observar todos los pasos contemplados. No es una tarea en el sentido estricto de *FreeRTOS*.
- Manejador del módulo GSM: consiste en una función con un bucle infinito que se encarga de cada pocos segundos recibir comandos/respuestas del módulo SMS leyendo el puerto serie y tomar una acción ante cada mensaje si corresponde. En la figura 3.3 se puede visualizar el bucle con las acciones que realiza. Los mensajes manejados son:
 - AT: mensaje que representa un OK.
 - CMT: mensaje que significa que se recibió un SMS; esto se utiliza para configurar el número de teléfono al cual se le deben enviar las alertas por SMS.
 - CMGS: mensaje que significa se envió correctamente un SMS; se utiliza para confirmar el envío correcto de una alerta.
 - CSQ: mensaje que incluye el nivel de calidad de la señal de telefonía móvil.
 - GSN: mensaje que incluye el IMEI del dispositivo; se utiliza como mecanismo de seguridad ya que el SMS de configuración de teléfono debe incorporar los primeros dígitos del IMEI como validación.
- Manejador del módulo GPS: consiste en una función con un bucle que se encarga de recibir comandos/respuestas del módulo GSM leyendo el puerto serie y guardar la información recibida. Existen diferentes tipos de mensajes, pero para los fines del registro de la localización, solamente se está guardado la información del tipo de mensaje GPGGA, que informa la posición actual y datos de los satélites[33]. En la figura 3.3 se puede visualizar el bucle con las operaciones.

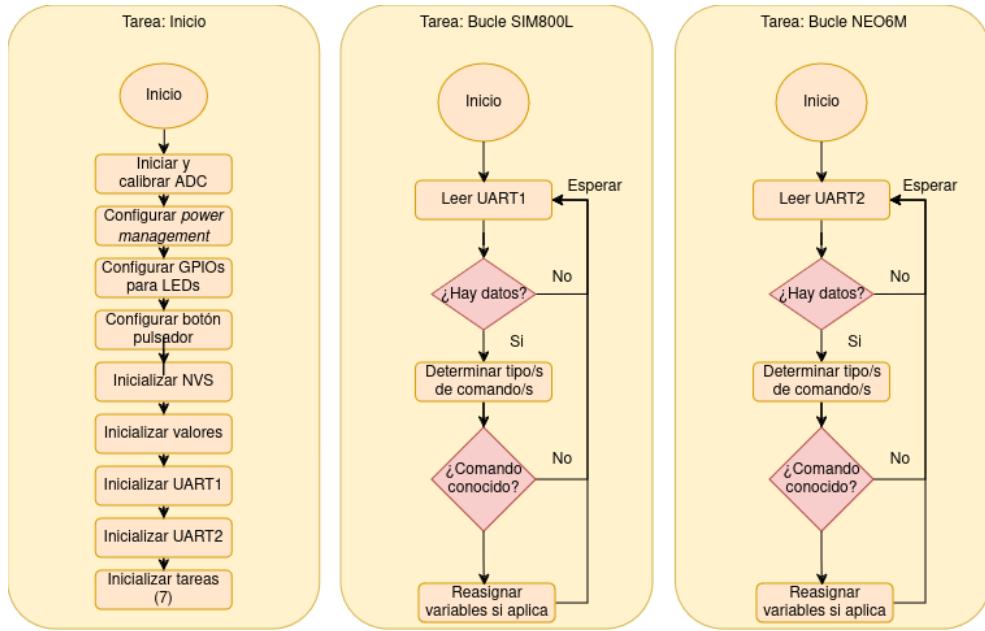


FIGURA 3.3. Diagramas de flujo de tarea inicial, y bucles del módulo GSM y GPS

- *Setup* de energía del módulo GSM: tarea que espera por la correcta inicialización del módulo GSM y posteriormente, aplica algunas optimizaciones de energía. En la figura 3.4 se listan los pasos ejecutados. Luego, la tarea termina su ejecución.
- *Setup* de energía del módulo GPS: tarea que espera por la inicialización del módulo GPS y comunicación con al menos 5 satélites para enviarle algunos comandos de optimización. Luego de esto, la tarea finaliza. En la figura 3.4 se pueden visualizar el flujo de esta.

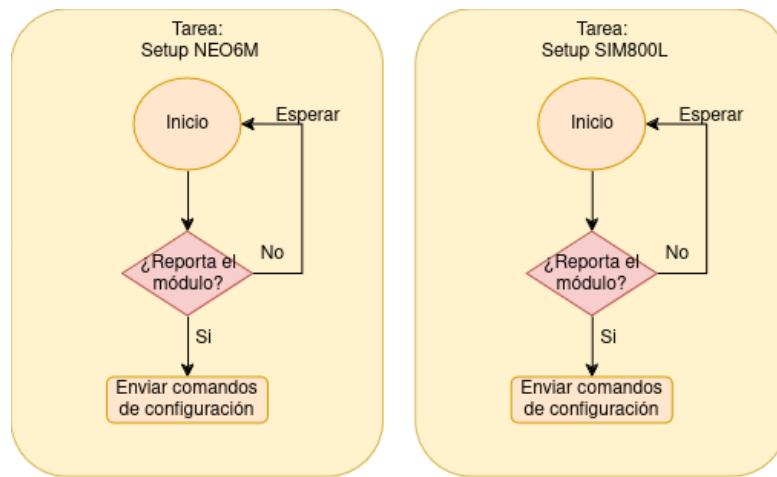


FIGURA 3.4. Diagramas de flujo de las tareas de *setup* de energía

- Control de estado de GPS, GSM y batería: esta tarea tiene como objetivo, cada 30 segundos, leer el estado de las variables de calidad de la señal GSM, cantidad de satélites conectados y nivel de batería, y parpadear 3 luces dependiendo del estado de estas variables. En el desarrollo, se repartió en tres

funciones pequeñas, pero a fines explicativos se presenta como una única tarea. En la figura 3.5 se puede observar el flujo de ejecución de la tarea.

- Almacenamiento periódico de datos: cada 1 minuto esta tarea se encarga de leer las variables de latitud, longitud, IMEI y número de teléfono asignado y almacenarlas en el almacenamiento no volátil. en el caso de que los valores fueran válidos. En la figura 3.5 se puede observar el bucle ejecutado por la tarea.
- Control periódico del módulo GSM: esta tarea se encarga de solicitarle información al módulo GSM información sobre el estado de la red cada 1 minuto. Esto es debido a que el módulo GSM no reporta automáticamente su estado, a diferencia del módulo GPS que si lo hace. En la figura 3.5 se puede apreciar el control que realiza la función.

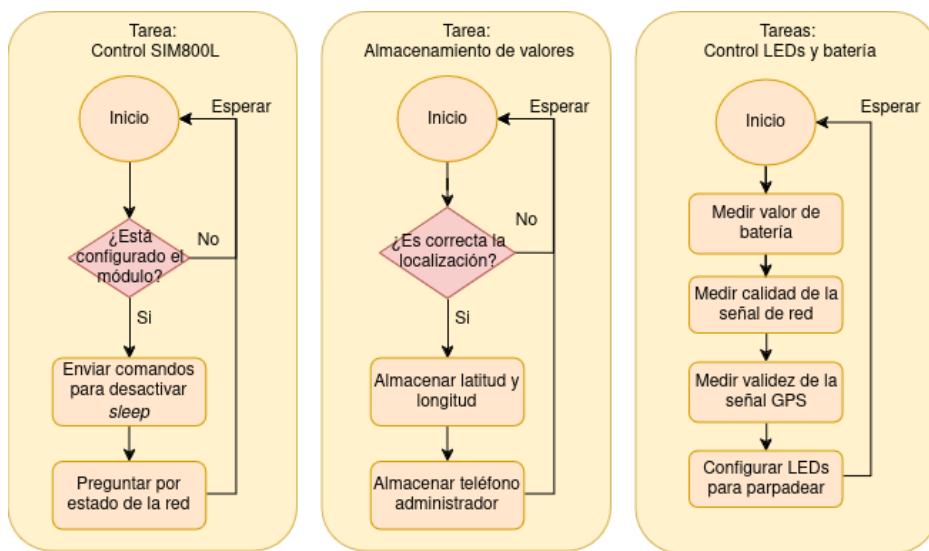


FIGURA 3.5. Diagramas de flujo de las tareas de control y almacenamiento

Por último, se puede considerar una última función más dentro del firmware que es el interruptor ante la pulsación del botón activador. En caso que se supere un umbral de 3 segundos del pulsador presionado, se activará *callback* definido para ese evento. Este evento se encarga de levantar los valores actuales de la última localización válida almacenada, determinar cual es el número de teléfono al cual se debe enviar el mensaje de alerta y enviar los comandos hacia el módulo GSM para enviar el SMS.

En la figura 3.1 se muestra un diagrama de flujo resumido de la operatoria de la interrupción. Para la asociación del *callback* con el evento, se puede ver el código definido en la función inicial del firmware en la figura 3.6, donde `sos_button_long_press_cb` es una función definida y `cfg` es la definición del evento.

```

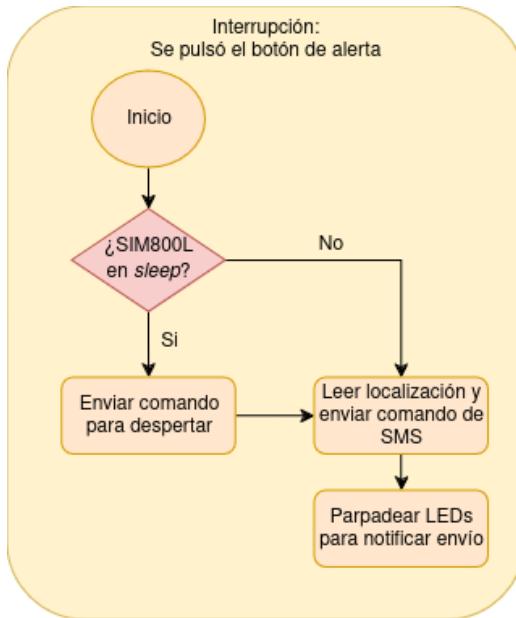
1 button_config_t gpio_btn_cfg = {
2     .type = BUTTON_TYPE_GPIO,
3     .long_press_time = SOS_BUTTON_LONG_PRESS_TIME_MS,
4     .gpio_button_config = {
5         .gpio_num = SOS_BUTTON,
6         .active_level = 1,
7         .enable_power_save = true },

```

```

8  };
9
10 button_handle_t gpio_btn = iot_button_create(&gpio_btn_cfg);
11 if (NULL == gpio_btn)
12 {
13     ESP_LOGE(TAG, "Button create failed");
14 }
15
16 button_event_config_t cfg = {
17     .event = BUTTON_LONG_PRESS_START,
18     .event_data.long_press.press_time = SOS_BUTTON_LONG_PRESS_TIME_MS,
19 };
20
21 ESP_ERROR_CHECK(iot_button_register_event_cb(gpio_btn, cfg,
22                 sos_button_long_press_cb, NULL));

```

CÓDIGO 3.1. Definición del evento y asociación del *callback*FIGURA 3.6. Diagrama de flujo del *callback* por pulsación

3.2.2. Optimizaciones de energía aplicadas

Se aplicaron diferentes técnicas de optimización sobre el ESP32-S y los módulos GSM y GPS con el objetivo de optimizar y disminuir el consumo de energía:

- Se configuró *FreeRTOS* para solamente utilizar un núcleo del ESP32-S.
- Se limitó la frecuencia del procesador para solamente utilizar como máximo 80 MHz.
- Se desactivaron todos los periféricos no usados, como WiFi, Bluetooth, etc.
- Se configuró el *Dynamic frequency scaling* o DFS del ESP32, que permitiría que se reduzca la velocidad del reloj del procesador[34].
- Se configuró el modo *sleep* del módulo SIM800L, desactivandolo solo cuando se le debe enviar comandos.

- Se desactivó el LED de notificación de calidad de señal del módulo SIM800L, ya que la comprobación se realiza de forma programática.
- Se redujo la frecuencia de actualización de mensajes del módulo NEO-6M hacia el ESP32.
- Se aplicó sobre NEO-6M una configuración de energía denominada *Power Save Mode* para reducir la frecuencia de actualización hacia los satélites GPS[35].
- Se configuró el modo *sleep* del módulo SIM800L, desactivandolo solamente cada ciertos intervalos de tiempo.
- Se utilizaron tiempos muertos de espera o *standby* del orden de varios segundos en el ESP32-S, resultando en que gran parte del tiempo, la placa se encuentre en espera.
- Se redujo el consumo de energía de los 3 LEDs indicadores de GSM, GPS y batería mediante la incorporación de resistencias, requiriendo aproximadamente 1 mA para todos los LEDs.

Además, se intentó aplicar un manejo tanto manual como automático (configurable mediante el uso de DFS) del modo *light sleep* pero se encontró que, después de varios intentos y pruebas, el consumo de energía no disminuyó. Se interpretó, en base a las pruebas durante el desarrollo, que es debido a características de fabricación del modelo ESP32-WROOM-32s de NodeMCU. Se midió un consumo en *idle* de aproximadamente 15 mA, aunque esté en *deep sleep* o *light sleep*.

3.3. Desarrollo de módulo de backend

Para el desarrollo de la aplicación *backend* y sus funcionalidades, se estructuró el proyecto siguiendo los lineamientos de Django, que consiste en principalmente dividir la aplicación en módulos web por temática o características. Para el caso de esta aplicación, se identificaron dos módulos:

- *Users*: para englobar toda la funcionalidad referida a usuarios.
- *Alerts*: para englobar toda la funcionalidad referida a beneficiarios y alertas.

Esto no significa que no puedan o no deban referenciarse los módulos entre sí, sino que es una forma lógica de estructurar la información perteneciente a la misma lógica de negocio. En la figura 3.7 y 3.8 se pueden visualizar las carpetas de cada módulo con sus respectivos componentes.



FIGURA 3.7. Estructura de archivos de módulo de *Users*



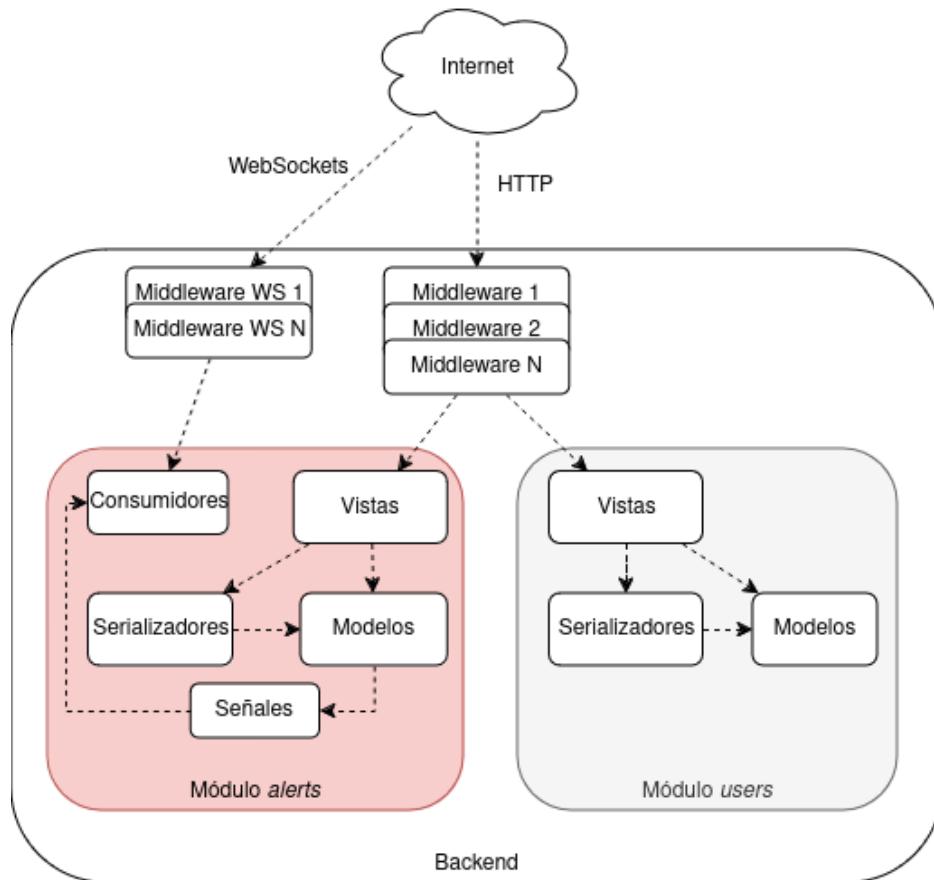
FIGURA 3.8. Estructura de archivos de módulo de *Alerts*

Para la implementación, se incorporaron varias bibliotecas externas, cuyo aporte se muestra en la tabla 3.2:

TABLA 3.2. Bibliotecas externas más relevantes

Biblioteca	Uso
<i>djangorestframework</i>	Biblioteca de Django para el desarrollo de APIs REST
<i>psycopg</i>	Biblioteca para conexión a PostgreSQL[36]
<i>django-twilio</i>	Biblioteca oficial de Twilio[37] para comunicación con <i>Twilio Messaging Services</i>
<i>channels</i>	Biblioteca para la implementación de WebSockets en Django[38]
<i>djangorestframework-simplejwt</i>	Biblioteca para el <i>middleware</i> de autenticación de usuarios
<i>drf_yasg</i>	Biblioteca para generar documentación de API en Swagger
<i>uvicorn</i>	Servidor web ASGI para Python[39]
<i>django-cors-headers</i>	<i>Middleware</i> para incorporar encabezados en las peticiones de la API
<i>dj-database-url</i>	Biblioteca para conectarse a base de datos PostgreSQL de Heroku
<i>django-channels-jwt</i>	<i>Middleware</i> para autenticación de WebSockets

En la figura 3.9 se presenta en forma de diagrama de bloques los principales componentes de la API:

FIGURA 3.9. Diagrama en bloques del *backend*

Se describen a continuación los componentes de mayor relevancia de los módulos:

- **Modelos:** clases que representan los modelos de datos, como pueden ser las entidades de una alerta y un beneficiario.
- **Serializadores:** clases que agrupan la lectura, escritura, validación y salida de modelos de datos. Los serializadores permiten encapsular lógica de validación para diferentes operaciones o adaptar la representación de los datos según la necesidad[40]. En la figura 3.10 se muestra el serializador de la entidad *Beneficiary* junto con validaciones y acciones que deben realizarse durante la creación y actualización, así como también definir restricciones al momento de realizar la carga de datos.

```

class BeneficiarySerializer(serializers.ModelSerializer):
    id = serializers.IntegerField(read_only=True)
    name = serializers.CharField(max_length=64, required=True)
    surname = serializers.CharField(max_length=64, required=True)
    telephone = serializers.CharField(
        max_length=32, required=True, validators=[only_int]
    )
    company = serializers.ChoiceField(
        choices=Beneficiary.COMPANY_CHOICES, default="OTH", required=False
    )
    enabled = serializers.BooleanField(default=True)
    type_id = serializers.IntegerField(allow_null=True, required=False)

    class Meta: ...

    def validate(self, data): ...

    def create(self, validated_data): ...

    def update(self, instance, validated_data): ...

```

FIGURA 3.10. Serializador de la entidad *Beneficiary*

- **Vistas:** clases que engloban peticiones HTTP sobre entidades o clases para atender un único tipo de petición, dependiendo del requerimiento[41]. En la figura 3.11 se puede apreciar la definición del *ViewSet* de la clase *Beneficiary*, pudiendo observarse que se asignaron de forma declarativa los permisos requeridos para realizar las peticiones, y los filtros habilitados, sin requerir código adicional.

```

class BeneficiaryViewSet(EnablePartialUpdateMixin, viewsets.ModelViewSet):
    """
    A viewset that provides the standard actions for beneficiaries
    """

    permission_classes = [
        IsSameOrganization,
    ]
    serializer_class = BeneficiarySerializer
    filter_backends = [DjangoFilterBackend]
    filterset_fields = ["telephone", "name", "surname", "enabled"]

    def get_queryset(self):
        user = self.request.user
        queryset = self.filter_queryset(
            Beneficiary.objects.filter(organization=user.organization).order_by("id")
        )
        return queryset

    def destroy(self, request, *args, **kwargs): ...
    def update(self, request, *args, **kwargs): ...

```

FIGURA 3.11. Serializador de la entidad *Beneficiary*

- **Consumidores:** representa un servicio de WebSockets que permite obtener una alerta nueva en tiempo real.
- **Middlewares:** clases que permiten interceptar peticiones hacia la API[42] y realizar comprobaciones como ver el rol del usuario, o que este tenga la correcta autorización en las peticiones. No se desarrolló ningún *middleware* sino que se utilizaron bibliotecas externas ya provistas.
- **Señales:** hace referencia a una notificación que se genera dentro de la aplicación ante un evento[43]. En el marco de esta aplicación, cuando se almacena una nueva alerta, se genera una señal que notifica en tiempo real mediante WebSockets el dato creado.

Respecto a las características, se diseñó el *backend* con la idea de un producto *multitenant*, es decir que permita que a futuro existan diferentes organizaciones cliente dentro de este[44]. Debido a esta característica, todos los datos están contenidos dentro de una misma organización, compartida por los usuarios, beneficiarios, entre otros datos cargados. En la figura 3.12 se puede visualizar el modelo de datos relacional con las entidades diseñadas.

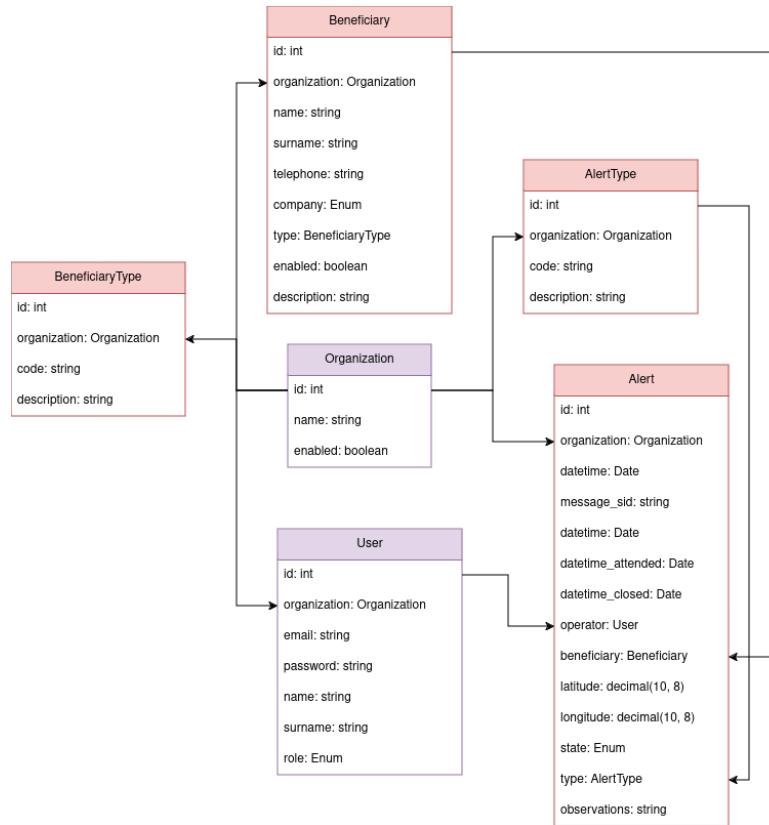


FIGURA 3.12. Modelo de datos

De todas maneras, para esta versión inicial, solo se contempla un usuario por organización en el *frontend*, aunque a futuro el diseño y la lógica de negocio en la API se encuentra desarrollada para contemplar una organización con un usuario raíz o padre y un conjunto de usuarios operadores, de menor privilegio.

3.3.1. Servicios desarrollados

A continuación, se muestran en la tabla 3.3 los servicios implementados:

TABLA 3.3. Endpoints implementados

Endpoint	Descripción
GET /alert-types/	Obtener tipos de alerta
POST /alert-types/	Crear un tipo de alerta
GET /alert-types/id/	Obtener un tipo de alerta
PUT /alert-types/id/	Editar un tipo de alerta
DELETE /alert-types/id/	Eliminar un tipo de alerta
GET /alerts/	Obtener alertas
GET /alerts-summary/	Obtener alertas de las últimas 24 horas
GET /alerts/id/	Obtener una alerta
PUT /alerts/id/	Editar una alerta
PATCH /alerts/id/	Editar parcialmente una alerta
GET /beneficiaries/	Obtener beneficiarios
GET /beneficiaries/id/	Obtener un beneficiario
PUT /beneficiaries/id/	Editar un beneficiario
PATCH /beneficiaries/id/	Editar parcialmente un beneficiario
DELETE /beneficiaries/id/	Desactivar un beneficiario
GET /beneficiary-types/	Obtener tipos de beneficiarios
GET /beneficiary-types/id/	Obtener un tipo de beneficiario
PUT /beneficiary-types/id/	Editar un tipo de beneficiario
DELETE /beneficiary-types/id/	Desactivar un beneficiario
POST /users/admin/	Crear un usuario administrador con una organización
GET /user-details/	Obtener datos del usuario
PATCH /user/id/	Editar parcialmente un usuario
POST /login/	Realizar login programático
GET /ws-auth/	Obtener token efímero para conexión a WebSockets
POST /twilio-webhook/	Webhook para generar una alerta desde Twilio
WS /ws/alerts/organization-id/	Conectarse a canal de WebSockets

Todos estos servicios fueron documentados mediante Swagger, una herramienta para documentación de APIs[45]. Todas las vistas, a excepción del registro de un nuevo usuario administrador, requieren de autenticación mediante un token JWT. Esto es autogestionado mediante la incorporación de un *middleware* de Django, sin requerir código adicional más allá de declarar en la configuración de Django de la aplicación, que *middlewares* utilizar para la autenticación, como se observa en la figura 3.13:

```
REST_FRAMEWORK = {
    "DEFAULT_SCHEMA_CLASS": "rest_framework.schemas.coreapi.AutoSchema",
    "DEFAULT_AUTHENTICATION_CLASSES": [
        "rest_framework_simplejwt.authentication.JWTAuthentication",
    ],
    "DEFAULT_PERMISSION_CLASSES": [
        "rest_framework.permissions.IsAuthenticated",
    ],
}
```

FIGURA 3.13. Configuración de *middlewares* para la autenticación de la API

3.4. Desarrollo de módulo de frontend

El desarrollo de la aplicación web en *Angular* se estructuró siguiendo la estructura estándar del framework[46], la cual se puede apreciar en la figura 3.14:

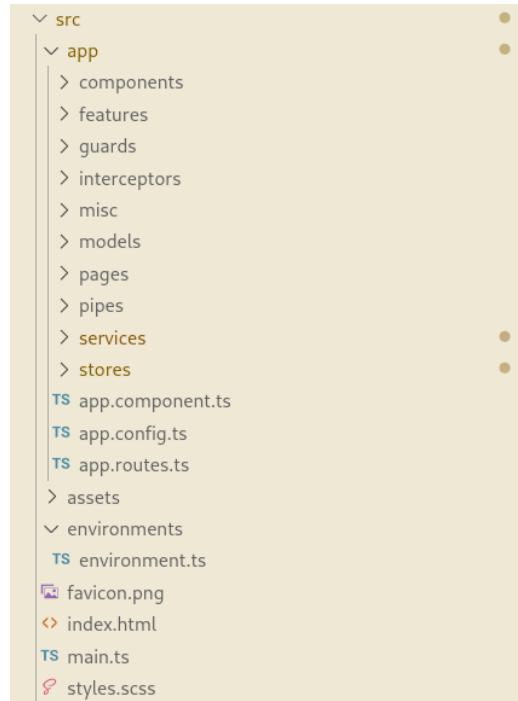


FIGURA 3.14. Estructura del *frontend*

Respecto al uso de cada carpeta, se pueden describir los más relevantes de la aplicación:

- *pages*: corresponde a componentes que representan vistas enteras o secciones de la aplicación web.
- *components*: corresponde con componentes invocados dentro de las páginas, como son los *modals* o ventanas de carga de datos, el *layout* o estructura base de la interfaz, y otros componentes.
- *stores*: clases que encapsulan datos y funcionan como una fuente de datos única para toda la aplicación[47].
- *services*: clases que encapsulan el acceso a diferentes métodos del *backend* para cada una de las entidades.
- *models*: definición de interfaces para representar los modelos de datos.
- *guards*: corresponde con clases que permiten limitar o no el acceso a una vista.
- *interceptors*: representa clases que interceptan llamadas HTTP y la comunicación de datos con el *backend* para aplicar transformaciones de datos.
- *pipes*: los *pipes* son clases que transforman la representación de objetos o datos hacia el usuario, como por ejemplo cambiar el formato de fecha y hora[48].

La implementación de la estructura, diseño de componentes, visualización de formularios, entre otros componentes gráficos, contó con el uso de la biblioteca de estilos *Angular Material* y con el uso de uno de los temas por defecto de la biblioteca. Toda la estructura se organizó solamente utilizando *flexbox*, un método de diseño de páginas propio de *CSS*, el lenguaje de estilos de los documentos web[49]. No se contempló la necesidad de incorporar alguna herramienta o biblioteca para estructuras complejas, como por ejemplo *Bootstrap* que contiene un sistema de grilla[50], debido a que las páginas y secciones de la aplicación web son simples.

En relación a las páginas implementadas, se desarrollaron las rutas descritas en la tabla 3.4:

TABLA 3.4. *Endpoints* implementados

<i>Endpoint</i>	Descripción
/login	Sección de login
/register	Sección de registro de un nuevo usuario
/map	Vista de mapa con las últimas alertas recibidas
/beneficiaries	Listado de beneficiarios
/beneficiary-types	Listado de tipos de beneficiarios
/alerts	Listado de alertas
/alert-types	Listado de tipos de alertas

3.4.1. Particularidades del desarrollo

En relación al manejo de datos desde la aplicación, se implementó en el *frontend* un patrón de manejo de estado para las entidades del sistema denominado *Store*, inspirado en patrones más complejos como *Redux*, que surgieron para simplificar el manejo de estados en aplicaciones de usuario cada vez más complejas. En el marco de esta aplicación, se utiliza el *Store* para almacenar y encapsular el acceso a las entidades, desde una única fuente de datos, compartida por todos los componentes del sistema, secciones, *popups*, etc.[51]. Para esto, se generó un *store* por cada modelo deseado y toda interacción de los componentes con datos, y comunicación de componentes que implican datos, fue eliminada, ya que se realiza mediante el *store* de cada entidad.

Debido a que todas las peticiones hacia el *backend*, a excepción del registro y login, requieren incluir un encabezado de autorización en la petición junto con el token *JWT*, se implementó un interceptor llamado *authGuard* que toma cualquier petición, y le incorpora el token de autorización almacenado en el almacenamiento local del navegador cliente. En la figura 3.15 se visualiza la implementación del interceptor:

```

export const authInterceptor: HttpInterceptorFn = (request, next) => {
  const store = inject(AuthStore);
  const token = store.getAccessToken();

  return next(
    request.clone({
      body: request.body ? snakecaseKeys(request.body as {}) : request.body,
      setHeaders: {
        Authorization: `Bearer ${token}`,
      },
    }),
  ).pipe(
    map((response) => {
      if (response instanceof HttpResponse && response.body) {
        return response.clone({ body: camelcaseKeys(response.body as {}) });
      } else {
        return response;
      }
    }),
  );
};

```

FIGURA 3.15. Estructura del *frontend*

Adicionalmente, este *interceptor* se utilizó para resolver una problemática que existe entre aplicaciones que están desarrolladas con diferentes convenciones de código. En este caso, el *backend* fue desarrollado en Python, que sigue la convención *snake case* para nombrar las variables, y el *frontend* fue desarrollado en TypeScript, que sigue la convención de *camel case*^[52]. En la figura 3.16 se observa mejor la diferencia que hay, lo que impide asociar de manera automática las propiedades de los datos enviados entre *backend* y *frontend*. Para esto, se implementaron 2 bibliotecas que convierten automáticamente los datos salientes y entrantes a la convención deseada.

Camel case	Snake case
<pre>{ "firstName": "John", "lastName": "Smith", "email": "john.smith@example.com", "createdAt": "2021-02-20T07:20:01", "updatedAt": "2021-02-20T07:20:01", "deletedAt": null }</pre>	<pre>{ "first_name": "John", "last_name": "Smith", "email": "john.smith@example.com", "created_at": "2021-02-20T07:20:01", "updated_at": "2021-02-20T07:20:01", "deleted_at": null }</pre>

FIGURA 3.16. Comparativa entre *snake case* y *camel case*

Por último, otra característica particular del desarrollo fue la incorporación de un mecanismo de *exponential backoff* para el establecimiento de la conexión mediante WebSockets ya que se encontraron algunos problemas durante la conexión segura por WS con el token efímero utilizado. En ciertas ocasiones, el token era marcado como inválido por el *backend*, lo que impedía establecer la conexión, requiriendo refrescar la aplicación y volver a probar establecer la conexión. Para esto, se incorporó una biblioteca para los reintentos de conexión separados por un tiempo determinado por este algoritmo, que aumenta el tiempo entre peticiones entre cada falla^[53].

Este problema ocurre con el *middleware* utilizado en el *backend* ya que es una biblioteca de terceros y aparenta tener errores espontáneos al momento de determinar si el token es válido o no.

En la tabla 3.5 se listan las principales bibliotecas importadas para el desarrollo del *frontend*:

TABLA 3.5. Bibliotecas de terceros utilizadas

Biblioteca	Descripción
<i>leaflet</i> @asymmetrik/ngx-leaflet	Implementación del mapa de alertas y funcionalidad asociada a este
<i>rxjs</i>	Implementación de métodos asíncronos para gestión de eventos
@ngrx	Implementación de manejo de estado y datos
@ngrx/operators @ngrx/signals	Importación de operadores de NgRx para el manejo de estado
<i>backoff-rxjs</i>	Implementación de reconexión en WebSockets ante errores
<i>camelcase-keys</i> <i>snakecase-keys</i>	Conversión de formato de datos enviados entre <i>backend</i> y <i>frontend</i>
<i>express</i>	Implementación de servidor para servir el <i>frontend</i>
<i>mat-table-exporter</i>	Funciones de exportación de tablas

3.5. Integración

El esquema de integración del sistema puede dividirse en tres etapas:

- Integración entre sistema embebido y Twilio.
- Integración entre Twilio y *backend*.
- Integración entre *backend* y *frontend*.

Respecto a la integración entre el sistema embebido y Twilio, se realiza únicamente mediante mensajes de texto. Se puede observar esta interacción que hay en la figura 3.17. El dispositivo embebido sabe cuál es el número de teléfono configurado para la recepción de alertas ya que es configurable mediante un SMS, cuyo contenido debe incluir como clave la parte de un número de identificación único por cada módulo GSM, denominado IMEI[54].

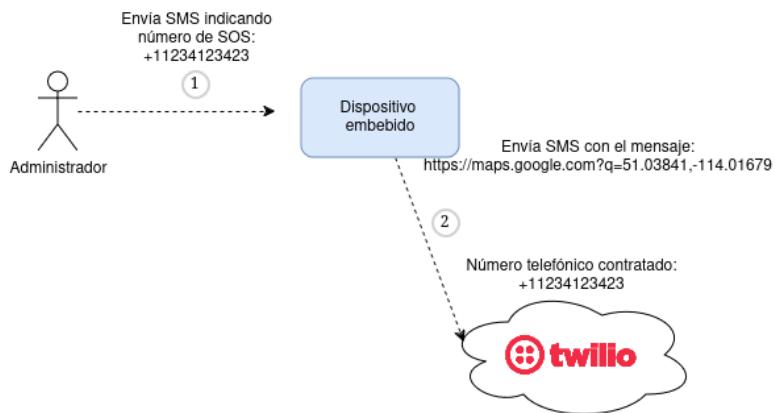


FIGURA 3.17. Esquema de integración entre botón antipánico y Twilio

Al momento de integrar el *backend*, *frontend*, bases de datos y servicios externos, se disponibilizó el sistema en un entorno *cloud*. Los componentes web se desplegaron en la nube de Heroku mediante el uso de *dynos* básicos, que son contenedores de aplicaciones web[55], y para la base de datos, se utilizó un *addon* de PostgreSQL, que consiste en un complemento que permite incorporar una base de datos como SaaS u otros componentes requeridos por los sistemas, como pueden ser bases de datos en memoria o brókers[56]. Al momento de desplegar, fueron asignadas dos URLs, una para el frontend y otra para la API, permitiendo que el *frontend* y Twilio apunten a la API correctamente.

Para la integración entre la API y Twilio, se implementó un único servicio del tipo *webhook* para que Twilio notifique en tiempo real la llegada de una nueva alerta. Para poder securizar las ejecuciones del *webhook* y asegurarse de que sea ejecutado solamente por los sistemas de Twilio, se utilizó el esquema de seguridad provisto por Twilio. Cuando Twilio realiza una petición hacia una API, incluye dos parámetros en el cuerpo de la petición?:

- ACCOUNT_SID: identificador de la cuenta de Twilio.
- AUTH_TOKEN: credencial para autenticación, única por cada cuenta.

Por este motivo, toda interacción debe realizarse bajo HTTPS o se corre el riesgo de que puedan filtrarse las credenciales enviadas como parámetros de la petición. El backend utiliza estas credenciales y las compara con las credenciales configuradas como variables y determina si quién está realizando la llamada al *webhook* es legítimo y procesa la petición. En la figura 3.18 se puede ver la interacción entre Twilio y el *backend*.

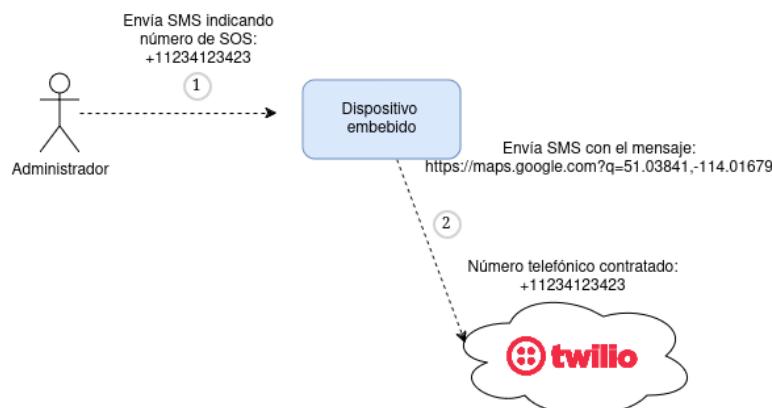


FIGURA 3.18. Esquema de integración entre Twilio y la API

Por último, en relación a la comunicación entre el *frontend* del usuario y la API, toda la interacción entre el *backend* y *frontend* es realizada mediante *endpoints* HTTP y *WebSockets* para comunicación en tiempo real, utilizando TLS/SSL para secu-
rizar las conexiones. Además, para poder autorizar cada petición HTTP, el usuario debe incluir un *JSON Web Token* o *JWT* en el encabezado de esta, que es un mensaje con formato estandarizado utilizado principalmente para autorización[57]. Este token es generado por el usuario al momento de loguearse en el sistema web.

Para el caso de la conexión mediante *WebSockets* o WS, usada para recibir nuevas alertas en tiempo real, se implementó otro mecanismo de autorización debido a la

imposibilidad de incluir encabezados en las peticiones o conexiones. Se incorporó un *endpoint* que permite que el usuario genere un token efímero de uso único para cada conexión de WS. Este token se envía como parámetro al momento de iniciar la conexión y tiene validez mientras se mantenga la conexión establecida. En la figura 3.19 se observan los mecanismos de comunicación entre el *frontend* y backend.

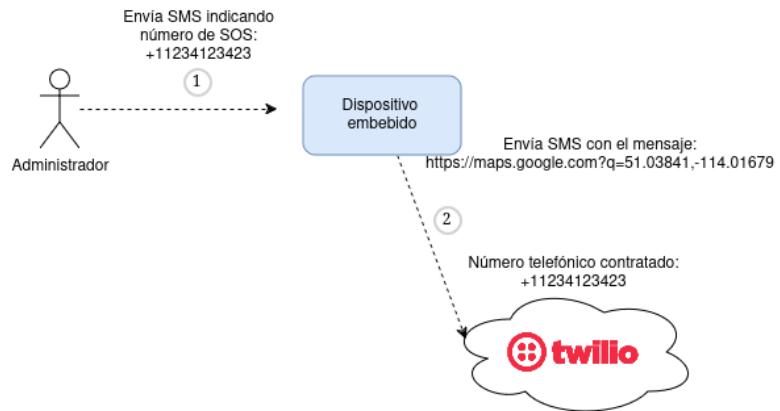


FIGURA 3.19. Esquema de integración entre *frontend* y la API

Capítulo 4

Ensayos y resultados

4.1. Pruebas funcionales del hardware

La idea de esta sección es explicar cómo se hicieron los ensayos, qué resultados se obtuvieron y analizarlos.

Capítulo 5

Conclusiones

5.1. Conclusiones generales

La idea de esta sección es resaltar cuáles son los principales aportes del trabajo realizado y cómo se podría continuar. Debe ser especialmente breve y concisa. Es buena idea usar un listado para enumerar los logros obtenidos.

Algunas preguntas que pueden servir para completar este capítulo:

- ¿Cuál es el grado de cumplimiento de los requerimientos?
- ¿Cuán fielmente se puedo seguir la planificación original (cronograma incluido)?
- ¿Se manifestó algunos de los riesgos identificados en la planificación? ¿Fue efectivo el plan de mitigación? ¿Se debió aplicar alguna otra acción no contemplada previamente?
- Si se debieron hacer modificaciones a lo planificado ¿Cuáles fueron las causas y los efectos?
- ¿Qué técnicas resultaron útiles para el desarrollo del proyecto y cuáles no tanto?

5.2. Próximos pasos

Acá se indica cómo se podría continuar el trabajo más adelante.

Bibliografía

- [1] Programa de las Naciones Unidas para el Desarrollo. *Sinopsis: Seguridad Ciudadana*.
<https://www.undp.org/es/publications/sinopsis-seguridad-ciudadana>. Abr. de 2014.
- [2] Traccar. *Chinese Clones*. <https://www.traccar.org/clones/>.
- [3] TKSTAR LK109 USER MANUAL.
<https://www.manualslib.com/manual/1035746/Tkstar-Lk109.html>.
- [4] TK102B Manual.pdf.
<https://www.munstergps.ie/wp-content/uploads/TK102B-Manual.pdf>.
- [5] Cobertura 3G / 4G / 5G en Buenos Aires - nPerf.com.
<https://www.nperf.com/es/map/AR>.
- [6] Traccar. Protocols. <https://www.traccar.org/protocols/>.
- [7] Espressif Systems. *ESP32 Series Datasheet*. https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.
- [8] u-blox. *NEO-6 Data Sheet*. https://content.u-blox.com/sites/default/files/products/documents/NEO-6_DataSheet_%28GPS.G6-HW-09005%29.pdf.
- [9] Wikipedia. *Hayes AT command set*.
https://en.wikipedia.org/wiki/Hayes_AT_command_set.
- [10] Simcom. *SIM800L Hardware Design*. https://www.makerhero.com/img/files/download/Datasheet_SIM800L.pdf.
- [11] Sherpany. *Home - Django REST Framework*.
<https://www.djangoproject-rest-framework.org/>.
- [12] Lionel Sujay Vailshery. *Most used web frameworks among developers worldwide, as of 2023*.
<https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/>. Jul. de 2023.
- [13] Angular. *Angular - What is Angular?*
<https://angular.io/guide/what-is-angular>.
- [14] Angular. *Angular - Introduction to Angular concepts*.
<https://angular.io/guide/architecture>.
- [15] Angular. *Getting Started | Angular Material*.
<https://material.angular.io/guide/getting-started>.
- [16] BlueHalo. *bluehalo/ngx-leaflet: Core Leaflet package for Angular.io*.
<https://material.angular.io/guide/getting-started>.
- [17] The PostgreSQL Global Development Group. *PostgreSQL: About*.
<https://www.postgresql.org/about/>.
- [18] solid IT gmbh. *DB-Engines Ranking - popularity ranking of database management systems*. <https://db-engines.com/en/ranking>.
- [19] Romaric Philogène. *Heroku vs AWS: What to choose as a startup in 2023?*
<https://www.qovery.com/blog/heroku-vs-aws-what-to-choose-as-a-startup>. Dic. de 2020.

- [20] SumatoSoft. *Heroku vs AWS: Which Cloud Hosting to choose in 2022?* <https://sumatosoft.medium.com/heroku-vs-aws-which-cloud-hosting-to-choose-in-2022-a9f2f1959aeb>. Dic. de 2023.
- [21] Edward Jones. *Google Cloud vs AWS (Comparing the Giants).* <https://kinsta.com/blog/google-cloud-vs-aws/>. Sep. de 2023.
- [22] Docker Inc. *Overview of the get started guide.* <https://docs.docker.com/get-started/>.
- [23] Microsoft. *Documentation for Visual Studio Code.* <https://code.visualstudio.com/docs>.
- [24] Ltd. Espressif Systems (Shanghai) Co. *Get Started - ESP32 - ESP-IDF Programming Guide v5.1.2 documentation.* <https://docs.espressif.com/projects/esp-idf/en/v5.1.2/esp32/get-started/index.html>.
- [25] PlatformIO. *PlatformIO IDE - PlatformIO latest documentation.* <https://docs.platformio.org/en/latest/integration/ide/pioide.html>.
- [26] SEMU Consulting. *Python Graphical GPS Client Application supporting NMEA, UBX, RTCM3, NTRIP & SPARTN Protocols.* <https://github.com/semuconsulting/PyGPSClient>.
- [27] Wikipedia. *Twilio.* <https://en.wikipedia.org/wiki/Twilio>.
- [28] Ivan Grokhotkov. *igrr/libnmea-esp32: ESP-IDF component for libnmea, NMEA parser.* <https://github.com/igrr/libnmea-esp32>.
- [29] North Coast Media. *What Exactly Is GPS NMEA Data? - GPS World.* <https://www.gpsworld.com/what-exactly-is-gps-nmea-data/>.
- [30] Ltd. Espressif Systems (Shanghai) Co. *Button - ESP-IoT-Solution latest documentation.* https://docs.espressif.com/projects/esp-iot-solution/en/latest/input_device/button.html.
- [31] Inc. Analog Devices. *Debounce | Analog Devices.* <https://www.analog.com/en/resources/glossary/debounce.html>.
- [32] Texas Instruments. *LM2596 SIMPLE SWITCHER Power Converter 150-kHz 3-A Step-Down Voltage Regulator datasheet (Rev. G).* <https://www.ti.com/lit/ds/symlink/lm2596.pdf>.
- [33] Hexagon AB. *GPGGA.* <https://docs.novatel.com/OEM7/Content/Logs/GPGGA.htm>.
- [34] Ltd. Espressif Systems (Shanghai) Co. *Power Management - ESP32 - ESP-IDF Programming Guide v5.2.2 documentation.* https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/power_management.html.
- [35] u-blox. *Power Management. Considerations with u-blox 6 GPS receivers.* https://content.u-blox.com/sites/default/files/products/documents/u6-PowerMgt_AppNote_%28GPS.G6-X-10014%29.pdf.
- [36] Daniele Varrazzo y The Psycopg Team. *psycopg 3.2.0.dev1 documentation.* <https://www.psycopg.org/psycopg3/docs/>.
- [37] Randall Degges. *Django-Twilio — django-twilio 0.9.0 documentation.* <https://django-twilio.readthedocs.io/en/latest/>.
- [38] Django Software Foundation. *Django Channels — Channels 4.1.0 documentation.* <https://channels.readthedocs.io/en/latest/>.
- [39] encode. *Uvicorn.* <https://www.unicorn.org/>.
- [40] Sherpany. *Serializers - Django REST Framework.* <https://www.django-rest-framework.org/api-guide/serializers/>.
- [41] Sherpany. *ViewSets - Django REST Framework.* <https://www.django-rest-framework.org/api-guide/viewsets/>.

- [42] Django Software Foundation. *Middleware* | *Django Documentation* | *Django*. <https://docs.djangoproject.com/en/5.0/topics/http/middleware/>.
- [43] Django Software Foundation. *Signals* | *Django Documentation* | *Django*. <https://docs.djangoproject.com/en/5.0/topics/signals/>.
- [44] IBM. *What is multi-tenant?* | *IBM*. <https://www.ibm.com/topics/multi-tenant>.
- [45] Chakray. *Swagger and Swagger UI: What is it and Why is it a must for your APIs?* <https://www.chakray.com/swagger-and-swagger-ui-what-is-it-and-why-is-it-a-must-for-your-apis/>.
- [46] Google. *File structure • Angular*. <https://angular.dev/reference/configs/file-structure>.
- [47] Google. *NgRx - Why use NgRx Store for State Management?* <https://ngrx.io/guide/store/why>.
- [48] Google. *Pipes • Overview • Angular*. <https://angular.dev/guide/pipes>.
- [49] MDN contributors. *CSS: Cascading Style Sheets* | MDN. <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [50] Mark Otto. *Grid system* | *Bootstrap v5.3*. <https://getbootstrap.com/docs/5.3/layout/grid/>.
- [51] Vanessa Marely Aristizabal Angel. *Arquitectura de Componentes en Angular*. <https://medium.com/notasdeangular/arquitectura-de-componentes-en-angular-761e9226fd9f>.
- [52] Dionysia Lemonaki. *Snake Case VS Camel Case VS Pascal Case VS Kebab Case – What's the Difference Between Casings?* <https://developer.mozilla.org/en-US/docs/Web/CSS>.
- [53] Wikipedia contributors. *Exponential backoff — Wikipedia, The Free Encyclopedia*. [Online; accedido 25-June-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Exponential_backoff&oldid=1230499135.
- [54] Thales Group. *What is an IMEI number and what is it for?* | *Thales Group*. <https://www.thalesgroup.com/en/markets/digital-identity-and-security/inspired/basics-of-phone-security/imei-number>.
- [55] Heroku. *Dynos and the Dyno Manager* | *Heroku Dev Center*. <https://devcenter.heroku.com/articles/dynos>.
- [56] Heroku. *Add-ons Overview* | *Heroku Dev Center*. <https://devcenter.heroku.com/articles/add-ons>.
- [57] auth0.com. *JSON Web Tokens - Introduction - jwt.io*. <https://jwt.io/introduction>.