



Escuela Técnica Superior de
Ingeniería Informática

TRABAJO FIN DE GRADO

Sistema de control de flota

**Grado en Ingeniería Informática - Ingeniería de
Computadores**

Realizado por

José María León Andrades

Dirigido por

Ángel Francisco Jiménez Fernández

Departamento

Arquitectura y Tecnología de Computadores

Sevilla, Julio 2025

1. Resumen

Este proyecto nace del objetivo de desarrollar un sistema portátil que permita monitorear la localización en tiempo real, entre otras funciones, de un dispositivo llamado **WAIT** (siglas de *Where Am I aT*). Para ello, se usarán diferentes sensores y actuadores que, tras un procesamiento, nos permitirán conocer la ubicación del dispositivo.

El sistema consta de una placa ESP32 cargada con un firmware que, conectada a un sensor GPS, irá recibiendo señales del satélite y, gracias a la API de Google, mostrará la ubicación en tiempo real a través de una app asociada. También incluye un sensor acelerómetro que detecta movimiento y un módulo GSM que permitirá enviar los datos de ambos sensores, desde cualquier ubicación con cobertura, a una base de datos. Además, tendrá un actuador (alarma) que emitirá señales a través de una app o cuando se supere una distancia definida.

Los datos de ubicación recogidos por la placa a través del módulo GPS se enviarán a una base de datos desarrollada en SQL alojada en una Raspberry Pi 3 modelo B, que actuará como servidor web. Paralelamente, una app desarrollada en Android Studio, usando Java, realizará peticiones GET/POST a la base de datos para mostrar la ubicación en tiempo real e iniciar la alarma cuando sea necesario.

En resumen, tendremos un dispositivo localizador y antirrobo que permitirá además la detección de accidentes, junto con una aplicación móvil que actuará como interfaz gráfica para el usuario final.

2. Summary

This project was born from the goal of developing a portable system capable of monitoring the real-time location, among other functions, of a device called **WAIT** (an acronym for *Where Am I at*). To achieve this, various sensors and actuators will be used, and after processing, they will allow us to determine the device's position.

The system is based on an ESP32 board loaded with firmware. Connected to a GPS sensor, it will receive satellite signals and, thanks to the Google API, display the real-time location through an associated app. Additionally, it includes an accelerometer sensor to detect movement and a GSM module that transmits data from both sensors to a database from any location with mobile coverage. The device also features an actuator (alarm) that can emit signals via the app or automatically trigger when a predefined distance is exceeded.

The location data collected by the board through the GPS module will be sent to an SQL database hosted on a Raspberry Pi 3 Model B, which will act as a web server. In parallel, a mobile app developed in Android Studio using Java will perform GET/POST requests to the database to display the real-time location and activate the alarm when necessary.

In summary, this project results in a locator and anti-theft device capable of accident detection, paired with a mobile application serving as a graphical interface for the end user.

3. Índice	
1. Resumen.....	1
2. Summary	2
4. Índice de Figuras	5
5. Objetivos del proyecto	7
5.1. Objetivos profesionales.....	7
5.2. Objetivos educativos	8
6. Introducción	9
7. Estado del arte.....	11
7.1. Invoxia GPS Tracker.....	11
7.2. TK905 GPS Tracker.....	13
7.3. Samsung SmartThings Tracker	15
8. Tecnologías empleadas	17
8.1. ESP32.....	17
8.1.1. Pin Layout.....	18
8.2. Módulo GPS micro USB NEO-6M	21
8.2.1. Esquema Interno Conceptual del Dispositivo.....	22
8.3. Módulo Acelerómetro GY-61 (ADXL335).....	23
8.3.1. Esquema Interno Conceptual del Dispositivo.....	23
8.3.2. Características Técnicas y Operativas.....	24
8.4. Módulo alarma	24
8.5. Raspberry Pi 3 Model B.....	25
8.5.1. Especificaciones.....	26
8.5.2. Especificaciones del Procesador y Memoria	26
8.5.3. Conectores.....	27
8.6. PlatformIO en Visual Studio Code	28
8.7. MariaDB	30
8.8. Eclipse con Vert.x	31
8.9. Android Studio.....	32
9. Arquitectura del Sistema.....	33
9.1. Componentes.....	33

10. Sistema Desarrollado	34
10.1. Componentes del sistema.....	34
10.1.1. Componentes del dispositivo	34
10.1.1.1. Modulo GPS NEO-6M.....	34
10.1.1.2. Módulo Acelerómetro GY-61 (ADXL335).....	37
10.1.1.3. Módulo GSM SIM800L.....	39
10.1.1.4. Módulo alarma	41
10.1.1.5. Microcontrolador ESP32.....	42
10.1.2. Componentes del servidor.....	44
10.1.2.1. Raspberry Pi.....	44
10.1.3. Componentes de la aplicación.....	46
10.1.3.1. Aplicación Android	46
10.2. Procedimiento	48
11. Pruebas del sistema	50
12. Planificación temporal	60
12.1. Investigación y Aprendizaje.....	60
12.2. Desarrollo Hardware.....	60
12.3. Desarrollo Software	61
12.3.1. Servidor backend con Vert.x y Maven en Eclipse	61
12.3.2. Base de datos con MariaDB.....	61
12.3.3. Firmware con PlatformIO	62
12.3.4. Aplicación móvil con Android Studio	62
12.4. Pruebas y corrección de errores	62
12.5. Visualización temporal.....	64
13. Costes.....	65

4. Índice de Figuras

Figura 1: Invoxia GPS Tracker	12
Figura 2: TK905 GPS Tracker	14
Figura 3: Samsung SmartThings Tracker.....	16
Figura 4: ESP32-WROOM32	17
Figura 5: ESP32-WROOM32 - Pin Layout	18
Figura 6: Módulo GPS micro USB NEO-6M.....	21
Figura 7: Módulo Acelerómetro GY-61	24
Figura 8: Módulo alarma.....	24
Figura 9: Raspberry Pi 3 Model B	25
Figura 10: Raspberry Pi – GPIO layout	26
Figura 11: Visual Studio.....	28
Figura 12: PlatformIO	29
Figura 13: MariaDB	30
Figura 14: Vert.x.....	31
Figura 15: Android Studio.....	32
Figura 16: Uso de librería TinyGPS.....	35
Figura 17: Imports de librerías para ESP32 en PlatformIO	43
Figura 18: Tablas para los sensores y sus estados	45
Figura 19: Uso de API Google.....	46
Figura 20: Código para mostrar los datos del GPS por el puerto serie	51
Figura 21; Datos obtenidos por el GPS.....	51
Figura 22: Código para obtener datos del sensor acelerómetro	52
Figura 23: Datos obtenidos por el sensor acelerómetro(En reposo)	53
Figura 24: Datos obtenidos por el sensor acelerómetro(Sensor girado)	53
Figura 25: Librerías usadas en el firmware.....	53
Figura 26: Inicialización de variables / Asignación de ID a componentes Hardware	54
Figura 27: Lanzamiento de aplicación en servidor	55
Figura 28: Datos obtenidos por el sensor GPS(Vista Servidor).....	55
Figura 29: Datos obtenidos por el sensor Acelerómetro(Vista Servidor)	56
Figura 30: Usuario añadido con el endpoint Register(Vista Servidor)	56
Figura 31: Dispositivos asociados a usuarios(Vista Servidor).....	56
Figura 32: Logs de aplicación(Vista Servidor)	57
Figura 33: Interfaz Register	57
Figura 34: Interfaz Login	58
Figura 35: Interfaz añadir nuevo dispositivo	58
Figura 36: Interfaz página de inicio	58
Figura 37: Interfaz visualización de mapa	58
Figura 38: Interfaz notificaciones	59
Figura 39: Notificación cuando se detecta un movimiento.....	59

Figura 40: Diagrama de Gantt..... 64

5. Objetivos del proyecto

En el desarrollo de este proyecto se han establecido varios objetivos. Entre los objetivos profesionales se plantea crear un sistema portátil capaz de capturar la ubicación y detectar posibles accidentes en tiempo real, así como su aplicación. Como objetivos educativos se busca aprender y mejorar las habilidades en hardware, software, desarrollo de aplicaciones móviles, análisis y procesamiento de datos.

5.1. Objetivos profesionales

- **Desarrollo de dispositivo portátil**

Desarrollo de un dispositivo portátil capaz de recibir, procesar y enviar señales GPS, así como detectar cambios de movimiento y emitir sonido en forma de alarma. Dispositivo geolocalizable como sistema antirrobo y detector de accidentes.

- **Geolocalización**

Implementación de sensor GPS micro USB NEO-6M capaz de obtener ubicación precisa en tiempo real a través de tramas NMEA, y posterior conversión a latitud y longitud para integración con la API de Google.

- **Detección de accidente**

Integración de sensor acelerómetro que captará los cambios de movimientos en los ejes X e Y, capaz de detectar posibles accidentes.

- **Alarma detectora**

Uso de actuador en forma de alarma para emitir sonido en caso de posible robo.

- **Gestión de datos**

Desarrollo de un servidor, corriendo sobre una Raspberry Pi, usando Vert.x como framework que permitirá consultas a la base de datos para la inserción y obtención de datos en tiempo real.

- **Aplicación móvil**
Aplicación móvil que servirá como interfaz gráfica para que los usuarios puedan consultar la ubicación de sus dispositivos en cualquier momento.
- **Almacenamiento de datos**
Implementación de un modelo de base de datos relacional, basado en SQL, para almacenar los datos mostrados por el dispositivo.

5.2.Objetivos educativos

- **Desarrollo hardware**
Ampliar la comprensión y la habilidad para trabajar con tecnologías emergentes, sensores, módulos GPS y su integración.
- **Mejora de habilidades en software**
Desarrollar competencias en programación utilizando lenguajes como C++ y Java.
- **Gestión y análisis de datos**
Adquirir conocimiento para procesar, almacenar y filtrar la información recogida por sensores, empleando bases de datos relacionales SQL.
- **Empleo de APIs**
Aprender a implementar APIs y gestionar los datos que se obtienen a través de ellas.
- **Creación de aplicaciones web**
Alcanzar dominio en el diseño y desarrollo de aplicaciones móviles con interacción para el usuario, utilizando Java en Android Studio.
- **Desarrollo integral**
Ser capaz de construir un sistema IoT aplicando todos los conocimientos adquiridos a lo largo del grado.

6. Introducción

En el contexto actual, marcado por la rápida evolución de la tecnología, vivimos en un entorno cada vez más interconectado. El desarrollo de dispositivos inteligentes y la expansión del Internet de las Cosas (IoT) han dado lugar a una nueva generación de sistemas capaces de ofrecer información en tiempo real y brindar soluciones efectivas a problemas cotidianos. Este avance ha abierto nuevas puertas en materia de seguridad y protección personal, especialmente en ámbitos como el transporte, la movilidad urbana, la vigilancia y el monitoreo remoto.

Fruto de esta transformación tecnológica surge **WAIT** (*Where Am I at*), un sistema portátil de localización en tiempo real, concebido como una herramienta antirrobo, de detección de accidentes y control de desplazamiento. Este dispositivo ha sido diseñado con el objetivo de brindar a los usuarios una solución confiable, autónoma y precisa, que no solo registre la ubicación de un objeto o persona, sino que también sea capaz de identificar comportamientos anómalos y emitir alertas cuando sea necesario. En definitiva, WAIT responde a una necesidad creciente: la de mantener siempre el control sobre lo que más importa.

El robo de vehículos, el extravío de objetos valiosos o la falta de seguimiento en tiempo real de personas vulnerables como menores o adultos mayores, representan hoy en día un desafío importante tanto para la seguridad individual como colectiva. A ello se suma la necesidad de una respuesta inmediata ante accidentes, algo crucial en entornos de alta movilidad. Si bien existen soluciones en el mercado, muchas presentan limitaciones en cuanto a portabilidad, autonomía o capacidad de integración con otras tecnologías. WAIT se plantea como una solución completa, de bajo costo y gran flexibilidad, capaz de operar en distintos escenarios y con una infraestructura accesible, gracias al uso de componentes ampliamente disponibles y estándares abiertos en su arquitectura.

Enfocándonos en la parte hardware, el corazón del sistema es una placa **ESP32**, un microcontrolador potente y versátil, que ejecuta un firmware personalizado desarrollado específicamente para la gestión eficiente de sensores y transmisión de datos. El ESP32 está conectado a una serie de componentes clave:

- **Módulo GPS:** Permite la recepción de señales satelitales para obtener coordenadas geográficas precisas, fundamentales para el rastreo en tiempo real.
- **Sensor acelerómetro:** Detecta movimientos en los ejes X, Y y Z, útil para identificar desplazamientos, vibraciones inusuales o posibles impactos, lo que lo hace aplicable en casos de colisiones o intentos de manipulación del dispositivo.

- **Módulo GSM:** Proporciona conectividad a redes móviles, permitiendo la transmisión de datos incluso en ubicaciones remotas, sin necesidad de depender de redes Wi-Fi.
- **Actuador (alarma):** Puede activarse automáticamente si el dispositivo supera una distancia definida o de forma remota desde la app, funcionando como disuasivo ante posibles robos o situaciones de emergencia.

Todos estos componentes han sido seleccionados bajo criterios de eficiencia energética, portabilidad y confiabilidad.

Enfocándonos en la parte software, el sistema se articula a través de una aplicación móvil desarrollada en Android Studio (Java). Esta app se conecta con una base de datos SQL alojada en una Raspberry Pi 3 Modelo B, que actúa como servidor central. Mediante peticiones HTTP GET y POST, la app consulta en tiempo real la ubicación del dispositivo, permite configurar el umbral de distancia para la activación de la alarma e integra la visualización del dispositivo o recibir notificaciones en caso de alerta.

La base de datos relacional asegura la integridad y persistencia de los datos, a la vez que permite escalar el sistema en caso de que se añadan nuevos dispositivos o funcionalidades en el futuro.

WAIT no es simplemente un rastreador GPS, es una plataforma de seguridad IoT integrada, capaz de responder de forma activa ante diferentes eventos. Algunas de sus aplicaciones prácticas incluyen:

- Antirrobo para vehículos, permitiendo la localización inmediata tras un movimiento no autorizado.
- Monitoreo de personas en situación de dependencia, como menores, adultos mayores o personas con discapacidad.
- Detección de accidentes mediante el análisis de datos del acelerómetro, lo que puede derivar en una alerta inmediata para solicitar asistencia.
- Sistemas de geofencing para controlar si un dispositivo sale de un área determinada.

Su tamaño compacto y diseño portátil permiten que sea fácilmente integrable en mochilas, vehículos o dispositivos personales.

WAIT representa una propuesta innovadora que pone al alcance del usuario común una herramienta de seguimiento y seguridad de alta eficiencia, basada en tecnologías abiertas y ampliamente accesibles. Su arquitectura modular permite futuras mejoras, como la incorporación de sensores adicionales, conexión a servicios en la nube o integración con asistentes inteligentes. Este proyecto destaca por su aplicabilidad, en una sociedad en constante movimiento. **WAIT** es una respuesta concreta a los desafíos del presente, que combina tecnología, conectividad y protección en un solo dispositivo.

7. Estado del arte

Con el reciente crecimiento de los IoT's en los últimos años, impulsado por los avances en Internet, existen múltiples soluciones comerciales destinadas al monitoreo de vehículos, objetos personales o personas vulnerables mediante sistemas GPS. Sin embargo, muchas de estas soluciones presentan limitaciones en cuanto a personalización, costo o escalabilidad.

A continuación, se analizan algunas de las alternativas más relevantes en el mercado actual, en relación con las funciones que ofrece el sistema **WAIT**, evaluando sus capacidades, alcance, ventajas y desventajas frente al dispositivo propuesto.

7.1. Invoxia GPS Tracker

Invoxia es una empresa tecnológica francesa especializada en dispositivos IoT inteligentes, especialmente conocidos por sus GPS Trackers para coches, bicicletas, mascotas y objetos personales. Fundada en 2010 por Éric Carreel (cofundador de Withings) y Serge Renouard, es una empresa consolidada en el desarrollo de IoT, especialmente orientada al control de flota.

Además del **Invoxia GPS Tracker**, que presenta grandes similitudes con **WAIT**, ha desarrollado otros productos como el **Bike Tracker**, un modelo discreto con forma de reflector, premiado en el CES.

El **Invoxia GPS Tracker** es un dispositivo de localización portátil diseñado principalmente para el rastreo de vehículos, bicicletas, mochilas o personas. Se caracteriza por su diseño compacto, peso ligero y larga autonomía. A diferencia de otros rastreadores tradicionales que utilizan redes GSM exclusivamente, este dispositivo opera sobre redes **LoRa** (Low Range Wide Area Network) o **LTE-M** (Long Term Evolution for Machines), dependiendo del país y de la infraestructura de

red disponible. Esta solución se ha vuelto popular en Europa y Estados Unidos gracias a su diseño compacto, autonomía extendida y conectividad eficiente.

El dispositivo también ofrece un modo antirrobo y se comunica con la aplicación móvil a través de servidores en la nube. Es resistente al agua y puede colocarse en distintos tipos de superficies, tanto en vehículos como en pertenencias personales. El coste del dispositivo se encuentra en torno a los 129 €, aunque requiere el pago de una suscripción mensual o anual para poder usar su red de comunicación LoRa o LTE-M.

La solución está pensada para un uso general, tanto personal como profesional, y tiene una fuerte presencia en el mercado europeo gracias a su simplicidad y autonomía.

A pesar de sus múltiples virtudes, el Invoxia GPS Tracker presenta ciertas limitaciones que pueden ser importantes dependiendo del contexto de uso, como funcionalidad cerrada. Como producto comercial cerrado, la personalización o integración con otros sistemas externos puede estar limitada o restringida, lo que puede ser un obstáculo para desarrolladores que busquen adaptar el dispositivo a necesidades específicas.

El proyecto **WAIT** se posiciona como una solución flexible y adaptable en el ámbito de localización y monitoreo IoT, incorporando aprendizajes y características que buscan superar algunas de las limitaciones observadas en dispositivos comerciales como el Invoxia GPS Tracker.



Figura 1: Invoxia GPS Tracker

7.2.TK905 GPS Tracker

La empresa desarrolladora del dispositivo **TK905 GPS Tracker** se llama Shenzhen Juneo Technology Co., Limited, con filiales como TKSTAR Technology Co., Limited y WINNES Technology Co., Limited, que forman parte de un único grupo dedicado a desarrollar soluciones de rastreo GPS y comunicaciones inalámbricas. Ubicada en la provincia de Guangdong, China, vende sus productos en plataformas como Alibaba.

El TK905 es un dispositivo GPS autónomo diseñado específicamente para el rastreo de vehículos. Cuenta con una carcasa robusta y un imán integrado que facilita su instalación en cualquier parte metálica del automóvil sin necesidad de cables o modificaciones.

Este dispositivo funciona sobre redes GSM 2G y permite recibir información a través de mensajes SMS o una aplicación móvil vinculada, así como mediante una plataforma web en la que se pueden consultar datos de ubicación, historial de trayectos y configuración de alarmas.

Aquí están algunas de sus características:

- Precisión GPS de hasta 5 metros.
- Autonomía de hasta 90 días (en espera).
- Imán integrado para fijación en superficies metálicas.
- Alarma por exceso de velocidad, vibración y salida de zona.
- Botón SOS para emergencias.

Además, el TK905 integra funciones de escucha remota mediante una tarjeta SIM, lo que lo hace útil para situaciones de vigilancia o recuperación de vehículos robados. El dispositivo se comercializa ampliamente por un precio estimado de entre 60 € y 80 €, sin cuotas mensuales, aunque requiere el uso de una tarjeta SIM con datos móviles para su funcionamiento completo.

A pesar de sus características atractivas, el TK905 presenta ciertas desventajas que deben considerarse, como por ejemplo, la dependencia de la red GSM 2G. El dispositivo utiliza redes 2G, que están siendo progresivamente desactivadas en muchos países, lo que puede afectar su funcionalidad futura y cobertura.

El proyecto **WAIT** busca posicionarse como una alternativa más flexible y adaptable en el ámbito del rastreo y monitoreo GPS, abordando algunas de las limitaciones de dispositivos comerciales como el TK905, principalmente mejorando en el aspecto de compatibilidad con múltiples tecnologías de comunicación WAIT no depende exclusivamente de redes GSM 2G, sino que puede integrarse con diferentes protocolos de comunicación más actuales y escalables, asegurando mayor cobertura y durabilidad a largo plazo.



Figura 2: TK905 GPS Tracker

7.3.Samsung SmartThings Tracker

Samsung, una de las empresas tecnológicas más reconocidas y consolidadas a nivel mundial, no solo ha liderado el mercado en sectores como teléfonos inteligentes, televisores y semiconductores, sino que también ha apostado con fuerza por el creciente universo del Internet de las Cosas (IoT). Como parte de esta expansión estratégica, la compañía ha desarrollado una serie de soluciones inteligentes para el hogar y la conectividad, entre las que destaca el **Samsung SmartThings Tracker**.

El **Samsung SmartThings Tracker** representa una propuesta innovadora en el ámbito del rastreo de objetos y personas. Es un dispositivo GPS que forma parte del ecosistema del hogar inteligente SmartThings. Está diseñado para el rastreo de personas, mascotas o bienes personales y opera sobre la red LTE-M, lo que permite una conexión eficiente con bajo consumo de energía.

Este rastreador cuenta con una carcasa pequeña, resistente al agua (certificación IP68), y una batería recargable con una duración de hasta 7 días en condiciones normales de uso. Se puede conectar a la aplicación SmartThings, desde donde el usuario puede establecer zonas seguras, recibir alertas si el dispositivo entra o sale de dichas zonas, acceder al historial de localizaciones y configurar diferentes perfiles de uso.

Una de las funciones destacadas es la capacidad de integrarse con otros dispositivos del ecosistema Samsung SmartThings, como cámaras, luces o sensores de movimiento. Por ejemplo, puede activar una cámara si detecta que un objeto está en movimiento fuera de la zona de seguridad preestablecida.

Este dispositivo se comercializa principalmente en Estados Unidos, con un precio inicial de aproximadamente 90 €, al que se debe sumar un plan de datos mensual para su funcionamiento. Aunque su uso está más enfocado al entorno familiar y de domótica, también ha sido empleado en aplicaciones comerciales ligeras como rastreo de entregas o monitorización de trabajadores en campo.

Es una solución moderna que busca fusionar la movilidad con el hogar inteligente, manteniendo un diseño atractivo, liviano y funcional para la vida diaria.

A pesar de sus ventajas y tecnología avanzada, este dispositivo presenta ciertas limitaciones que es importante considerar. La principal sería Compatibilidad limitada al ecosistema Samsung para aprovechar al máximo las capacidades del tracker, es necesario estar integrado dentro del ecosistema SmartThings, lo que puede restringir su uso para quienes no posean otros dispositivos Samsung o prefieran soluciones más abiertas.

Eso lo solucionamos con WAIT, ya que tiene Independencia tecnológica, WAIT no está condicionado a operar en una red específica ni a un ecosistema cerrado, lo que permite utilizar diversas tecnologías de comunicación y adaptarse a distintas infraestructuras de red, ampliando su aplicabilidad geográfica.



Figura 3: Samsung SmartThings Tracker

8. Tecnologías empleadas

8.1.ESP32

La placa **ESP32-WROOM32** es un módulo diseñado por Espressif Systems. Contiene un potente y genérico módulo MCU (Microcontroller Unit) dedicado a una gran variedad de aplicaciones. Su módulo está destinado a una amplia variedad de aplicaciones, desde sensores de baja potencia, codificación de voz, transmisión de música o decodificación MP3.

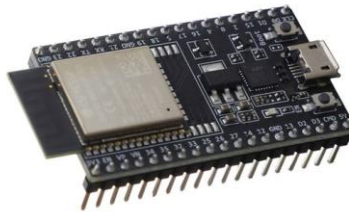


Figura 4: ESP32-WROOM32

El dispositivo tiene una antena PCB integrada en el propio diseño, se trata de una antena de traza que se crea sobre el cobre en la PCB. La antena tiene una frecuencia de 2.4GHz, compatible con Wi-Fi y Bluetooth.

En el núcleo de la placa se encuentra el chip **ESP32-D0WD-V3** o **ESP32 D0WDR2-V3**, diseñado para ser escalable y adaptable. El dispositivo tiene una frecuencia de reloj ajustable desde 80 MHz a 240 MHz. El chip tiene un coprocesador de bajo consumo que es el encargado de realizar tareas que no requieren tanta potencia.

La integración de Bluetooth y Wifi permite que se pueda usar en una gran gama de aplicaciones de forma polivalente; permite alcance físico y conexión a internet a través de un enrutador Wifi, mientras que el uso del Bluetooth permite conectarnos al teléfono o transmitir balizas de baja energía para su detección.

El ESP32 en reposo trabaja con una corriente inferior a 5 μ A, lo que hace que funcione especialmente bien en aplicaciones portátiles y alimentadas por batería. El módulo tiene una velocidad de datos de hasta 150 Mbps y una potencia de salida de 20 dBm en la antena.

El sistema operativo de la ESP32 es **freeRTOS** con **LwIP** y seguridad **TLS 1.2**.

La ESP32 tiene un diseño robusto, capaz de funcionar de forma fiable en todos los entornos con una temperatura de funcionamiento entre -40 °C y +125 °C. El dispositivo es capaz de eliminar imperfecciones de circuitos y adaptarse a cambios en condiciones extremas.

8.1.1.Pin Layout

La disposición de pines de **ESP32-WROOM-32UE** es la misma que la de ESP32 WROOM-32E, excepto que ESP32-WROOM-32UE no tiene zona de exclusión.

Los pines GPIO6 a GPIO11 en el chip ESP32-D0WD-V3 y ESP32-D0WDR2-V3 están conectados al SPI Flash integrado en el módulo y no son LED. En las variantes de módulo que tienen QSPI PSRAM incorporado, es decir, que incorporan ESP32 D0WDR2-V3, IO16 está conectado a la PSRAM integrada y no se puede utilizar para otras funciones.

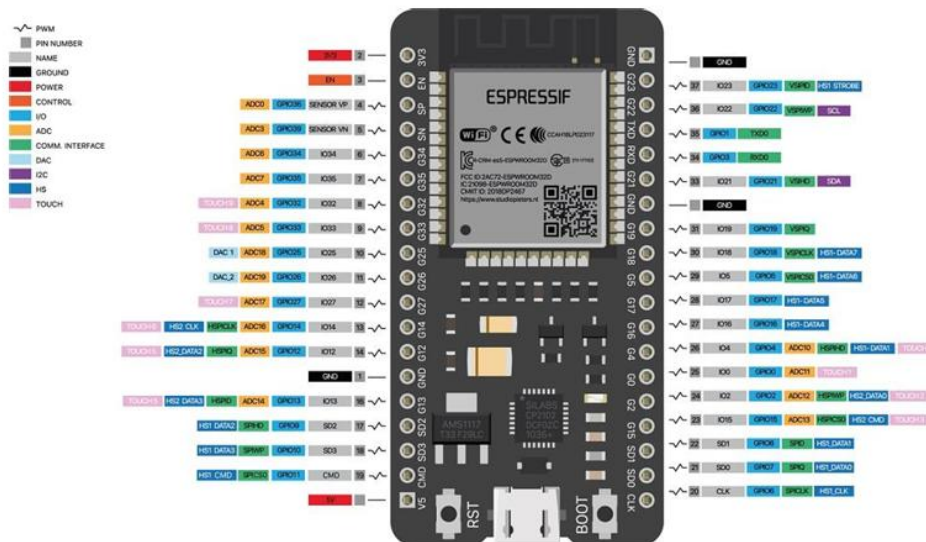


Figura 5: ESP32-WROOM32 - Pin Layout

GPIO	Input	Output	Notes	GPIO
0	pulled up	OK	outputs PWM signal at boot, must be LOW to enter flashing	0
1	TX pin	OK	debug output at boot	1
2	OK	OK	connected to on-board LED, must be left floating or LOW	2
3	OK	RX pin	HIGH at boot	3
4	OK	OK		4
5	OK	OK	outputs PWM signal at boot, strapping pin	5
6	x	x	connected to the integrated SPI flash	6
7	x	x	connected to the integrated SPI flash	7
8	x	x	connected to the integrated SPI flash	8
9	x	x	connected to the integrated SPI flash	9

10	x	x	connected to the integrated SPI flash	10
11	x	x	connected to the integrated SPI flash	11
12	OK	OK	boot fails if pulled high, strapping pin	12
13	OK	OK		13
14	OK	OK	outputs PWM signal at boot	14
15	OK	OK	outputs PWM signal at boot, strapping pin	15
16-33	OK	OK		16-33
34	OK		input only	34
35	OK		input only	35
36	OK		input only	36
39	OK		input only	39

8.2. Módulo GPS micro USB NEO-6M

El módulo GPS **NEO-6M** es una solución de posicionamiento satelital desarrollada por u-blox, ampliamente utilizada en proyectos de localización por su precisión, fiabilidad y bajo consumo energético. Se trata de un dispositivo compacto que permite obtener coordenadas geográficas en tiempo real utilizando señales de satélites GPS. El módulo viene comúnmente integrado con una antena cerámica, una batería de respaldo y una interfaz de comunicación serie TTL o USB.

Este sensor se utiliza para determinar la posición (latitud, longitud, altitud), velocidad y tiempo, funcionando mediante la recepción de señales GNSS. Cada versión representa una mejora respecto a la anterior en cuanto a velocidad de adquisición, precisión y número de satélites soportados. El módulo está diseñado para una integración sencilla con microcontroladores como ESP32, Arduino o Raspberry Pi.

Además, está optimizado para uso en exteriores y puede mantener el posicionamiento con una precisión aproximada de 2.5 metros. Funciona sobre una interfaz UART, permitiendo una rápida comunicación con microcontroladores para la adquisición de datos GPS.

El módulo está fabricado cumpliendo las normativas RoHS, lo que garantiza la ausencia de materiales peligrosos. Tiene un amplio rango de funcionamiento, siendo ideal para proyectos como rastreadores, drones, sistemas de navegación y aplicaciones IoT.



Figura 6: Módulo GPS micro USB NEO-6M

8.2.1. Esquema Interno Conceptual del Dispositivo

El módulo GPS está compuesto por los siguientes elementos:

- Chip receptor GNSS u-blox (modelos NEO-6, NEO-7 u NEO-8).
- Antena cerámica pasiva o activa.
- EEPROM para configuración.
- Regulador de voltaje.
- Batería de respaldo RTC (Real Time Clock) para mantener los datos cuando se pierde alimentación.

A continuación, se describen los parámetros clave del módulo GPS NEO-6M, NEO-7M y NEO-8M. Estos valores permiten comprender su funcionamiento y capacidades operativas:

Parámetro	NEO-6M	Unidad
Voltaje de operación	2.7 – 3.6	V
Precisión de posición	~2.5	m
Tiempo inicial (TTFF)	27 (frío), 1 (caliente)	s
Protocolos soportados	NMEA, UBX	—
Sensibilidad	-161	dBm
Corriente promedio	45 – 65	mA
Temperatura operativa	-40 a +85	°C
Número de canales	22	—

8.3.Módulo Acelerómetro GY-61 (ADXL335)

El **GY-61** es un módulo de acelerómetro analógico de tres ejes basado en el chip **ADXL335** de Analog Devices. Está diseñado para detectar aceleraciones lineales en los ejes X, Y y Z, permitiendo medir tanto movimientos como la orientación espacial del dispositivo donde se encuentra instalado.

Este sensor es ideal para aplicaciones portátiles por su bajo consumo, pequeño tamaño y respuesta rápida ante variaciones de movimiento. Su funcionamiento se basa en un sistema microelectromecánico (MEMS) capaz de generar una señal de salida analógica proporcional a la aceleración medida en cada eje.

El módulo es ampliamente utilizado en proyectos donde se desea detectar caídas, vibraciones, movimientos bruscos, inclinación o impactos, siendo especialmente útil en sistemas de seguridad como **WAIT**, para la detección de accidentes o desplazamientos inesperados.

8.3.1.Esquema Interno Conceptual del Dispositivo

El módulo **GY-61** está compuesto por:

- **Chip ADXL335** de Analog Devices.
- Filtro RC pasivo para suavizar las señales analógicas.
- Pines de salida analógica para los tres ejes: **X, Y, Z**.
- Pines de alimentación de 3.3V o 5V.

El chip ADXL335 convierte las aceleraciones físicas en señales de voltaje. La salida en estado de reposo (sin aceleración) se encuentra en torno a **1.65V** (en alimentación de 3.3V), y varía hacia arriba o hacia abajo dependiendo de la dirección y magnitud de la aceleración, con una sensibilidad de **~330 mV/g**.

8.3.2. Características Técnicas y Operativas

A continuación, se detallan las principales características técnicas del **GY-61** y su chip base ADXL335:

Parámetro	Valor Típico	Unidad
Rango de medición	± 3	g
Sensibilidad	~ 330	mV/g
Tensión de alimentación	3.0 – 5.0	V
Corriente típica	350	μA
Salida de señal	Analógica	—
Dimensiones del módulo	$\sim 19 \times 15$	mm
Temperatura de operación	-40 a +85	$^{\circ}\text{C}$
Ruido en la señal	< 150	$\mu\text{g}/\sqrt{\text{Hz}}$
Tiempo de respuesta	< 1	ms

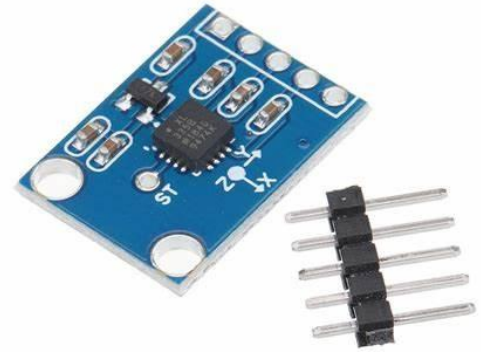


Figura 7: Módulo Acelerómetro GY-61

8.4. Módulo alarma

El módulo AZ Delivery KY-012 es un sensor de alarma del buzzer, es un componente electrónico que sirve para generar una alarma. Este buzzer crea un sonido de frecuencia de 2.5 kHz. Se trata de un módulo activo, es decir, no necesita una onda cuadrada a diferencia de los pasivos. Si obtiene un voltaje mínimo de 3,3 V en su pin de señal, el sensor creará una onda cuadrada por sí solo.

Este dispositivo tiene una tensión de alimentación de 3.3 V hasta 5 V, un consumo de corriente de 10 mA y un nivel de sonido de 85dB.



Figura 8: Módulo alarma

8.5.Raspberry Pi 3 Model B

La Raspberry Pi es un dispositivo creado en 2012 por Raspberry Pi Foundation, con el pensamiento de utilizarla en la enseñanza de la computación en escuelas y universidades. En sus inicios se lanzaron dos modelos, el modelo A y el modelo B .



Figura 9: Raspberry Pi 3 Model B

La Raspberry Pi 3 Model B es el modelo de Raspberry de tercera generación. Actualmente es el modelo más popular de Raspberry Pi. Este dispositivo es un computador con un tamaño muy reducido, puede ser conectado a un monitor mediante el HDMI y usarse con teclado y ratón [16]. Este dispositivo está incorporado con el sistema operativo Linux e incluye todo lo necesario para programar el dispositivo con diversos lenguajes. Su procesador es 10 veces más rápido que el de la Raspberry Pi de primera generación, además, dispone de conectividad LAN inalámbrica y Bluetooth.

Puede ser utilizada para hacer las tareas comunes de un ordenador de escritorio. También es útil para realizar conexiones con el mundo exterior, como reproductores de música, estaciones meteorológicas o montar sistemas IoT.

Este dispositivo tiene un GPIO de 40 pines con los que puede conectarse con el exterior, ya sea con sensores o con actuadores. Su GPIO funciona con 3.3 V, para conectar sensores que funcionan con 5 V será necesario un convertor de niveles lógicos.

La Raspberry Pi no dispone de convertor analógico a digital, para leer sensores analógicos tendremos que usar un ADC externo. El dispositivo dispone de puertos de comunicación I2C, SPI y UART.

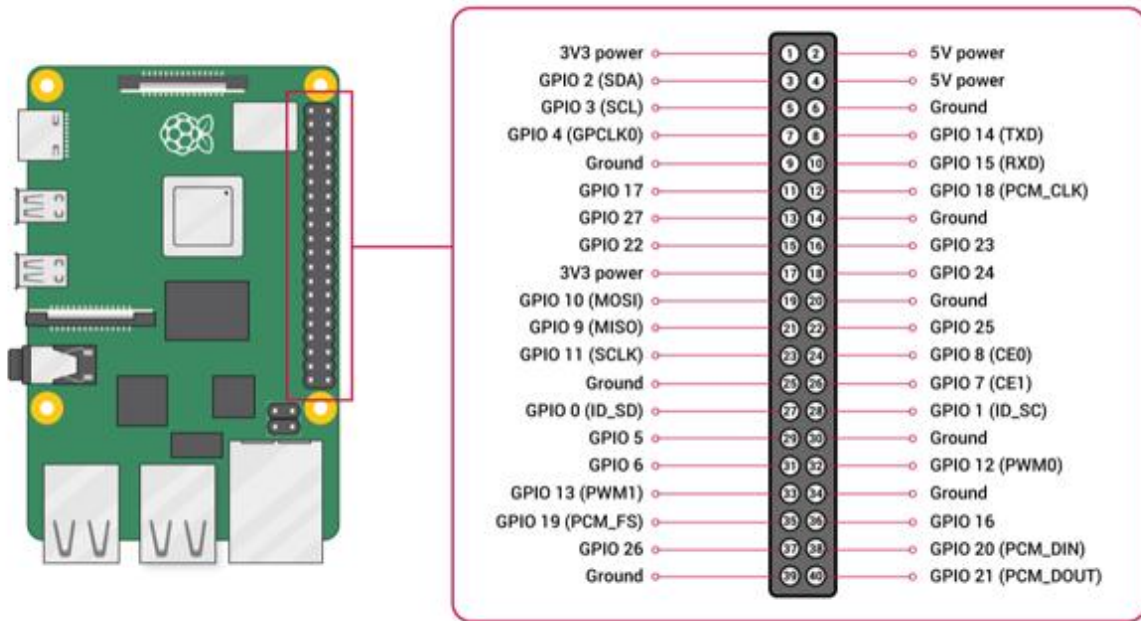


Figura 10: Raspberry Pi – GPIO layout

8.5.1. Especificaciones

La **Raspberry Pi 3 Model B** incorpora un procesador **Broadcom BCM2837**, un **CPU System on Chip (SoC)**. Este tipo de chip integra la **CPU**, **GPU** y diversos componentes esenciales para el funcionamiento de un computador.

8.5.2. Especificaciones del Procesador y Memoria

CPU

- Arquitectura ARMv8-A
- Quad-core
- Frecuencia del reloj: 1.2 GHz
- Núcleo: ARM Cortex-A53

GPU

- Coprocesador multimedia VideoCore IV de doble núcleo
- Soporte para OpenVG acelerado por hardware
- Decodificación de alto perfil 1080p30

MEMORIA

- 1 GB LPDDR2

8.5.3. Conectores

CONECTORES	CARACTERÍSTICAS
ETHERNET	Ethernet 10/100 BaseT
SALIDA DE VIDEO	HDMI (rev. 1.3 y 1.4), RCA compuesto (PAL y NTSC)
SALIDA DE AUDIO	Conector jack de 3,5 mm, HDMI
USB	4 x puertos USB 2.0
CONECTOR GPIO	Conector de expansión de 40 pines, paso de 2,54 mm
CONECTOR DE CÁMARA	Interfaz serie de cámara MIPI de 15 pines (CSI-2)
CONECTOR DE PANTALLA	Conector plano de 15 vías, interfaz serie de pantalla (DSI), con dos líneas de datos y una línea de reloj

8.6. PlatformIO en Visual Studio Code

PlatformIO es un entorno de desarrollo integrado (IDE) moderno, multiplataforma y de código abierto que funciona como una extensión dentro de Visual Studio Code. Ha sido diseñado para facilitar la programación de microcontroladores y sistemas embebidos, como ESP32, Arduino, STM32, entre otros.



Figura 11: Visual Studio

A diferencia del Arduino IDE tradicional, PlatformIO ofrece un ecosistema avanzado que incluye gestión de bibliotecas, soporte para múltiples entornos de compilación, integración con sistemas de control de versiones, y compatibilidad con una amplia gama de plataformas y frameworks embebidos.

PlatformIO utiliza como lenguaje de programación una versión estándar de C/C++, manteniendo compatibilidad con bibliotecas de Arduino. Durante el proceso de compilación, PlatformIO transforma el código fuente en código binario, que luego puede ser cargado directamente al microcontrolador a través de un puerto USB.

Entre sus características más destacadas, se encuentra su integración con Visual Studio Code, lo cual proporciona una interfaz poderosa con autocompletado, navegación entre archivos, herramientas de depuración, control de versiones (Git) y mucho más.

Además, PlatformIO cuenta con un potente sistema de gestión de proyectos basado en el archivo `platformio.ini`, donde se configuran detalles como la tarjeta objetivo, las bibliotecas requeridas, flags de compilación, entornos personalizados, entre otros.

PlatformIO representa una evolución del desarrollo embebido, permitiendo un flujo de trabajo más robusto y profesional comparado con otros entornos tradicionales. Es especialmente útil en proyectos complejos o colaborativos donde la organización, el control de versiones y la escalabilidad del código son fundamentales.



Figura 12: PlatformIO

8.7.MariaDB

MariaDB es un sistema de gestión de bases de datos relacional (RDBMS) de código abierto, derivado de MySQL y totalmente compatible con este. Fue creado como una bifurcación (fork) de MySQL tras su adquisición por Oracle, con el objetivo de garantizar que el proyecto se mantuviera libre y con una comunidad activa.

MariaDB utiliza el lenguaje de consulta estructurado SQL (Structured Query Language) para gestionar, manipular y consultar los datos en sus bases. Es ampliamente utilizada en aplicaciones web, sistemas empresariales y plataformas IoT gracias a su rendimiento, fiabilidad y facilidad de integración.

Este sistema puede ejecutarse tanto de manera local como remota y es capaz de gestionar grandes volúmenes de información de forma eficiente. Se integra perfectamente con distintos lenguajes de programación y entornos de desarrollo, como Python, PHP, Node.js, C/C++, Java, entre otros.

Entre sus principales características se encuentra el uso de motores de almacenamiento como InnoDB, MyISAM o Aria, así como soporte para replicación, procedimientos almacenados, triggers, vistas y funciones definidas por el usuario.

MariaDB se utiliza comúnmente en combinación con servidores web (como Apache o Nginx) dentro de pilas de software como LAMP (Linux, Apache, MySQL/MariaDB, PHP/Python) o MEAN/MERN para proyectos web y de IoT.



Figura 13: MariaDB

8.8.Eclipse con Vert.x

Eclipse es un entorno de desarrollo integrado (IDE) de código abierto, ampliamente utilizado para programar en lenguajes como Java, C/C++, Python, entre otros. Se caracteriza por su arquitectura modular basada en plugins, lo que le permite extender sus funcionalidades fácilmente.

En el contexto del desarrollo de aplicaciones modernas, Eclipse puede utilizarse junto con Vert.x, un framework de desarrollo reactivo para la creación de aplicaciones asincrónicas, escalables y de alto rendimiento sobre la Java Virtual Machine (JVM).

Vert.x está diseñado para aprovechar al máximo los procesadores multinúcleo, utilizando un modelo de ejecución basado en eventos (event-driven), lo que lo hace ideal para aplicaciones que requieren alta concurrencia y baja latencia, como servicios REST, servidores web, microservicios o componentes IoT.

Gracias a su naturaleza no bloqueante y ligera, Vert.x es una excelente opción para proyectos en tiempo real, donde se gestionan múltiples eventos o conexiones simultáneas.



Figura 14: Vert.x

8.9. Android Studio

Android Studio es el entorno de desarrollo integrado (IDE) oficial para el desarrollo de aplicaciones en el sistema operativo Android. Está basado en IntelliJ IDEA de JetBrains y es distribuido por Google como software gratuito y de código abierto.

Este IDE proporciona un entorno completo y potente que permite diseñar, desarrollar, probar, compilar y depurar aplicaciones móviles Android desde una sola plataforma. Ofrece un conjunto de herramientas robustas que facilitan el desarrollo de aplicaciones modernas tanto para smartphones como para tabletas.

Android Studio utiliza Gradle como sistema de automatización de compilación, lo que permite gestionar dependencias, construir múltiples variantes de la aplicación y automatizar tareas comunes. Además, permite desarrollar tanto en Java como en Kotlin, lenguajes oficiales para Android, y utilizar XML para la definición de interfaces gráficas.

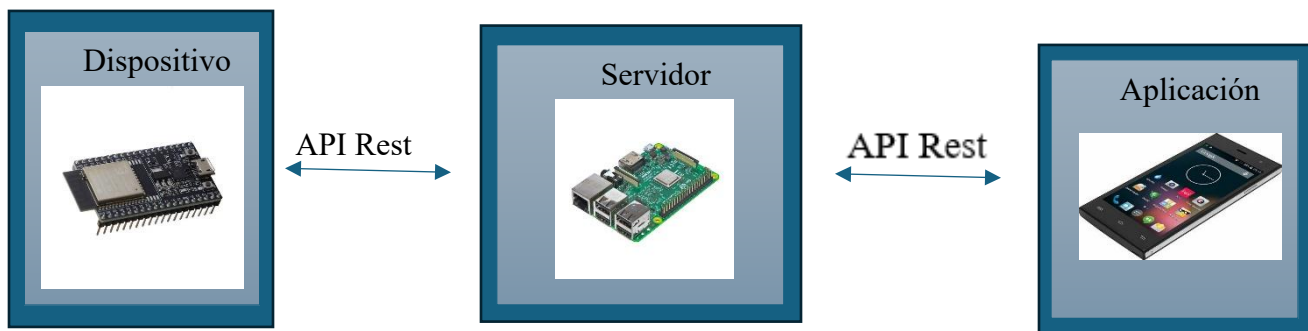


Figura 15: Android Studio

9. Arquitectura del Sistema

La arquitectura del sistema WAIT (Where Am I aT) ha sido diseñada con el objetivo de ofrecer una solución de monitoreo portátil, inteligente y robusta, orientada a la seguridad personal y patrimonial. Esta arquitectura está basada en la integración de múltiples sensores y módulos de comunicación, todos conectados a una unidad central de procesamiento que permite la gestión eficiente de los datos, la generación de alertas y el control remoto del dispositivo.

El núcleo funcional de WAIT es una placa ESP32, un microcontrolador altamente versátil que cuenta con conectividad WiFi y Bluetooth, además de capacidad de procesamiento suficiente para manejar múltiples entradas y ejecutar firmware personalizado. La ESP32 es responsable de la adquisición de datos provenientes de los sensores integrados en el sistema, su preprocesamiento y su posterior envío mediante comunicación serial o inalámbrica a una Raspberry Pi 3, que actúa como servidor y núcleo de respaldo.



9.1. Componentes

El dispositivo tiene integrado 1 actuador y 3 sensores para conseguir el funcionamiento deseado:

- **Módulo alarma:** Este actuador es una alarma que permitirá emitir sonido facilitando así su detección en distancias cortas.
- **Módulo Acelerómetro GY-61 (ADXL335):** Proporciona cambios de movimientos y giro, gracias a él podremos detectar cambios bruscos de movimiento o giro permitiéndonos identificar posibles accidentes.
- **Módulo GPS micro USB NEO-6M:** Emite datos de ubicación precisa en tiempo real, para tras un procesado poder mostrarla en la interfaz de usuario.

- **Módulo GSM:** Con este módulo podemos dotar de conectividad a internet al dispositivo desde cualquier ubicación.
- **Microcontrolador ESP32:** El microcontrolador ESP32 es el encargado de recopilar todos los datos de los sensores conectados a él y de activar la alarma según unos criterios cargados en el firmware. En definitiva, es el corazón del dispositivo.
- **Servidor (Raspberry Pi):** La Raspberry Pi será la encargada de desplegar la aplicación corriendo un archivo .jar donde están desarrolladas las consultas API, además de contener la base de datos relacional. Se comunicará con el dispositivo y la aplicación móvil.
- **Interfaz de usuario:** La interfaz de usuario será una aplicación móvil desarrollada en Android Studio que permitirá a los usuarios finales ver la ubicación de su dispositivo y gestionar las alertas.

10. Sistema Desarrollado

El sistema desarrollado trata de un dispositivo portátil que permitirá monitorear la ubicación en tiempo real y detectar posibles accidentes o robos. Acompañado de una base de datos para el almacenamiento y análisis de la información corregida y una aplicación móvil para que el usuario final pueda consultar el estado de sus dispositivos en cualquier momento. En esta sección veremos el proceso llevado a cabo para desarrollar el sistema desde partes más específicas hasta acabar con el proceso general.

10.1. Componentes del sistema

10.1.1. Componentes del dispositivo

10.1.1.1. Módulo GPS NEO-6M

Este sensor es el componente principal del proyecto. Cumple una función esencial al proporcionar datos de ubicación geográfica en tiempo real. Estos datos permiten al sistema conocer en todo momento la posición exacta del dispositivo, lo que es fundamental para el correcto funcionamiento de las funcionalidades de seguimiento, localización y contexto espacial que requiere el proyecto.

Para obtener las coordenadas geográficas y poderlas mostrar por la aplicación, se utiliza la librería TinyGPS, que permite decodificar fácilmente los datos del protocolo NMEA enviados por el módulo GPS. Gracias a esta librería, se obtienen de forma directa los valores de latitud y longitud, que luego pueden ser utilizados por el sistema para determinar la ubicación actual del dispositivo y enviar estos datos a un servidor para posteriormente mostrarlos en una interfaz.

```

if(gps.encode(c))
{
    gps.f_get_position(&latitude, &longitude);
    Serial.print("Latitud/Longitud: ");
    Serial.print(latitude,5);
    Serial.print(", ");
    Serial.println(longitude,5);

    gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);
    Serial.print("Fecha: "); Serial.print(day, DEC); Serial.print("/");
    Serial.print(month, DEC); Serial.print("/"); Serial.print(year);
    Serial.print(" Hora: "); Serial.print(hour, DEC); Serial.print(":");
    Serial.print(minute, DEC); Serial.print(":"); Serial.print(second, DEC);
    Serial.print("."); Serial.println(hundredths, DEC);
    Serial.print("Altitud (metros): ");
    Serial.println(gps.f_altitude());
    Serial.print("Rumbo (grados): "); Serial.println(gps.f_course());
    Serial.print("Velocidad(kmph): ");
    Serial.println(gps.f_speed_kmph());
    Serial.print("Satelites: "); Serial.println(gps.satellites());
    Serial.println();
    gps.stats(&chars, &sentences, &failed_checksum);
    snprintf(fechaHora, sizeof(fechaHora), "%04d-%02d-%02d %02d:%02d:%02d",
        year, month, day, hour, minute, second);
    POST_tests();
}
}

```

Figura 16: Uso de libreria TinyGPS

La conexión del módulo GPS a la ESP32 se realiza mediante los siguientes pines:

- VCC del GPS → conectado al pin de 3.3V de la ESP32.
- GND → conectado al GND de la ESP32.
- TX (Transmisión del GPS) → conectado al pin RX2 (GPIO 16) de la ESP32.
- RX (Recepción del GPS) → conectado al TX2 (GPIO 17) de la ESP32.

Este módulo viene equipado con una antena de cerámica tipo patch, diseñada específicamente para mejorar la recepción de señales satelitales.

Durante la integración del módulo GPS, se presentaron diversos desafíos, siendo el principal la correcta recepción de señal satelital. Comenzando por el aspecto técnico de la conexión con el microcontrolador, el primer obstáculo surgió al intentar utilizar la UART0, la cual se encuentra ocupada por el puerto serial destinado a la programación y depuración del ESP32 (utiliza los GPIO1 y GPIO3), lo que imposibilita su uso para otros fines. Posteriormente, se intentó usar la UART1; sin embargo, esta también presentó limitaciones, ya que comparte pines con la memoria

flash del dispositivo (GPIO9 y GPIO10), los cuales están reservados y no deben utilizarse en muchos modelos de ESP32.

Finalmente, se optó por emplear la UART2, que permitió establecer una conexión funcional con el módulo GPS. No obstante, surgió un desafío importante relacionado con la ubicación física del dispositivo. Dado que este tipo de sensores requiere línea de vista directa con los satélites, la recepción de señal se ve considerablemente afectada en entornos cerrados. Por ello, fue necesario ubicar el dispositivo en espacios abiertos para garantizar una recepción óptima.

Una vez establecida la conexión y colocado el módulo en una ubicación adecuada, el sensor comenzó a emitir tramas no decodificables (comúnmente llamadas “basura”) durante los primeros minutos. Este comportamiento se debe al tiempo necesario para adquirir señal satelital, que puede oscilar entre 10 y 15 minutos bajo condiciones normales. Además, se observó que el módulo GPS es particularmente delicado, especialmente en lo relativo a su antena. En mi experiencia, fue necesario adquirir tres módulos distintos: los dos primeros, sin antena integrada, dejaron de funcionar correctamente tras múltiples conexiones y desconexiones. El tercer módulo, que sí contaba con antena integrada, fue el que finalmente se utilizó en el proyecto.

Respecto al desarrollo de software, el principal reto fue interpretar correctamente las tramas GPS. Para ello, se empleó la librería **TinyGPS**, que permite traducir las cadenas NMEA a valores legibles de latitud y longitud. Durante las pruebas iniciales, realizadas con sensores defectuosos (sin saber aún que lo estaban), la librería no mostraba datos válidos, a pesar de que las tramas eran visibles en el monitor serie. Tras un análisis más exhaustivo, se comprobó que estas tramas no cumplían con el formato NMEA estándar, lo que impedía su decodificación. Esto motivó el reemplazo por un nuevo sensor funcional. A partir de entonces, la librería comenzó a operar correctamente, extrayendo y mostrando los datos de posición en el formato esperado.

10.1.1.2. Módulo Acelerómetro GY-61 (ADXL335)

En el proyecto WAIT, el módulo **MPU6050** cumple un rol esencial en la detección de movimiento y orientación. Este sensor combina un acelerómetro y un giroscopio de tres ejes, permitiendo medir tanto la aceleración como la velocidad angular. Esta información es crucial para detectar eventos físicos como caídas, movimientos bruscos o cambios de orientación.

Para facilitar la lectura de los datos del MPU6050, se utilizó la librería Wire.h, que permite la comunicación mediante el protocolo I2C, junto con funciones personalizadas que acceden directamente a los registros del sensor. La lectura de datos crudos se toma desde los registros internos y se convierte a unidades físicas comprensibles: g (aceleración) para el acelerómetro y °/s (grados por segundo) para el giroscopio.

Pasos para la lectura de datos:

1. **Inicialización I2C:** Comunicación entre la ESP32 y el MPU6050 usando GPIO 21 (SDA) y GPIO 22 (SCL).
2. **Activación del sensor:** Salida del modo de suspensión mediante el registro PWR_MGMT_1 escribiendo 0x00.
3. **Configuración de escalas:** $\pm 2g$ para el acelerómetro y ± 250 °/s para el giroscopio, configurados en los registros ACCEL_CONFIG y GYRO_CONFIG.
4. **Lectura de datos crudos:** Acceso a los registros ACCEL_XOUT, GYRO_XOUT, y otros, leyendo 14 bytes secuenciales.
5. **Conversión a datos físicos:**
 - Aceleración (X, Y, Z) = valor crudo / 16384.0
 - Velocidad angular (X, Y, Z) = valor crudo / 131.0

```

void readRawMPU()
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(ACCEL_XOUT);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 14);

  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();

  Tmp = Wire.read() << 8 | Wire.read();

  GyX = Wire.read() << 8 | Wire.read();
  GyY = Wire.read() << 8 | Wire.read();
  GyZ = Wire.read() << 8 | Wire.read();

  // Conversión a unidades físicas
  ax = AcX / 16384.0; // Aceleración en g
  ay = AcY / 16384.0;
  az = AcZ / 16384.0;

  gx = GyX / 131.0; // Velocidad angular en °/s
  gy = GyY / 131.0;
  gz = GyZ / 131.0;
}

```

Figura : Conversión de datos

Conexiones a la ESP32:

- VCC → 3.3V de la ESP32
- GND → GND
- SDA → GPIO 21
- SCL → GPIO 22

Se accede directamente a los registros del sensor mediante la librería **Wire.h**. Esto permite un mayor control sobre la configuración del sensor y facilita un procesamiento más optimizado para los requerimientos del proyecto.

Los datos se formatean manualmente mediante operaciones de desplazamiento de bits ($\ll 8$) para combinar los bytes alto y bajo. Cada par de bytes se combina en un valor de 16 bits con signo. (El primero es el MSB y el segundo es el LSB), y luego se convierten a sus respectivas unidades dividiendo por los factores estándar definidos por el fabricante (16384 para aceleración y 131 para giroscopio).

Esta información procesada se puede usar tanto para la detección de condiciones físicas del entorno del dispositivo como para ser enviada al servidor y visualizarse en la aplicación, permitiendo interpretar la actividad física del dispositivo en tiempo real.

La integración del **sensor acelerómetro MPU6050** dentro del proyecto resultó ser considerablemente más sencilla en comparación con el módulo GPS. Sin embargo, este proceso también implicó ciertos desafíos específicos que fue necesario abordar durante el desarrollo.

Uno de los principales retos consistió en **definir un umbral adecuado para la detección de posibles accidentes**. Al no contar inicialmente con experiencia previa en el uso de acelerómetros, fue necesario realizar una investigación teórica acerca del comportamiento típico de estos sensores y su sensibilidad ante movimientos bruscos. Posteriormente, se llevaron a cabo pruebas, simulando distintos escenarios de impacto o movimiento repentino, con el fin de establecer un valor de aceleración que pudiera considerarse indicativo de un evento anómalo o riesgoso. Esta fase fue crucial para evitar tanto **falsos positivos** (detecciones erróneas por movimientos normales) como **falsos negativos** (accidentes reales que no se detectan).

A diferencia del módulo GPS, cuyo funcionamiento está condicionado a factores externos como la visibilidad al cielo abierto y que requiere un **tiempo de espera considerable (10-15 minutos)** para la sincronización con los satélites, el acelerómetro **ofrece lecturas inmediatas** tras su inicialización. Esta característica representa una ventaja significativa durante el proceso de desarrollo y pruebas, ya que permite recibir los datos inmediatamente facilitando la **calibración de parámetros y la validación funcional del sistema** sin necesidad de tiempos de espera.

10.1.1.3. Módulo GSM SIM800L

El módulo GSM SIM800L cumple una función clave en el proyecto, ya que se encarga de permitir la comunicación móvil del sistema a través de la red celular. Este componente es fundamental para enviar mensajes de alerta, realizar llamadas de emergencia o transmitir datos hacia servidores remotos en casos donde no exista conexión WiFi disponible. Gracias a este módulo, el dispositivo puede mantenerse conectado en cualquier lugar con cobertura GSM, lo que amplía significativamente su alcance y autonomía.

Para el manejo de este módulo, se utiliza la librería SoftwareSerial para establecer una comunicación por puerto serial entre el SIM800L y la ESP32. Los datos que se intercambian entre

el microcontrolador y el módulo GSM están basados en comandos AT, que permiten realizar operaciones como enviar SMS, verificar el estado de la red, o iniciar llamadas.

Conexión del módulo GSM SIM800L a la ESP32:

- **VCC del SIM800L** → conectado a una fuente externa de 3.7–4.2V.
- **GND** → conectado al GND de la ESP32.
- **TX (del SIM800L)** → conectado al RX2 (GPIO 16) de la ESP32.
- **RX (del SIM800L)** → conectado al TX2 (GPIO 17) de la ESP32, a través de un divisor de voltaje para evitar daño por el nivel lógico de 3.3V.

El módulo GSM se comunica en modo texto utilizando **comandos AT**. Estos comandos permiten estructurar mensajes SMS de forma sencilla. A diferencia del GPS, no requiere decodificación de protocolos complejos como NMEA. En lugar de ello, el procesamiento de datos se basa en enviar cadenas de texto y esperar respuestas desde el módulo que indiquen el estado de la operación (por ejemplo, OK, ERROR, o el número de caracteres enviados).

El flujo básico para enviar un mensaje o recibir uno es:

1. Configurar el modo de mensaje (AT+CMGF=1).
2. Definir el número de destino (AT+CMGS="número").
3. Escribir el contenido del mensaje.
4. Confirmar el envío con el carácter ASCII 26 (CTRL+Z).

La integración del módulo GSM con el dispositivo presentó inicialmente un problema importante relacionado con la conexión física a los puertos UART del microcontrolador ESP32. Como se mencionó anteriormente, el módulo requiere una conexión UART para poder comunicarse correctamente, pero la UART2 ya estaba asignada y en uso por el módulo GPS. Este conflicto de recursos obligó a buscar una solución para poder incorporar ambos módulos sin interferencias.

Después de investigar las opciones disponibles, se decidió realizar un **reasignamiento de los pines UART** en la ESP32. En lugar de usar la UART2 ocupada, se configuraron los pines GPIO26 y GPIO27 para que funcionaran como la UART1, reservándolos para la comunicación con el módulo GSM. Esta reconfiguración permitió mantener ambas conexiones UART activas y evitar colisiones en la transmisión de datos.

Además, para proteger el módulo GSM de posibles daños por diferencias en los niveles de voltaje, se incorporó un **módulo level shifter basado en MOSFET**. Este dispositivo actúa como un adaptador de nivel lógico que previene picos o caídas de voltaje que podrían afectar la integridad del módulo GSM SIM800L, el cual es sensible a variaciones de voltaje en sus líneas de comunicación. La implementación de este circuito elevó la confiabilidad y la durabilidad del sistema, asegurando una comunicación estable y segura entre la ESP32 y el módulo GSM.

10.1.1.4. Módulo alarma

Este actuador es un componente esencial en el proyecto, ya que cumple la función de generar señales sonoras de alarma o aviso. Su principal utilidad radica en emitir un sonido audible para alertar al usuario ante ciertas condiciones o eventos detectados por el sistema, lo que facilita la comunicación inmediata sin necesidad de monitoreo visual constante.

Para controlar el buzzer y producir sonidos o tonos, se utiliza la función **tone()** disponible en Arduino, que permite generar señales PWM de diferentes frecuencias sobre el pin conectado al buzzer. Esto posibilita emitir tonos simples o secuencias sonoras según lo requiera la lógica del proyecto.

La conexión del módulo buzzer piezoeléctrico a la ESP32 se realiza de la siguiente manera:

- **VCC del buzzer** → **conectado al pin de 3.3V o 5V de la ESP32** (según especificación del buzzer).
- **GND** → **conectado al GND de la ESP32**.
- **Pin de señal** → **conectado a un pin GPIO digital de la ESP32** (por ejemplo, GPIO 25) para controlar la activación del sonido.

10.1.1.5. Microcontrolador ESP32

La **ESP32** es una familia de microcontroladores muy versátil que integra conectividad **Wi-Fi** y **Bluetooth**, lo que la convierte en una opción popular entre desarrolladores y entusiastas de la electrónica. Su alto rendimiento, flexibilidad y compatibilidad con múltiples protocolos de comunicación la hacen ideal para una amplia gama de aplicaciones.

Este dispositivo puede programarse en diversos entornos de desarrollo y soporta varios lenguajes, lo que facilita su integración en distintos proyectos. Entre las plataformas más comunes para programarla se encuentran:

- **Arduino IDE**
- **ESP-IDF** (Espressif IoT Development Framework)
- **MicroPython**
- **PlatformIO**

Gracias a sus características, la ESP32 es una excelente opción para el desarrollo de soluciones en el campo del **Internet de las Cosas (IoT)**. Permite realizar tareas como el monitoreo y control remoto de dispositivos a través de la red, la recopilación de datos desde sensores y el envío de información a servidores en la nube. Además, también se utiliza en la creación de **wearables**, controladores de **robots**, **drones** y otros dispositivos móviles inteligentes.

Entre sus principales ventajas destacan su **bajo costo** en relación con todas las funcionalidades que ofrece, así como la gran **disponibilidad de recursos, bibliotecas y documentación**, tanto por parte del fabricante como de una comunidad muy activa que facilita el desarrollo y la solución de problemas.

El trabajo con el microcontrolador no ha supuesto una gran dificultad, principalmente porque ya contaba con experiencia previa en su uso antes de comenzar este proyecto. Esta familiaridad ha facilitado el proceso de desarrollo, permitiendo avanzar de forma estructurada y ordenada a lo largo de cada etapa.

En primer lugar, se realizó la conexión física de los distintos **sensores y actuadores** a los pines correspondientes del microcontrolador, siguiendo las especificaciones técnicas de cada componente para garantizar una integración adecuada. Posteriormente, se procedió con el

desarrollo del software, el cual fue llevado a cabo utilizando **Visual Studio Code**, en conjunto con el entorno de programación de Arduino.

A nivel de programación, se trabajó sobre el firmware del dispositivo, incorporando funciones esenciales como la configuración inicial de los sensores, la lectura de datos, su conversión a formatos comprensibles mediante el uso de bibliotecas específicas, así como el diseño de rutinas para la comunicación con una base de datos remota. También se implementaron funciones que permiten enviar y recibir información desde otros dispositivos o plataformas, facilitando así la interoperabilidad del sistema.

Durante todo este proceso se realizaron múltiples **pruebas de validación**, con el fin de comprobar el correcto funcionamiento del hardware y el software, asegurando que los datos capturados fueran precisos y que la comunicación entre componentes se diera sin errores. Gracias a este enfoque metódico, el desarrollo del sistema basado en el microcontrolador se llevó a cabo de forma eficiente y sin contratiempos significativos.

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s
framework = arduino
upload_port = COM4
monitor_speed = 115200
board_build.flash_mode = qio
board_build.flash_freq = 40m
lib_deps =
  mikalhart/TinyGPSPlus@^1.0.2
  bblanchon/ArduinoJson@^6.17.3
  arduino-libraries/NTPClient@^3.2.1
  knolleary/PubSubClient@^2.8
  dlloyddev/ESP32 ESP32S2 AnalogWrite@^4.3.4
  adafruit/Adafruit Unified Sensor@^1.1.9
  adafruit/DHT sensor library@^1.4.4
  mikalhart/TinyGPS
```

Figura 17: Imports de librerías para ESP32 en PlatformIO

10.1.2. Componentes del servidor

10.1.2.1. Raspberry Pi

La **Raspberry Pi** es una placa de desarrollo considerada un **microordenador de bajo coste**, diseñada originalmente con fines educativos, pero que ha alcanzado una enorme popularidad en proyectos de electrónica, automatización, robótica y aplicaciones del Internet de las Cosas (IoT). Su principal atractivo radica en su **accesibilidad económica**, su reducido tamaño y su gran **versatilidad**, lo que la convierte en una herramienta muy valorada tanto en entornos académicos como en proyectos personales tipo *Do It Yourself* (DIY).

Este dispositivo es capaz de ejecutar sistemas operativos completos basados en Linux, y permite realizar múltiples tareas de forma simultánea, gracias a su **procesador Quad-core ARM Cortex-A53** y **1 GB de memoria RAM LPDDR2** (en el caso del modelo Raspberry Pi 3). Estas especificaciones técnicas, si bien modestas en comparación con ordenadores tradicionales, son más que suficientes para ejecutar programas de control, scripts en Python, servicios web ligeros, procesamiento de sensores y otras aplicaciones propias de sistemas embebidos.

Para comenzar a utilizar una Raspberry Pi, es necesario realizar una **configuración inicial del sistema operativo**. Este proceso se puede llevar a cabo de manera sencilla mediante la herramienta oficial llamada **Raspberry Pi Imager**, disponible gratuitamente en la página web oficial del proyecto. Una vez descargada e instalada, esta aplicación permite seleccionar una imagen del sistema operativo que se desee utilizar —como Raspberry Pi OS u otras distribuciones compatibles—, e instalarla directamente en una tarjeta **microSD**, que actuará como unidad de almacenamiento principal del dispositivo.

Después de grabar la imagen, basta con insertar la tarjeta microSD en la Raspberry Pi y conectarla a una fuente de alimentación, una pantalla, teclado y ratón (o configurar el acceso remoto vía SSH si se prefiere). A partir de ese momento, la placa estará lista para ser utilizada como un ordenador funcional o como un centro de control para proyectos electrónicos más avanzados.

En el caso particular de mi proyecto, la Raspberry Pi desempeñará un papel fundamental como **servidor centralizado**. Su función principal será alojar y gestionar la **base de datos**, así como ejecutar la aplicación que servirá de interfaz para el sistema. Esta aplicación estará diseñada para ofrecer acceso mediante consultas API, facilitando la comunicación y el intercambio de información con otros sistemas externos que requieran interactuar con los datos almacenados.

Para lograr este despliegue, es imprescindible instalar el entorno de ejecución de **Java**, ya que la aplicación está empaquetada en un archivo ejecutable con extensión **.jar**. Este archivo contiene todo el código necesario para correr el servidor de aplicaciones, por lo que la instalación correcta de Java en la Raspberry Pi es un requisito indispensable para garantizar el funcionamiento óptimo del sistema.

Simultáneamente, es necesario configurar el sistema de gestión de bases de datos, para lo cual hemos optado por utilizar **MariaDB**, un sistema de gestión de bases de datos relacional (RDBMS) de código abierto que se basa en MySQL y ofrece una solución robusta, escalable y eficiente para el almacenamiento y gestión de datos. A diferencia de soluciones embebidas como SQLite, otras de las opciones planteadas, MariaDB opera bajo un modelo cliente-servidor, lo que significa que se ejecuta como un servicio independiente y permite gestionar múltiples conexiones simultánea.

```
CREATE TABLE sensorsAC(  
  idSensorsAC int NOT NULL AUTO_INCREMENT,  
  idDevice int NOT NULL,  
  removed tinyint(1) DEFAULT NULL,  
  PRIMARY KEY (idSensorsAC)  
)ENGINE=InnoDB AUTO_INCREMENT=1369 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;  
  
LOCK TABLES sensorsAC WRITE;  
INSERT INTO sensorsAC VALUES (71,1,1);  
UNLOCK TABLES;  
  
CREATE TABLE sensorsACStates(  
  idsensorsACStates int NOT NULL AUTO_INCREMENT,  
  idSensorAC int NOT NULL,  
  valueAc int NOT NULL,  
  valueGir int NOT NULL,  
  removed tinyint(1) DEFAULT NULL,  
  PRIMARY KEY (idsensorsACStates)  
)ENGINE=InnoDB AUTO_INCREMENT=1369 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_general_ci;
```

Figura 18: Tablas para los sensores y sus estados

Para asegurar una comunicación estable y eficiente entre la Raspberry Pi y otros dispositivos o sistemas externos que necesiten acceder a la base de datos o a la aplicación, es fundamental asignarle a la Raspberry Pi una **dirección IP fija** dentro de la red local. Esto evitará problemas derivados de cambios dinámicos de IP y facilitará la configuración de consultas y conexiones remotas a través de la red.

10.1.3. Componentes de la aplicación

10.1.3.1. Aplicación Android

La aplicación Android desarrollada para este proyecto cumple un rol esencial en la interacción del usuario con el sistema. Su principal función es permitir la visualización en tiempo real de los datos recolectados por los sensores conectados al dispositivo, así como facilitar la configuración y gestión de ciertos parámetros críticos del sistema. Para lograr esta interacción, la aplicación está diseñada para comunicarse eficazmente con el servidor web alojado en la Raspberry Pi, la cual actúa como núcleo central de procesamiento y almacenamiento de información.

Utilizando una API RESTful, el servidor expone diversos endpoints (puntos de acceso) que la aplicación Android consume para obtener información actualizada. La comunicación se realiza mediante solicitudes HTTP, principalmente peticiones GET para recuperar datos como la concentración de partículas de polvo, valores de sensores, estado del dispositivo y más.

Estas consultas API permiten que la app Android reciba datos en formato JSON, que luego son procesados y mostrados gracias al uso de la API de Google Maps en interfaces amigables y visualmente claras.

```
public void onMapReady(GoogleMap map) {  
    this.googleMap = map;  
    Log.d(tag: "MAP_DEBUG", msg: "Mapa listo");  
  
    if (locationLoaded && latitude != 0.0 && longitude != 0.0) {  
        addMarkerToMap(latitude, longitude);  
    } else {  
  
        LatLng defaultLocation = new LatLng( latitude: 37.7749, longitude: -122.4194); // San Francisco por defecto  
        googleMap.moveCamera(CameraUpdateFactory.newLatLngZoom(defaultLocation, zoom: 10));  
    }  
}
```

Figura 19: Uso de API Google

Además de la simple visualización de datos, la aplicación ofrece funcionalidades avanzadas de gestión y configuración del sistema. A través de peticiones HTTP POST o PUT a la API, el usuario puede modificar parámetros críticos, tales como:

- Cambiar el nombre del dispositivo para facilitar su identificación dentro de una red de sensores.
- Ajustar el umbral de detección para la alarma sonora, estableciendo el nivel de partículas contaminantes a partir del cual se activará una alerta.

Estas configuraciones enviadas desde la app se procesan en el servidor, actualizando la base de datos y configuraciones del sistema en tiempo real, permitiendo una gestión remota y personalizada de cada dispositivo.

Tanto la aplicación móvil como el dispositivo físico forman la interfaz directa entre el sistema y el usuario final. Por esta razón, se ha puesto un especial énfasis en que la parte visual de la app esté cuidadosamente diseñada y desarrollada, al mismo nivel de detalle y calidad que los demás componentes del proyecto, como el hardware y el backend.

La creación de esta aplicación móvil representó para mí un importante proceso de aprendizaje, ya que era la primera vez que desarrollaba una app para dispositivos móviles. Para facilitar el proceso y asegurar un desarrollo eficiente, decidí utilizar Java como lenguaje principal para la programación, complementándolo con XML para el diseño y estructura de las interfaces de usuario. Esta elección respondió a que, aunque Kotlin es una opción muy popular y moderna para el desarrollo Android, su curva de aprendizaje sería un desafío demasiado grande en un proyecto con limitaciones de tiempo y experiencia previa.

El desarrollo de la aplicación comenzó con la implementación de las funcionalidades básicas, que forman la base sobre la cual se construye el resto del sistema. La primera etapa consistió en crear la actividad principal o Main Activity, que sirve como punto de entrada a la aplicación y estructura la navegación general. Paralelamente, desarrollé un dashboard o panel de control, diseñado para mostrar de manera clara y accesible la información más relevante obtenida desde el dispositivo y el servidor, como la ubicación en tiempo real y el estado de los sensores.

Con estas bases establecidas, el siguiente paso fue abordar retos más complejos, como la creación de actividades adicionales que permitieran una experiencia más completa. Entre estas se incluyen la implementación de un sistema de autenticación de usuarios, esencial para proteger la información y asegurar que solo usuarios autorizados puedan acceder a las funcionalidades y datos sensibles. Además, se prestó atención al detalle en el diseño visual, asegurando una experiencia intuitiva y agradable mediante la incorporación de elementos gráficos, estilos consistentes y una navegación fluida.

A lo largo de este proceso, no solo se construyó una aplicación funcional, sino que también adquirí un profundo entendimiento de los principios y buenas prácticas del desarrollo móvil. Esto incluye la gestión de actividades y ciclos de vida, el manejo de recursos gráficos y layouts en XML, y la integración de servicios externos a través de APIs para la comunicación con el servidor. Todo esto ha sentado las bases para futuros desarrollos más avanzados y ha ampliado significativamente mi conjunto de habilidades como desarrollador.

10.2. Procedimiento

En esta sección veremos el procedimiento y la lógica seguida en el desarrollo del proyecto.

El punto de partida del proyecto fue la construcción del backend encargado de gestionar la comunicación con los dispositivos y aplicaciones clientes. Para ello, se utilizó Vert.x, un framework basado en la JVM diseñado para desarrollar aplicaciones reactivas, asíncronas y altamente escalables. Vert.x es especialmente adecuado para sistemas que requieren manejar múltiples conexiones concurrentes sin bloquear recursos, gracias a varias de sus características clave:

- **Modelo orientado a eventos (event-driven):** Vert.x funciona mediante un loop de eventos que procesa de manera eficiente callbacks y eventos sin bloquear los hilos de ejecución, lo que permite responder rápidamente a múltiples solicitudes simultáneas.
- **Operaciones no bloqueantes (non-blocking I/O):** Las operaciones de entrada y salida se ejecutan sin detener el hilo principal, favoreciendo la concurrencia y el rendimiento, algo fundamental para la comunicación constante con los dispositivos.
- **Event Bus interno:** Este sistema facilita el envío de mensajes asíncronos entre los diferentes componentes del servidor (verticles), e incluso entre máquinas diferentes, lo cual es especialmente útil para manejar las peticiones GET y POST recurrentes que se reciben desde los dispositivos y aplicaciones móviles.

Dentro del ecosistema Vert.x, se utilizó el módulo **Vert.x-Web**, que simplifica la creación de servidores web y la exposición de endpoints HTTP. Esto permitió construir una API robusta y flexible para que los clientes, ya sean aplicaciones móviles, dispositivos o navegadores web, pudieran realizar consultas, enviar datos o controlar remotamente los dispositivos conectados.

Uno de los avances más relevantes fue el desarrollo y la integración del sistema de autenticación dentro de los endpoints. Inicialmente, las consultas API eran simples peticiones GET y POST que interactuaban directamente con las tablas de la base de datos. Sin embargo, una vez incorporado

el sistema de autenticación, todas las consultas se adaptaron para incluir mecanismos de validación y autorización, asegurando que cada usuario accediera únicamente a la información y dispositivos que le correspondían. Por ejemplo, se implementó la funcionalidad para mostrar exclusivamente los dispositivos asociados al usuario autenticado, aumentando la seguridad y personalización del sistema.

Paralelamente al desarrollo del backend, se diseñó y modeló la base de datos utilizando MariaDB en un entorno local. Inicialmente, se creó un esquema simple con un número reducido de tablas para facilitar las pruebas funcionales y validar el flujo de datos, en este primer modelo se incluyeron las tablas de dispositivos, sensores y actuadores. Posteriormente se añadieron tablas asociadas a las anteriores para controlar el estado de los sensores en tablas separadas mejorando su escalabilidad. Finalmente, se añadieron tablas específicas para gestionar la información de usuarios, incluyendo credenciales y permisos, integrando así toda la estructura necesaria para la autenticación y autorización descrita anteriormente.

Una vez desarrollados los endpoints y configurada la base de datos, se realizaron pruebas utilizando Postman, una herramienta muy útil para simular y validar las peticiones HTTP. Estas pruebas permitieron verificar el correcto funcionamiento de las consultas API, asegurando que las operaciones de lectura, escritura y autenticación respondieran como se esperaba antes de avanzar en el desarrollo del dispositivo.

Con el backend consolidado, se procedió al desarrollo del dispositivo físico, comenzando por la parte hardware. Esto implicó el ensamblaje de los diferentes sensores y actuadores conectados al microcontrolador, un proceso que estuvo precedido por un análisis detallado de los datasheets y características de cada componente para garantizar la compatibilidad y el correcto funcionamiento.

Tras el ensamblado, se pasó a la etapa de desarrollo software. Se planteó una estructura clara para el código que permitiera gestionar de forma ordenada las funcionalidades necesarias: desde la lectura de datos en tiempo real desde los sensores, pasando por el control y activación de actuadores, hasta la comunicación bidireccional con el servidor mediante los endpoints previamente desarrollados.

Durante este proceso, fue fundamental el uso de librerías especializadas que, aunque inicialmente desconocidas para mí, fueron aprendidas y dominadas, como TinyGPS para el procesamiento de datos del módulo GPS y Wire para la comunicación I2C con ciertos sensores.

Este aprendizaje incrementó significativamente mi familiaridad y confianza con el ecosistema de desarrollo hardware-software.

Antes de avanzar, se realizaron pruebas para verificar la integración del dispositivo dentro del sistema completo. Esto implicó cargar el firmware en el microcontrolador y comprobar que los datos capturados se enviaban correctamente al servidor y se almacenaban en la base de datos sin errores.

Por ultimo dio lugar el desarrollo de la aplicación móvil haciendo uso de Android Studio y siguiendo el procedimiento descrito en el apartado anterior.

11. Pruebas del sistema

Para asegurar el correcto funcionamiento del sistema WAIT, se ha llevado a cabo una validación y por separado de cada uno de los módulos que lo componen. Esto implica, en primer lugar, un conocimiento detallado de las características técnicas de cada componente, tanto a nivel hardware como software, así como de su integración dentro del sistema IoT completo.

La metodología de prueba se ha basado en la definición de objetivos concretos para cada módulo, permitiendo comprobar si responden adecuadamente en distintos entornos y condiciones de uso. A partir de estos objetivos, se diseñaron y ejecutaron diferentes pruebas funcionales

A continuación, veremos un desglose de las distintas pruebas llevadas a cabo en las que encontraremos:

- Pruebas hardware
- Pruebas servidor Backend
- Pruebas aplicación

11.1. Pruebas hardware

Aquí podremos ver las distintas pruebas y los resultados realizados sobre los distintos dispositivos hardware que componen el sistema.

11.1.1. Módulo GPS (NEO-6M)

Para el **módulo GPS NEO-6M**, se realizaron diversas pruebas funcionales con el objetivo de verificar su capacidad de localización y la precisión de los datos proporcionados. Este módulo es fundamental dentro del sistema WAIT, ya que es el encargado de obtener las coordenadas geográficas (latitud y longitud) que permiten rastrear la posición del dispositivo en tiempo real.

Las pruebas se llevaron a cabo en diferentes entornos, tanto en espacios abiertos como en interiores, para comprobar su comportamiento bajo distintas condiciones de recepción satelital. En entornos exteriores, con buena visibilidad del cielo, el dispositivo logró fijar la señal de manera eficiente, ofreciendo datos con una precisión adecuada, generalmente inferior a los 5 metros. Sin embargo, en espacios cerrados o con estructuras que obstaculizaban la señal (como edificios, techos), el módulo presentó dificultades para obtener una señal GPS válida. En estos casos, se observaron retrasos en la adquisición de los datos o incluso pérdidas temporales de la conexión.

```
gps.f_get_position(&latitude, &longitude);  
Serial.print("Latitud/Longitud: ");  
Serial.print(latitude,5);  
Serial.print(", ");  
Serial.println(longitude,5);
```

Figura 20: Código para mostrar los datos del GPS por el puerto serie

```
Latitud/Longitud: 37.32173, -5.96745  
Fecha: 19/6/2025 Hora: 15:5:22.0  
Altitud (metros): 30.50  
Rumbo (grados): 190625.00  
Velocidad(kmph): 0.83  
Satelites: 5
```

Figura 21: Datos obtenidos por el GPS

11.1.2. Sensor Acelerómetro (MPU6050)

Para verificar el correcto funcionamiento del sensor acelerómetro integrado en el sistema WAIT, se realizaron diversas pruebas, entre ellas, el comportamiento del sensor en estado de reposo. Se observaron los valores registrados en los ejes X, Y y Z sin movimiento y provocando movimientos bruscos, vibraciones y cambios de orientación del dispositivo.

Durante las pruebas, se comenzó evaluando el comportamiento del sensor en estado de reposo. Se observaron los valores registrados en los ejes X, Y y Z sin movimiento, lo cual confirmaba una lectura inicial correcta y una calibración adecuada. Posteriormente, se realizaron ensayos provocando movimientos bruscos, vibraciones y cambios de orientación del dispositivo. El sensor respondió con rapidez y precisión.

```
void readRawMPU()
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(ACCEL_XOUT);
  Wire.endTransmission(false);
  Wire.requestFrom(MPU_ADDR, 14);

  AcX = Wire.read() << 8 | Wire.read();
  AcY = Wire.read() << 8 | Wire.read();
  AcZ = Wire.read() << 8 | Wire.read();

  Tmp = Wire.read() << 8 | Wire.read();

  GyX = Wire.read() << 8 | Wire.read();
  GyY = Wire.read() << 8 | Wire.read();
  GyZ = Wire.read() << 8 | Wire.read();

  // Conversión a unidades físicas
  ax = AcX / 16384.0; // Aceleración en g
  ay = AcY / 16384.0;
  az = AcZ / 16384.0;

  gx = GyX / 131.0; // Velocidad angular en °/s
  gy = GyY / 131.0;
  gz = GyZ / 131.0;
}
```

Figura 22: Código para obtener datos del sensor acelerómetro

```
AcX = 0.03g | AcY = 0.01g | AcZ = 1.06g | Tmp = 45.38°C | GyX = 6.15°/s | GyY = 0.80°/s | GyZ = -1.31
Test POST with actuator state
HTTP Response code: 201
Estado Actuador añadido correctamente
{"idSensorGps":71,"fechaHora":"","valueLong":0,"valueLat":0,"removed":false}
```

Figura 23: Datos obtenidos por el sensor acelerómetro(En reposo)

```
Sensor añadido correctamente
AcX = -0.17g | AcY = -0.06g | AcZ = 1.05g | Tmp = 45.85°C | GyX = 11.60°/s | GyY = -4.69°/s | GyZ = 5.39
Test POST with actuator state
HTTP Response code: 201
```

Figura 24: Datos obtenidos por el sensor acelerómetro(Sensor girado)

11.1.3. Modulo de alarma

Conectamos el módulo de alarma al GPIO 5 y sonó cuando superó los valores establecidos en el firmware. De todos los componentes hardware este es el que requiere menos complicación.

11.1.4. ESP32

Se comprobó la integración de los distintos sensores con el microcontrolador. Pudimos leer los valores obtenidos por el puerto serie.

```
#include <HTTPClient.h>
#include "ArduinoJson.h"
#include <NTPClient.h>
#include <WiFiUdp.h>
#include <PubSubClient.h>
#include <Adafruit_Sensor.h>
#include <pwmWrite.h>
#include <TinyGPS.h>
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>
#include <Wire.h>
```

Figura 25: Librerías usadas en el firmware

```

HardwareSerial gpsSerial(1);
int idSensor;
float actuador;
char fechaHora[20];
int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ;
float ax, ay, az, gx, gy, gz;
float mySTATUS;
bool STATUSBINARY;
int year;
byte month, day, hour, minute, second, hundredths;
unsigned long chars;
unsigned short sentences, failed_checksum;

const int DEVICE_ID = 71;
const int SENSOR_GPS_ID = 71;
const int SENSOR_AC_ID = 71;
const int ACTUADOR_ID = 71;

```

Figura 26: Inicialización de variables / Asignación de ID a componentes Hardware

11.2. Pruebas servidor Backend

Aquí podremos ver las distintas pruebas y los resultados realizados sobre los distintos servidor Backend.

11.2.1. Pruebas recepción de datos en la Raspberry Pi

Se comprobó la correcta transmisión y recepción de los datos desde el dispositivo al servidor. Para ello se realizaron múltiples pruebas centradas en la recepción, tratamiento y envío de datos entre los distintos elementos del sistema.

En primer lugar, se verificó la **transmisión y recepción de datos** desde el microcontrolador (ESP32) hacia el servidor mediante solicitudes HTTP. Se observó que los datos llegaban correctamente al servidor, eran procesados sin errores y almacenados en la base de datos.

También se realizaron pruebas en redes distintas (local y externa) para evaluar la accesibilidad del backend desde cualquier ubicación, teniendo en cuenta la configuración dinámica de direcciones IP. En estos escenarios, el servidor respondió adecuadamente.

```
josleoand@raspberrypi:~ $ cd /home/josleoand/Desktop/TFG
josleoand@raspberrypi:~/Desktop/TFG $ java -jar mysql-0.0.1-SNAPSHOT.jar run es.
us.dad.mysql.rest.RestAPIVerticle
Jun 19, 2025 4:50:42 PM io.vertx.core.impl.BlockedThreadChecker
WARNING: Thread Thread[vert.x-eventloop-thread-0,5,main]=Thread[vert.x-eventloop
-thread-0,5,main] has been blocked for 2009 ms, time limit is 2000 ms
API Rest is listening on port 8080
Jun 19, 2025 4:50:43 PM io.vertx.core.impl.launcher.commands.VertxIsolatedDeploy
er
INFO: Succeeded in deploying verticle
```

Figura 27: Lanzamiento de aplicación en servidor

```
MariaDB [wait]> select * from sensorsgpsstates;
```

idsensorsGpsStates	idsensorsGps	fechaHora	valueLong	valueLat	removed
1372	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1373	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1374	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1375	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1376	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1377	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1378	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1379	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1380	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1381	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1382	71	2025-06-19 00:00:00	37.3217	-5.96743	0
1383	71	2025-06-19 00:00:00	37.3217	-5.96743	0
1384	71	2025-06-19 00:00:00	37.3217	-5.96743	0
1385	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1386	71	2025-06-19 00:00:00	37.3217	-5.96742	0
1387	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1388	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1389	71	2025-06-19 00:00:00	37.3217	-5.96741	0
1390	71	2025-06-19 00:00:00	37.3217	-5.9674	0
1391	71	2025-06-19 00:00:00	37.3217	-5.9674	0
1392	71	2025-06-19 00:00:00	37.3217	-5.9674	0
1393	71	2025-06-19 00:00:00	37.3217	-5.9674	0
1394	71	2025-06-19 00:00:00	37.3217	-5.9674	0

Figura 28: Datos obtenidos por el sensor GPS(Vista Servidor)


```

|          1460 |          71 |          0 |          0 |          0 |
|          1461 |          71 |          0 |          0 |          0 |
|          1462 |          71 |          0 |          19 |          0 |
|          1463 |          71 |          0 |          9 |          0 |
|          1464 |          71 |          0 |          0 |          0 |
|          1465 |          71 |          0 |          1 |          0 |
|          1466 |          71 |          0 |         -11 |          0 |
|          1467 |          71 |          0 |          2 |          0 |
|          1468 |          71 |          0 |         -4 |          0 |
|          1469 |          71 |          0 |          0 |          0 |
+-----+-----+-----+-----+-----+
37 rows in set (0.002 sec)

MariaDB [wait]>

```

Figura 29: Datos obtenidos por el sensor Acelerómetro(Vista Servidor)

```

|          7 | josejr@icloud.com | jose | $2a$10$BS4DSHXFfqPHBxBTwir8U..Bzfs
QjkRovRe8AJVibwfbLR9X7uxy. | 661672298 |
+-----+-----+-----+-----+-----+
-----+-----+

```

Figura 30: Usuario añadido con el endpoint Register(Vista Servidor)

```

MariaDB [wait]> select * from devices_users;
+----+-----+-----+-----+-----+-----+
| id | user_id | idDevice | permission_level | nickname |
+----+-----+-----+-----+-----+-----+
| 9 | 6 | 394 | 1 | WAIT DE JOSE |
+----+-----+-----+-----+-----+-----+
1 row in set (0.001 sec)

```

Figura 31: Dispositivos asociados a usuarios(Vista Servidor)

```

Token received: NjpKb3NlOjE3NTAzNTE4MjAwMzc6ZDI0OWM4ZDIyNjYlMzM0Yg==
Token decoded: 6:Jose:1750351820037:d249c8d22665334b
☑ Token parsed - User ID: 6, Username: Jose
Token age: 3 minutes
Device ID requested: 394
☑ Database connection successful
Permission query: SELECT COUNT(*) as count FROM devices_users WHERE user_id = ?
AND idDevice = ?
Permission check result: 1
☑ User has permission for device
GPS query: SELECT gps_states.idsensorsGpsStates, gps_states.idsensorsGps, gps_st
ates.fechaHora, gps_states.valueLong, gps_states.valueLat, gps_states.removed FR
OM wait.sensorsgpsstates gps_states INNER JOIN wait.sensorsgps gps_sensor ON gps
_states.idsensorsGps = gps_sensor.idSensorsGps WHERE gps_sensor.idDevice = ? AND
gps_states.removed = 0 ORDER BY gps_states.idsensorsGpsStates DESC LIMIT 1 with
deviceId=394
☑ GPS query successful. Results found: 1
☑ Returning GPS data: 1 records

```

Figura 32: Logs de aplicación(Vista Servidor)


11.3. Pruebas aplicación

Uno de los objetivos principales de las pruebas fue comprobar la **recepción de los datos desde el servidor**, asegurando que la aplicación pudiera establecer conexión con el backend y visualizar correctamente la información enviada por los dispositivos. Para ello, se realizaron pruebas con distintos tipos de datos —como coordenadas GPS, valores del acelerómetro o notificaciones de eventos— verificando su actualización en tiempo real dentro de la interfaz de usuario. Además, se evaluó el comportamiento general de las **interfaces gráficas (Activities)** para asegurar que la navegación entre pantallas fuese fluida, sin cierres inesperados ni errores de carga.

Crear Cuenta

¿Ya tienes cuenta? Inicia sesión

Figura 33: Interfaz Register



Iniciar Sesión

Correo electrónico

Contraseña

INICIAR SESIÓN

¿No tienes cuenta? Regístrate

Figura 34: Interfaz Login






WAIT DE JOSE



Home



Location



Notifications

Figura 36: Interfaz página de inicio

Añadir Nuevo Dispositivo

1232324

WAIT DE JOSE

Apodo (opcional)

CANCELAR AÑADIR

Figura 35: Interfaz añadir nuevo dispositivo

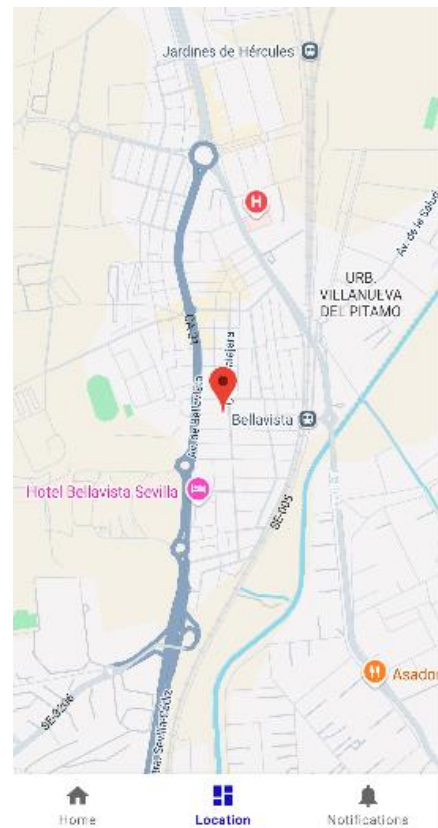


Figura 37: Interfaz visualización de mapa

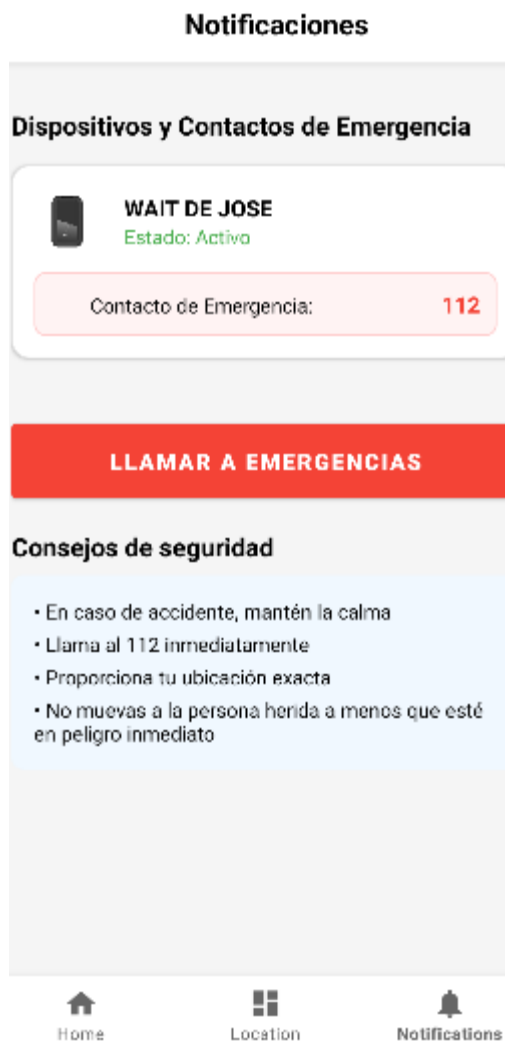


Figura 38: Interfaz notificaciones

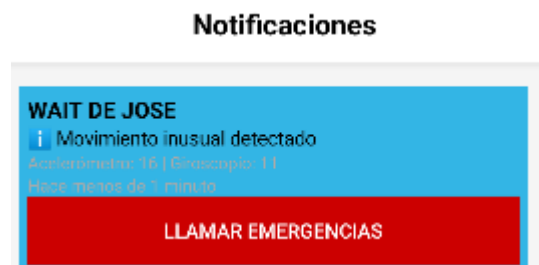


Figura 39: Notificación cuando se detecta un movimiento

12. Planificación temporal

El punto de partida del proyecto y uno de los mas importante fue realizar una planificación temporal donde, marcando objetivos y metas, organizar el trabajo a realizar para el desarrollo completo del proyecto.

En el caso de WAIT dividimos el proyecto en 4 tareas principales cada tarea pensada para desarrollarse en un máximo de 5 semanas, pudiendo ser este tiempo flexible.

Fase 1	Fase 2	Fase 3	Fase 4
Investigación y Aprendizaje	Desarrollo Hardware	Desarrollo Software	Pruebas y corrección de errores

12.1. Investigación y Aprendizaje

Durante las primeras cuatro semanas del proyecto se desarrollará una fase inicial centrada en la investigación y el aprendizaje profundo de todos los componentes que formarán parte del sistema. Esta etapa es fundamental para asegurar que la elección del hardware sea adecuada a los requerimientos del proyecto.

En este periodo se evaluarán distintos sensores disponibles en el mercado para cada tipo de medición necesaria, comparando su rendimiento, precisión y compatibilidad. Se analizarán en detalle sus respectivas hojas de datos (datasheets) para comprender su comportamiento en diferentes condiciones ambientales y operativas.

12.2. Desarrollo Hardware

La fase de desarrollo hardware representa una de las partes más importantes del proyecto, ya que en ella se lleva a cabo la integración física de todos los componentes que forman parte del sistema. Esta etapa implica no solo la conexión adecuada de sensores, actuadores y microcontroladores, sino también la comprensión técnica del funcionamiento de cada uno de ellos.

Una vez se ha realizado el estudio teórico, se procede al **diseño del esquema de conexiones**, identificando los pines adecuados en el microcontrolador para cada sensor o actuador. Se consideran también aspectos prácticos como el orden de conexión, la necesidad de usar módulos

adicionales (como convertidores de nivel o expansores de pines), y el tipo de comunicación (I2C, UART, digital, analógica).

Posteriormente, se realiza el **montaje del prototipo sobre una protoboard**, lo cual permite verificar las conexiones antes de realizar una instalación más permanente.

12.3. Desarrollo Software

Otras de las partes principales fue el desarrollo software, ya que implica la creación de los distintos componentes que permiten la comunicación, el procesamiento de datos y la interacción del sistema con el usuario final. Dada la naturaleza distribuida del sistema, este apartado se ha dividido en varias capas que han sido desarrolladas utilizando diferentes tecnologías especializadas, cada una cumpliendo un rol específico.

12.3.1. Servidor backend con Vert.x y Maven en Eclipse

La lógica principal del servidor fue desarrollada usando Vert.x, fue elegido por su capacidad para gestionar múltiples solicitudes concurrentes sin comprometer el rendimiento, algo clave en un sistema donde dispositivos y aplicaciones móviles realizan peticiones frecuentes.

El proyecto se estructuró como un proyecto **Maven**, lo que facilitó la gestión de dependencias, la modularidad del código y la integración con el entorno de desarrollo **Eclipse**. Se implementaron múltiples **endpoints HTTP** para exponer una **API RESTful**, utilizando el módulo **vertx-web**, permitiendo así la interacción con dispositivos y la app Android mediante métodos **GET**, **POST** y **PUT**.

12.3.2. Base de datos con MariaDB

Como sistema de almacenamiento de datos, se utilizó **MariaDB**, una base de datos relacional de código abierto compatible con MySQL. Esta base de datos se desplegó inicialmente en el entorno de desarrollo local y posteriormente desplegado en la **Raspberry Pi**, usando **MariaDB**, que actúa como servidor.

El diseño de la base de datos evolucionó desde una estructura básica de prueba hasta un modelo más robusto con relaciones normalizadas entre usuarios, dispositivos, sensores, lecturas y eventos

de alarma. Esto permitió mejorar la escalabilidad y el control de integridad de los datos, además de facilitar las consultas optimizadas necesarias para la app y los dispositivos.

12.3.3. Firmware con PlatformIO

Para el desarrollo del **firmware del dispositivo**, se utilizó el entorno **PlatformIO** sobre **VS Code**, con el microcontrolador **ESP32** como núcleo del sistema embebido. El firmware fue responsable de gestionar la lectura de sensores, la activación de actuadores (como la alarma sonora), y el envío y recepción de datos mediante **HTTP**.

Se utilizaron librerías específicas como TinyGPS++ para el procesamiento de datos del módulo GPS, y se establecieron rutinas para manejar tanto la comunicación con el backend como el control local.

12.3.4. Aplicación móvil con Android Studio

La interfaz directa con el usuario fue desarrollada mediante una aplicación móvil nativa utilizando **Android Studio**, programada principalmente en **Java** y con las interfaces gráficas diseñadas en **XML**. La app permite al usuario consultar la ubicación actual de sus dispositivos en un mapa, además de visualizar el estado de sensores y autenticarse con su cuenta mediante llamadas a la API, entre otras opciones.

La app se comunica directamente con el backend a través de **consultas API REST** autenticadas, consumiendo los datos que están almacenados en la base de datos gestionada por la Raspberry Pi.

12.4. Pruebas y corrección de errores

La etapa de pruebas y depuración fue esencial para garantizar el correcto funcionamiento del sistema completo, asegurando que la integración entre sus distintas partes —dispositivo embebido, servidor, base de datos y aplicación móvil— fuese fluida, confiable y robusta. Este proceso se abordó de forma progresiva dividiéndolo en pruebas unitarias, pruebas de integración y pruebas funcionales.

Durante el desarrollo del firmware con PlatformIO, se realizaron **pruebas unitarias locales** en cada componente de hardware conectado (sensores, actuadores y módulos de comunicación). Los errores más comunes en esta etapa incluyeron problemas con las librerías (como incompatibilidades con versiones de PlatformIO), fallos en la comunicación serie, y lecturas erróneas debido a conexiones inestables.

Una vez desplegada la API desarrollada con Vert.x, se empleó la herramienta **Postman** para realizar pruebas de los distintos endpoints. Estas pruebas permitieron simular múltiples escenarios de uso sin necesidad de la aplicación móvil o el dispositivo físico. Se detectaron errores de validación, errores de autenticación mal gestionados y problemas de asincronía en algunas rutas, que fueron solucionados ajustando el flujo de lógica en los verticles y añadiendo control de errores centralizado. En paralelo, se realizaron pruebas sobre la base de datos MariaDB tanto en entorno local como sobre la Raspberry Pi.

En Android Studio, se ejecutaron pruebas funcionales en dispositivos virtuales (emuladores) y reales. Durante estas pruebas se detectaron varios problemas como errores de parsing JSON, fallos de diseño en pantalla para resoluciones pequeñas, y errores de sincronización entre la lógica de la app y el backend. Se solucionaron ajustando el layout XML, refactorizando funciones y mejorando el manejo de excepciones.

Una vez todos los componentes estuvieron en su fase estable, se realizaron pruebas de integración completas.

12.5. Visualización temporal

Aquí podemos ver un desglose de la planificación temporal en forma de diagrama de Gantt,

Identificador	Numero de horas
Fase 1: Investigación y Aprendizaje	120
Fase 2: Desarrollo Hardware	210
Fase 3: Desarrollo Software	245
Fase 4: Pruebas y corrección de errores	65

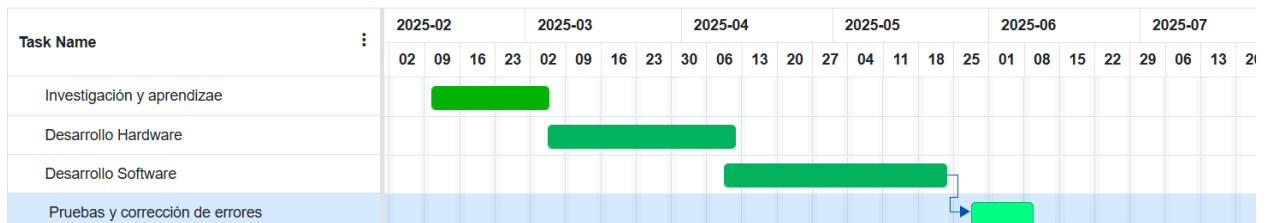


Figura 40: Diagrama de Gantt

13. Costes

El desarrollo del sistema completo ha sido realizado por 4 personas, que abarcaron todas las partes del mismo. Se llegó a un acuerdo con una empresa de hardware, donde pidiendo grandes cantidades de componentes, se redujera el precio unitario.

13.1. Materiales

Desglose de los materiales utilizados y el precio asociado a ellos.

Identificador	Uds	Precio(Unidad)	Total
ESP32	1500	1,80€	2.700,00€
Sensor GPS NEO-6M	1500	1,00€	1.500,00€
Sensor Acelerómetro	1500	1,00€	1.500,00€
Módulo GSM	1500	2,50€	3.750,00€
Módulo Alarma	1500	0,50€	750,00€
Alimentación	1500	1,00€	1.500,00€
Carcasa	1500	1,30€	1.950,00€
TOTAL			13.650,00€

13.2. Mano de Obra

Gracias al profesional equipo de trabajo que se contrato, se consiguió reducir en 4 semanas el tiempo total del proyecto, a continuación quedan detallados el coste humano que llevo relacionado el proyecto

Identificador	Coste/Hora	Horas	Total
Desarrollador Hardware	17,00€	225	3.820,00€
Desarrollador Software	17,00€	280	4.760,00€
Personal de Pruebas	6,00€	135	810,00€
TOTAL			9.390,00€

13.2.1. Estimación de Cocomo

El Modelo Constructivo de Costes COCOMO -Constructive Cost Model – es utilizado en proyectos de software para estimar los costes del mismo en función de tres submodelos: básico, intermedio y detallado. Aquí esta un análisis aplicando el modelo para el proyecto WAIT.

1. Fórmula para calcular el esfuerzo (E) en persona-meses:

- $E = a \times (KLOC)^b$

Donde: $a=2.4$, $b=1.05$ y $KLOC=1.4$

- $E = 2.4 \times (1.4)^{1.05} = 3.41$ persona-meses

2. Tiempo de desarrollo (TDEV) en meses:

- $TDEV = c \times (E)^d$

Donde: $c=2.5$, $d=0.38$.

- $TDEV = 2.5 \times (3.41)^{0.38} \approx 3.98$ meses

13.3. Coste del sistema

La unificación del coste de materiales y personal da como resultado los siguientes costes:

Identificador	Total
Materiales	13.650,00€
Mano de Obra	7.060,00€
TOTAL	23.040,00€

Coste de producción por 1500 unidades: 23.040,00€.

Teniendo en cuenta que el precio de venta al usuario final será de 40€, podemos concluir que para 1500 unidades podemos conseguir un beneficio del 260%.

14. Conclusiones

A lo largo del desarrollo del proyecto **WAIT**, se ha logrado cumplir con los principales objetivos propuestos en la fase inicial. Esta solución, concebida como un sistema integral de rastreo y monitoreo, se apoya en una arquitectura distribuida fundamentada en tecnologías del Internet de las Cosas (IoT), permitiendo integrar de manera efectiva todas las capas funcionales del sistema: desde la captación de datos en tiempo real hasta su procesamiento, almacenamiento, visualización y control a través de una aplicación móvil.

El núcleo del sistema está constituido por un dispositivo físico basado en el microcontrolador **ESP32**, que fue capaz de capturar información proveniente de múltiples sensores integrados. Entre ellos, se destaca la adquisición de datos desde el sensor de aceleración (acelerómetro) y el módulo GPS, fundamentales para las funcionalidades de rastreo y detección de movimiento. La selección de estos componentes no fue arbitraria: se realizó un análisis de sus **datasheets** para comprender su comportamiento en distintos contextos operativos. Esto permitió llevar a cabo un buen ensamblaje. Posteriormente, se desarrolló un firmware sólido, capaz de gestionar de forma eficiente la interacción entre sensores, comunicaciones y lógica interna, cumpliendo con los requerimientos establecidos desde la fase de diseño.

En paralelo, se abordó uno de los pilares del sistema: la lógica de comunicación entre dispositivos, especialmente entre el hardware, el servidor y la aplicación móvil. Inicialmente, se partió de una estructura sencilla inspirada en proyectos anteriores que utilizaban endpoints básicos para el intercambio de información. Sin embargo, con el tiempo, se logró extender esta lógica hacia una arquitectura mucho más robusta y escalable. Se diseñaron endpoints RESTful más sofisticados, gestionados mediante el framework **Vert.x**, que permitieron implementar operaciones seguras y autenticadas. Estos endpoints se integraron con una base de datos **MariaDB**, cuya estructura evolucionó para adaptarse al crecimiento funcional del sistema, mediante la creación de nuevas tablas que permiten asociar datos directamente con usuarios, dispositivos, sensores y configuraciones específicas.

El desarrollo de la aplicación móvil constituye otro de los hitos importantes del proyecto. Considerando el avance constante del smartphone en la sociedad actual y la necesidad de ofrecer una interfaz accesible y portátil, se diseñó una aplicación Android completamente funcional. Esta app permite a los usuarios interactuar con el sistema de forma intuitiva, accediendo a la información recopilada por el dispositivo en tiempo real, gestionando configuraciones y visualizando localizaciones geográficas. Su desarrollo se realizó en **Android Studio**, utilizando **Java** y **XML**, con un enfoque en la experiencia de usuario y la comunicación fluida con el backend mediante consultas API.

Otro aspecto técnico que fue necesario dominar a lo largo del proceso fue la **configuración de puertos**, el control de la red y la gestión de **direcciones IP dinámicas**, que resultaron fundamentales para garantizar la disponibilidad del servidor desde redes externas. Esta parte implicó un aprendizaje valioso sobre aspectos de conectividad y seguridad de redes que son esenciales en cualquier sistema IoT real.

Desde una perspectiva personal y profesional, el proyecto **WAIT** ha representado una experiencia formativa integral. Ha servido como base sólida para comprender el funcionamiento de sistemas distribuidos que combinan hardware, servidores backend y aplicaciones móviles. La cantidad de conocimientos adquiridos a lo largo del desarrollo —en temas de electrónica, programación, bases de datos, redes y diseño de interfaces— me proporciona ahora una visión mucho más clara de los retos reales que implica construir una solución tecnológica de este tipo. Sin duda, este proyecto sienta las bases necesarias para afrontar con éxito desarrollos más complejos que utilicen arquitecturas similares en el ámbito del Internet de las Cosas y más allá.

15. Trabajo a futuro

Aunque el sistema desarrollado en el proyecto WAIT ha logrado alcanzar con éxito los objetivos definidos en la fase inicial, durante su desarrollo se han identificado diversas áreas con potencial de mejora que podrían abordarse en futuras iteraciones del proyecto. Estas mejoras no solo permitirían perfeccionar la solución actual, sino también ampliar su alcance, aumentar su escalabilidad y optimizar su rendimiento general.

Una de las limitaciones más evidentes de la versión actual del sistema es que la aplicación móvil ha sido desarrollada exclusivamente para dispositivos Android. Aunque funcional y estable, esto restringe su adopción a usuarios de dicho sistema operativo. Una mejora clave sería el desarrollo de una versión compatible con dispositivos iOS, permitiendo que usuarios con iPhone también puedan utilizar el sistema sin limitaciones.

Por otro lado, actualmente el dispositivo central del Sistema tiene un tamaño compacto y fácil de manejar pero, realizando una reelección de componentes hardware para que el sistema tuviera unas dimensiones aún mas pequeñas, se podría mejorar.

Por último, actualmente el backend del sistema se ejecuta en una **Raspberry Pi local**, lo cual ha permitido un desarrollo y una validación efectiva durante la fase inicial. Sin embargo, este enfoque presenta limitaciones en cuanto a disponibilidad, fiabilidad y escalabilidad. Las caídas de red, interrupciones eléctricas o el desgaste del hardware son factores que afectan negativamente la estabilidad del sistema.

Por ello, una mejora sustancial sería **migrar el servidor y la base de datos a una infraestructura en la nube**, eliminando la dependencia física de la Raspberry Pi.

16. Bibliografía

- [ESP32 Series Datasheet](#)
- [NEO-6 GPS Module Datasheet](#)
- [MPU-6000/6050 Datasheet](#)
- [SIM800L GSM/GPRS Module Datasheet](#)
- [Invoxia GPS Tracker](#)
- [TK905 GPS Tracker](#)
- [Samsung SmartThings Tracker](#)
- [Google Maps API](#)
- [Configuración Raspberry PI](#)
- [Uso de PlatformIO](#)
- [Uso de Vert.x](#)
- [Estimación de Cocomo](#)
- [Diagrama de Gantt](#)

