**UNIVERSIDAD DE CASTILLA-LA MANCHA**
**ESCUELA SUPERIOR DE INFORMÁTICA**

**GRADO EN INGENIERÍA INFORMÁTICA**

**TECNOLOGÍA ESPECÍFICA DE**
**INGENIERÍA DE COMPUTADORES**

**TRABAJO FIN DE GRADO**

# GVIDI: Monitorización inteligente de grupos de expedición para la mejora de seguridad

**Iván García Herrera**

Julio, 2019

# GVIDI: Monitorización inteligente de grupos de expedición para la mejora de seguridad

**UNIVERSIDAD DE CASTILLA-LA MANCHA**

**ESCUELA SUPERIOR DE INFORMÁTICA**

**Tecnologías y Sistemas de Información**

**TECNOLOGÍA ESPECÍFICA DE**
**INGENIERÍA DE COMPUTADORES**

**TRABAJO FIN DE GRADO**

# GVIDI: Monitorización inteligente de grupos de expedición para la mejora de seguridad

Autor: Iván García Herrera

Director: Dr. Javier Alonso Albusac Jiménez

Julio, 2019

**Iván García Herrera**

Ciudad Real – Spain

*E-mail:* Ivan.Garcia16@alu.uclm.es
*Teléfono:* 628 669 551
*Web site:* `http://ivangarrera.herokuapp.com`

## TRIBUNAL:

**Presidente:**

**Vocal:**

**Secretario:**

## FECHA DE DEFENSA:

## CALIFICACIÓN:

| PRESIDENTE | VOCAL | SECRETARIO |
|---|---|---|
| Fdo.: | Fdo.: | Fdo.: |

# Resumen

En los últimos años la inclusión de las nuevas tecnologías en el senderismo y otros deportes de montaña ha hecho que estos sufran una profesionalización. En el senderismo los usuarios se ponen metas más peligrosas conforme aumenta su experiencia en la montaña, lo que hace que se incremente la posibilidad de sufrir accidentes. Además, los guías de senderismo no disponen de herramientas suficientes en el mercado actual para realizar un buen análisis de la situación del grupo de expedición.

El presente Trabajo de Fin de Grado (TFG) surge como una solución a esta problemática y tiene la intención de cubrir el vacío que existe en el mercado actual de herramientas para realizar un análisis grupal en expediciones en la montaña. En este proyecto se plantea la construcción de un sistema formado por un dispositivo físico, una aplicación móvil y una plataforma *web* que permita monitorizar la situación del grupo de expedición con el objetivo principal de maximizar la seguridad del mismo. El dispositivo físico está equipado con una serie de sensores que captan información acerca del entorno del usuario y la envían a la aplicación móvil para que realice un procesamiento de dicha información. La plataforma *web* proporciona una vista detallada de las expediciones realizadas y en curso.

En este proyecto, los usuarios de la expedición podrán visualizar tanto en tiempo real como de forma histórica información relevante acerca de la expedición que están realizando. Los guías además, tendrán una vista global de la expedición (visualizando el estado de todos los participantes de la expedición) y de las alertas que en esta se generan con el objetivo de actuar lo más rápidamente posible para evitar situaciones de riesgo.

# Abstract

In the last few years the inclusion of new technologies in hiking and other mountain sports has made them suffer a professionalization. In hiking, users set more dangerous goals as their experience in mountaineering increases, which increases the likelihood of accidents. In addition, in today's market there are no tools for hiking guides to perform a good analysis of the situation of the expedition group.

This TFG comes as a solution to this problem and is aimed at filling the gap in tools for group analysis in mountain expeditions that exists in today's market. This project involves the construction of a system consisting of a physical device, a mobile application and a web platform to monitor the situation of the expedition group with the main objective of maximizing its safety. The physical device is equipped with a number of sensors which capture information about the user's environment and send it to the mobile application for processing this information. The web platform provides a detailed view of finished and ongoing expeditions.

In this project, the users of the expedition will be able to visualize in real time as well as in a historical way relevant information about the expedition they are currently carrying out. The guides will also have a global view of the expedition (visualizing the status of all participants in the expedition) and of the alerts that are generated in the expedition in order to act as quickly as possible to avoid risk situations.

# Agradecimientos

En primer lugar a mis padres Rafa y Oliva, por darme la oportunidad de labrar mi futuro en la profesión de mis sueños y confiar en mí y en mis posibilidades. A mis hermanos Diego y Óscar, por su sonrisa en los momentos más difíciles.

A Lidia, por comenzar y finalizar la etapa universitaria juntos y por ser mi gran apoyo a lo largo de la misma.

A mis compañeros, por hacer las clases más amenas y por darme un buen motivo para asistir a ellas.

Por último y no menos importante, a todo el profesorado de la Escuela Superior de Informática, por transmitirme aún más su pasión por la informática. En especial a mi director de proyecto, Javier Alonso Albusac Jiménez, por su implicación a lo largo de este trabajo, su disponibilidad total para la resolución de dudas tanto presencialmente como por videoconferencia y por su disposición para revisar el proyecto los fines de semana y festivos.

Iván.

*A mi abuelo Mateo*

# Índice general

**Resumen** V

**Abstract** VII

**Agradecimientos** IX

**Índice general** XIII

**Índice de cuadros** XVII

**Índice de figuras** XIX

**Índice de listados** XXIII

**Listado de acrónimos** XXV

**1. Motivación del trabajo** 1

1.1. Trabajos previos . . . . . 4

1.1.1. Aplicaciones existentes para rutas de expedición . . . . . 4

1.1.2. Dispositivos existentes para rutas de expedición . . . . . 5

1.2. Estructura del documento . . . . . 12

**2. Objetivos** 13

2.1. Objetivo general . . . . . 13

2.2. Objetivos específicos . . . . . 13

2.3. Objetivos docentes . . . . . 16

**3. Método de trabajo** 19

3.1. Arquitectura . . . . . 19

3.1.1. Visión general de la arquitectura . . . . . 19

3.2. Microcontrolador *Arduino* . . . . . 24

3.2.1. Diseño del dispositivo *hardware* . . . . . 24

3.2.2. Toma de datos . . . 27
3.2.3. Creación de la *Printed Circuit Board* (PCB) . . . 34
3.3. Aplicación móvil . . . 36
3.3.1. Diseño de la aplicación móvil . . . 36
3.3.2. Funcionalidad de la aplicación móvil . . . 37
3.3.3. Comunicación entre la aplicación móvil y el dispositivo físico . . . 38
3.3.4. Autenticación de usuarios en `Firebase` y almacenamiento de datos en `Firestore` . . . 43
3.4. Detección automática de situaciones anómalas . . . 47
3.4.1. Algoritmo de detección de caídas . . . 49
3.4.2. Grado de dispersión del grupo . . . 55
3.4.3. Análisis del riesgo de caída en la expedición de cada integrante de la misma . . . 56
3.5. Plataforma *web* . . . 60
3.5.1. Diseño de la plataforma *web* . . . 60
3.5.2. Implementación de la plataforma *web* . . . 63
3.6. Organización del proyecto en paquetes de trabajo . . . 71
3.6.1. Paquetes de trabajo (PT)1. Diseño y desarrollo de dispositivo *Hardware* (Semanas 1 a 33) . . . 73
3.6.2. PT2. Diseño y desarrollo de aplicación móvil (Semanas 9 a 24) . . . 74
3.6.3. PT3. Diseño y desarrollo de la plataforma *web* (Semanas 25 a 37) . 74
3.6.4. PT4. Análisis Inteligente(Semanas 9 a 34) . . . 75
3.7. Metodología de trabajo . . . 76
3.7.1. Iteraciones . . . 77
3.8. Características *hardware* y *software* del desarrollo . . . 83
3.8.1. Medios *hardware* . . . 83
3.8.2. Medios *software* . . . 84

**4. Resultados** **87**
4.1. Desarrollo de un dispositivo *hardware* equipado con distintos sensores. . . 87
4.2. Desarrollo de una aplicación móvil. . . . 87
4.3. Detección inteligente de situaciones anómalas en una expedición. . . . 87
4.3.1. Cumplimiento del objetivo de detección de caídas. . . . 88
4.3.2. Cumplimiento del objetivo de análisis del grado de dispersión en el grupo de expedición. . . . 90
4.3.3. Cumplimiento del objetivo de análisis del riesgo de caída. . . . 92
4.4. Desarrollo de una plataforma web . . . 95

4.5. Competencias específicas de intensificación trabajadas durante este proyecto 96

**5. Conclusiones** **99**

5.1. Trabajo futuro . . . . . 101

**A. Antecedentes** **105**

A.1. Microcontroladores . . . . . 105

A.1.1. Arduino . . . . . 106

A.1.2. Raspberry Pi . . . . . 107

A.1.3. Otros microcontroladores . . . . . 108

A.2. Sensores . . . . . 110

A.3. Sistemas Operativos para Dispositivos Móviles . . . . . 117

A.3.1. Android . . . . . 117

A.4. Bluetooth . . . . . 119

A.5. Desarrollo Web . . . . . 122

A.5.1. HTML5 + CSS3 . . . . . 123

A.5.2. JavaScript . . . . . 125

A.6. *Application Programming Interface* (API)s . . . . . 127

A.6.1. API de Agencia Estatal de Meteorología (AEMET) . . . . . 129

A.6.2. API de *Google Maps* . . . . . 130

**B. Diagrama de clases** **133**

**Bibliografía** **135**

# Índice de cuadros


1.1. Comparación entre dispositivos *Personal Locator Beacon* (PLB) y *Satellite Emergency Notification Devices* (SEND). . . . . 11

2.1. División de la funcionalidad del proyecto entre sus componentes. . . . . 16

3.1. Componentes que se han usado para construir el prototipo *hardware*. . . . . 22
3.2. Magnitudes y sensores que se usan en este proyecto. . . . . 25
3.3. Características principales del sensor `DHT22`. . . . . 28
3.4. Valores digitales que mide el sensor `MPU6050` para una sensibilidad de $\pm 2g$ en la aceleración y de $\pm 250\ deg/s$ en la velocidad angular. . . . . 29
3.5. Intervalos de voltaje que el sensor `YL-83` proporciona y su relación con la cantidad aproximada de lluvia. . . . . 32
3.6. Descripción de los los eventos que se pueden generar en el algoritmo de detección de caídas y los valores de aceleración que los generan. . . . . 52
3.7. Variables de entrada y conjuntos difusos de dichas variables, del algoritmo de riesgo de caída. . . . . 57
3.8. Variable de salida y conjuntos difusos de dicha variable, del algoritmo de riesgo de caída. . . . . 57
3.9. Descripción resumida de la primera iteración. . . . . 78
3.10. Descripción resumida de la segunda iteración. . . . . 79
3.11. Descripción resumida de la tercera iteración. . . . . 79
3.12. Descripción resumida de la cuarta iteración. . . . . 80
3.13. Descripción resumida de la quinta iteración. . . . . 80
3.14. Descripción resumida de la sexta iteración. . . . . 81
3.15. Descripción resumida de la séptima iteración. . . . . 82
3.16. Descripción resumida de la octava iteración. . . . . 82
3.17. Descripción resumida de la novena iteración. . . . . 83
3.18. Descripción resumida de la décima iteración. . . . . 83

4.1. Precisión del algoritmo de detección de caídas, ante distintas pruebas. . . . 90
4.2. Coordenadas en grados y en radianes de los usuarios de la expedición hipotética planteada. . . . . 91

4.3. Comparativa del grado de separación obtenido de diversas maneras. . . . . 93
4.4. Valores y conjuntos difusos de las variables de entrada del *Test* 1. . . . . 93
4.5. Valores y conjuntos difusos de las variables de entrada del *Test* 2. . . . . 94
4.6. Valores y conjuntos difusos de las variables de entrada del *Test* 3. . . . . 94

A.1. Comparativa de las principales características de los distintos microcontroladores . . . . . 110
A.2. Clasificación de los dispositivos Bluetooth según su potencia. . . . . 120
A.3. Resumen de las características más importantes añadidas a las distintas versiones de Bluetooth. . . . . 120

# Índice de figuras


1.1. Aplicación móvil *GPX Viewer*. [1] . . . . . 4
1.2. Aplicación móvil *AllTrails*. [2] . . . . . 4
1.3. Aplicación móvil Primeros Auxilios. [3] . . . . . 5
1.4. Aplicación móvil *Alpify*. . . . . 5
1.5. Dispositivo Appareil de Recherche de Victimes d'Avalanches (ARVA) para la búsqueda de montañistas en avalanchas. [4] . . . . . 7
1.6. Sonda de fibra de carbono. [5] . . . . . 8
1.7. Baliza PLB para la localización y rescate de accidentados. [6] . . . . . 9
1.8. Dispositivo de notificación por satélite. [7] . . . . . 10
1.9. Pareja de *walkies Personal Mobile Radio* (PMR). [8] . . . . . 11
1.10. Anuncio para el uso del Canal 7-7 en la montaña. [9] . . . . . 12

3.1. Esquema general de `GVIDI`. . . . . 21
3.2. Arquitectura general, en la que se pueden observar los distintos componentes y tareas que conforman el proyecto. . . . . 23
3.3. Conexiones de todos los componenetes en el microcontrolador `Arduino`. . . 26
3.4. Imagen del sensor `DHT22` [10] . . . . . 28
3.5. Imagen del sensor `MPU6050` [11] . . . . . 29
3.6. Componentes del acelerómetro y giroscopio `MPU6050`. . . . . 31
3.7. Imagen del sensor de lluvia `YL-83` [12] . . . . . 32
3.8. Imagen del sensor `NEO-6M` [13] . . . . . 32
3.9. Diseño de la PCB. . . . . 35
3.10. Vista en 3D de la PCB. . . . . 35
3.11. Impresión final de la PCB. . . . . 35
3.12. Prototipo final del microcontrolador conectado con todos los sensores que se usan. . . . . 36
3.13. Esquema general del patrón de arquitectura de *software Model-View-Controller* (MVC). . . . . 37
3.14. Pantallas de inicio de sesión, registro y reestablecimiento de contraseña. . . 39

3.15. Pantallas de unión a expedición en curso, creación de expedición y vista del estado general de la expedición. . . . . 39
3.16. Pantalla de vista en detalle de un participante y pantalla principal. . . . . 40
3.17. Arquitectura de la comunicación *Bluetooth* en el dispositivo móvil. . . . . 40
3.18. Problema en el uso de dos *sockets Bluetooth* y su solución. . . . . 43
3.19. Credenciales necesarias para poder acceder a la API de `Firebase` en la aplicación *Android*. . . . . 45
3.20. Diagrama de secuencia en llamadas a la base de datos en `Firestore`. . . . . 47
3.21. Formato de un mensaje de datos que se envía a través de *Bluetooth*. . . . . 48
3.22. Variación de la aceleración lineal cuando el individuo se encuentra en reposo total. La unidad de los datos de aceleración son $m/s^2$. . . . . 49
3.23. Fuerzas que actúan sobre el cuerpo en caída libre (de forma ideal). . . . . 51
3.24. Variación de la aceleración lineal durante una caída. La unidad de los datos de aceleración son $m/s^2$. . . . . 51
3.25. Diagrama de estados del algoritmo de detección de caídas. . . . . 53
3.26. Valores de los conjuntos difusos, de las variables de entrada de humedad y lluvia. . . . . 59
3.27. Valores de los conjuntos difusos, de la variable de entrada de luz y la variable de salida de riesgo. . . . . 59
3.28. Arquitectura de la plataforma *web*. . . . . 61
3.29. Página de *Login* de la *web*. . . . . 62
3.30. Página que contiene el histórico de rutas de un usuario. . . . . 62
3.31. Tarjetas con información general acerca de la ruta. . . . . 62
3.32. Mapa en tiempo real de la expedición con información detallada del punto en el que se haga *click*. . . . . 63
3.33. Esquema del *backend* de la plataforma *web*. . . . . 66
3.34. Habilitar `Firestore` desde el proyecto de `Firebase`. . . . . 67
3.35. Mapa durante el recorrido de la expedición, en el que se pueden consultar detalles y alertas de dicha expedición. . . . . 68
3.36. Distancia ortodrómica entre dos puntos sobre la superficie de una esfera. [14] 71
3.37. Diagrama de Gantt. . . . . 72
3.38. Tablero de `Trello` . . . . . 77
3.39. Repositorio `git` del proyecto, alojado en `GitHub`. . . . . 77

4.1. Datos de los sensores equipados en el microcontrolador `Arduino`. . . . . 88
4.2. Salida del algoritmo que calcula el grado de dispersión de la expedición, para una situación hipotética planteada. . . . . 91
4.3. Localización de los usuarios de la expedición hipotética planteada. . . . . 92

4.4. Salida del algoritmo que calcula el riesgo de caída en la expedición, para los *tests* aquí propuestos. . . . . 95
4.5. Páginas de *login* e histórico de rutas desde un dispositivo móvil `iPhone X` . . 96
4.6. Página de información específica acerca de una expedición, vista desde un dispositivo móvil. . . . . 97

A.1. Esquema de un microcontrolador. . . . . 105
A.5. Esquema de conexión del *Light-Dependent Resistor* (LDR) y Arduino. . . . 111
A.6. Esquema de conexión del sensor de lluvia `YL-83`. . . . . 112
A.7. Esquema de conexión del sensor de temperatura y humedad `DHT22`. . . . . 113
A.8. Esquema de conexión del sensor de pulso cardíaco `PulseSensor`. . . . . 115
A.9. Esquema de conexión del acelerómetro y giroscopio `MPU6050`. . . . . 116
A.11. Establecimiento de una conexión Bluetooth. . . . . 122
A.12. Componentes en React. . . . . 126
A.13. Modelo de ejecución conducido por eventos en NodeJs. . . . . 127
A.14. Esquema básico de funcionamiento de una API. . . . . 128
A.15. Ubicuidad en el uso de las APIs. . . . . 128
A.16. API *key* para el uso del servicio AEMET. . . . . 130

B.1. Diagrama de clases completo. . . . . 134

# Índice de listados


3.1. **Borrado de la memoria *Electrically Erasable Programmable Read-Only Memory* (EEPROM) cuando se deja pulsado un botón durante más de 5 segundos.** . . . . . 26

3.2. **Ejemplo de funcionalidad añadida a librería de código libre del sensor DHT22.** . . . . . 27

3.3. **Ejemplo de toma de datos del sensor `DHT22`.** . . . . . 28

3.4. **Cálculo de los *offsets* (en este ejemplo se han sustituido las constantes por números para un mejor entendimiento).** . . . . . 29

3.5. **Lectura de datos del módulo GPS usando una comunicación serial.** . . 32

3.6. **Ejemplo de valores en formato *National Marine Electronics Association* (NMEA).** . . . . . 33

3.7. **Búsqueda de dispositivos *Bluetooth* emparejados para obtener la dirección *Media Access Control* (MAC).** . . . . . 41

3.8. **Hilo secundario que lee datos del *socket Bluetooth* y los envía a un manejador.** . . . . . 41

3.9. **Inicio de sesión usando `Firabase Auth`.** . . . . . 44

3.10. **Formato de los datos almacenados en `Firestore` para una expedición.** . 45

3.11. **Obtención de todos los participantes de una determinada expedición.** . 46

3.12. **Obtener datos de localización usando el dispositivo móvil.** . . . . . 48

3.13. **Proceso de comprobación del formato de los datos recibidos por el *socket Bluetooth*.** . . . . . 48

3.14. **Creación de eventos a partir de los datos de aceleración y transición a otro estado.** . . . . . 52

3.15. **Exclusión mutua en la transición entre estados, en el algoritmo de detección de caídas.** . . . . . 54

3.16. **Máquina de estados que gestiona las transiciones entre estados, a partir de eventos generados.** . . . . . 54

3.17. **Algunas de las reglas difusas definidas para obtener el grado de pertenencia de una variable de salida a los conjuntos difusos.** . . . . . 60

3.18. **Creación del servidor `Express` de la plataforma *web*.** . . . . . 64

3.19. **Rutas que gestiona el servidor al manejar peticiones a la API de expediciones.** . . . . . 64

3.20. **Controlador de expediciones, que gestiona la lógica de la petición.** . . . 64
3.21. **Creación de la conexión con `Firebase` en la aplicación *web*.** . . . 66
3.22. **Consulta de los datos de todas las expediciones usando `Firestore` y `Javascript`.** . . . 67
3.23. **Creación del mapa en la plataforma *web*, incluyendo la ruta y los marcadores de alertas**. . . . 68
3.24. **Obtención de la localización más cercana a un punto dado**. . . . 70
3.25. **Obtención de la distancia entre dos puntos usando la formula de *Harvesine*.** 70
4.1. **Test de análisis del grado de dispersión en una situación hipotética planteada.** . . . 90
4.2. **Cálculo del riesgo de caída en una expedición, para unos valores de humedad, luz y lluvia dados.** . . . 95
A.1. **Esquema básico de un código Arduino.** . . . 106
A.2. **Encender y apagar un led en Raspberry.** . . . 108
A.3. **Utilización básica del sensor GL5528.** . . . 111
A.4. **Utilización básica del sensor de lluvia YL38.** . . . 112
A.5. **Utilización básica del sensor de temperatura y humedad DHT22.** . . . 114
A.6. **Utilización básica del sensor de pulso cardíaco.** . . . 115
A.7. **Utilización básica del acelerómetro y giroscopio MPU6050.** . . . 116
A.8. **Esqueleto de código HTML5.** . . . 124
A.9. **Sintaxis de una regla en CSS.** . . . 124
A.10. **Uso de propiedades experimentales en distintos navegadores.** . . . 124
A.11. ***Hello world* en JavaScript.** . . . 125
A.12. **Creación de un objeto en JavaScript.** . . . 125
A.13. ***Hello world* usando React.** . . . 126
A.14. ***Hello world* en NodeJs.** . . . 127
A.15. **Inventario con todas las características de todas las estaciones climatológicas.** . . . 129
A.16. **Ejemplo básico del uso de la API de *Google Maps*, para visualizar en un mapa Ciudad Real.** . . . 130

# Listado de acrónimos

| | |
|---|---|
| **FEDME** | Federación Española de Deportes de Montaña y Escalada |
| **CSD** | Consejo Superior de Deportes |
| **GPS** | *Global Positioning System* |
| **TFG** | Trabajo de Fin de Grado |
| **API** | *Application Programming Interface* |
| **CPU** | *Central Processing Unit* |
| **RAM** | *Random Access Memory* |
| **ROM** | *Read-Only Memory* |
| **IDE** | *Integrated Development Environment* |
| **SD** | *Secure Digital* |
| **GPIO** | *General Purpose Input/Output* |
| **RISC** | *Reduced Instruction Set Computing* |
| **GPU** | *Graphics Processing Unit* |
| **SDRAM** | *Synchronous Dynamic Random-Access Memory* |
| **SRAM** | *Static Random-Access Memory* |
| **EEPROM** | *Electrically Erasable Programmable Read-Only Memory* |
| **USB** | *Universal Serial Bus* |
| **RTOS** | *Real Time Operative System* |
| **LDR** | *Light-Dependent Resistor* |
| **IBI** | *Interbeat Interval* |
| **SDK** | *Software Development Kit* |
| **OPENJDK** | *Open Java Development Kit* |
| **AVDM** | *Android Virtual Device Manager* |
| **AVD** | *Android Virtual Device* |
| **HTTP** | *Hypertext Transfer Protocol* |
| **RFCOMM** | *Radio Frequency Communications* |

| | |
|---|---|
| **L2CAP** | *Logical Link Control and Adaption Protocol* |
| **ACL** | *Asynchronous Connection-Oriented Logical* |
| **SCO** | *Synchronous Connection-Oriented* |
| **SDP** | *Service Discovery Protocol* |
| **UUID** | *Universally Unique Identifier* |
| **SSP** | *Secure Simple Pairing* |
| **IoT** | *Internet of Things* |
| **WWW** | *World Wide Web* |
| **CERN** | Conseil Européen pour la Recherche Nucléaire |
| **HTML** | *HyperText Markup Language* |
| **XHTML** | *eXtensible HyperText Markup Language* |
| **DTD** | *Document Type Definition* |
| **DOM** | *Document Object Model* |
| **XML** | *eXtensible Markup Language* |
| **CSS** | *Cascading Style Sheets* |
| **NPM** | *node package manager* |
| **REST** | *Representational State Transfer* |
| **SOAP** | *Simple Object Access Protocol* |
| **RPC** | *Remote Procedure Call* |
| **AEMET** | Agencia Estatal de Meteorología |
| **WGS 84** | *World Geodetic System 84* |
| **SMS** | *Short Message Service* |
| **ARVA** | Appareil de Recherche de Victimes d'Avalanches |
| **PLB** | *Personal Locator Beacon* |
| **MOB** | *Man Overboard* |
| **GEOSAR** | *Geosynchronous Search and Rescue Satellite System* |
| **LEOSAR** | *Low Earth Orbit Search and Rescue Satellite System* |
| **SEND** | *Satellite Emergency Notification Devices* |
| **PMR** | *Personal Mobile Radio* |
| **PCB** | *Printed Circuit Board* |
| **PT** | Paquetes de trabajo |
| **T** | Tareas |

| | |
|---|---|
| **HDD** | *Hard Disk Drive* |
| **SSD** | *Solid-State Drive* |
| **NMEA** | *National Marine Electronics Association* |
| **MVC** | *Model-View-Controller* |
| **MAC** | *Media Access Control* |
| **LOPD** | Ley Orgánica de Protección de Datos |
| **JSON** | *JavaScript Object Notation* |

Capítulo 1

# Motivación del trabajo

El **senderismo** se ha convertido en una de las prácticas deportivas preferidas por los españoles. Podemos darnos cuenta de ello si echamos un vistazo a los datos del año 2017 [FED], con más de $112.000$ personas federadas según la Federación Española de Deportes de Montaña y Escalada (FEDME). De entre los federados, según un estudio de la FEDME [FED17b] un $96,7\,\%$ de los encuestados practican senderismo y un $86,1\,\%$ practican montañismo. Si tenemos en cuenta también las licencias autonómicas, el número de afiliados sube hasta más de $200.000$. Según datos del Consejo Superior de Deportes (CSD) [CSD], se puede observar que el senderismo ocupa el quinto lugar en la lista de deportes con más federados.

La relación entre el hombre y la montaña [MS04] es tan antigua como la propia humanidad, ya que gracias a la montaña el hombre ha podido satisfacer sus necesidades, las cuales han ido cambiando a lo largo del tiempo. De ser un simple lazo natural en los inicios de la humanidad, a ser hoy en día un entorno social dónde las personas pueden compartir sus inquietudes. Las prácticas sociales más desarrolladas en la montaña son aquellas que tienen que ver con la expedición, bien sea en solitario o en grupos de expedición.

Los grupos de expedición son cada vez más numerosos, encontrándose en ellos senderistas con distinto nivel de experiencia en la montaña. Muchos senderistas se lanzan a la aventura, sin experiencia y sin conocer las dificultades que presentará el terreno, por lo que la existencia de un guía en los grupos de expedición es clave. En estos grupos hay personas de distinto nivel y forma física, por lo que hacer una ruta apta para todo el mundo (tanto en recorrido, como en velocidad de marcha) puede llegar a ser una tarea complicada para el guía. Además, el guía es el responsable de la seguridad de los senderistas y en grupos numerosos es más difícil tener controlados todos los riesgos sin el uso de mecansimos de monitorización y control. En el año 2016, un $0,16\,\%$ de los federados tuvo que ser rescatado [FED17a].

Conforme los senderistas van completando rutas, empiezan a buscar otras de más duración, que transcurran por lugares más difíciles y en medios más hostiles (nieve, agua, zonas rocosas), rutas en las que existen menos señales que identifiquen el camino, etc. En definitiva, rutas en las que la seguridad es un factor clave.

La expedición en la montaña puede resultar peligrosa. Precisamente, la peligrosidad es

un factor que los humanos encuentran motivante y no es sorprendente que se busquen actividades con este factor de peligrosidad. Las actividades peligrosas hacen que las personas puedan volverse adictas a la adrenalina [Azi18]. La adrenalina es un neurotransmisor que provoca excitación, por lo que el cerebro puede desensibilizarse a esta hormona una vez que ha sido expuesto a la misma y la única forma de obtener una sensación igual es aumentar la peligrosidad de la actividad. Según el estudio [Ewe85], los individuos con mayor experiencia son más propensos a esta desensibilización, ya que buscan sentimientos relacionados con la euforia, el desafío personal y la toma de decisiones, mientras que los individuos con menor experiencia buscan la participación en actividades sociales y el sentimiento de evasión.

Además hay otros factores que están realacionados directamente con la peligrosidad, como puede ser la dispersión del grupo de expedición. Un grupo muy disperso es difícil de controlar y por tanto, es más propenso a sufrir accidentes. La rápida actuación ante un accidente, como puede ser la caída de un miembro del grupo, es clave para minimizar el impacto de la caída, por lo que el grado de dispersión es una cuestión a tener controlada. Cuanto menor sea este grado de dispersión, más rápida será la actuación ante un accidente. Los factores meteorológicos como la lluvia y la iluminación del ambiente también son claves en el aumento de la peligrosidad.

A grandes altitudes (por encima de los 2500 metros) [Esf14], el riesgo y el peligro aumenta en los senderistas. El principal peligro es el llamado «mal de altura». Con el crecimiento de la expedición en la montaña, y la participación de gente inexperta, los accidentes en la misma aumentan, ya sea por una baja preparación o por comportamientos inadecuados. Los guías de expedición, intentan minimizar los accidentes, priorizando la seguridad. Por esto, es crucial que los guías tengan mecanismos que les faciliten esta tarea.

Con el auge del senderismo se han empezado a comercializar bastantes dispositivos cuyo objetivo es la mejora de la seguridad para los senderistas, los cuales poseen GPS, sensor de frecuencia cardiaca, herramientas para medir altitud y otros datos relevantes. Cabe destacar que son dispositivos de alto coste, con la mayoría por encima de los 300-400 euros. También, han nacido aplicaciones de tipo social que buscan el seguimiento de las rutas individuales y la competitividad entre los usuarios en ciertos tramos de esas rutas.

El factor común de estas aplicaciones y dispositivos es la individualidad. Con estas aplicaciones no se resuelve la problemática del control de un grupo, para maximizar la seguridad de sus miembros. Sigue siendo difícil la correcta comunicación de los guías con los miembros de su grupo, que con un simple golpe de vista sepa dónde se encuentran los senderistas para evitar que los mismos se pierdan por el camino, además de llevar un control sobre el estado de salud de cada uno de los miembros del grupo.

Los dispositivos usados en expediciones en la montaña deben tener algunos requisitos especiales [FD17], como son la capacidad de trabajar en tiempo real. La información que

proporcionan estos dispositivos (por ejemplo parámetros de localización, altitud, salud física, etc.) debe estar disponible de forma casi inmediata. Además, no es viable portar un dispositivo de grandes dimensiones, por comodidad del senderista al llevarlo encima (deben ser dispositivos con poco peso y de dimensiones reducidas). Se suelen utilizar, por tanto, dispositivos pequeños y de bajo poder de cómputo, por lo que la eficiencia en la programación de estos dispositivos es clave.

En este contexto nace este proyecto, como una herramienta para monitorizar los eventos que se puedan producir en un grupo de expedición y para ayudar a los guías a tomar decisiones para maximizar la seguridad de los integrantes durante la expedición.

Para maximizar la eficiencia se tendrán en cuenta diseños de *hardware* empotrado sobre un microcontrolador. De esta forma, el diseño se centrará en resolver la problemática planteada por lo que la productividad será mayor que usando un sistema de propósito general. Además, con arquitecturas de *hardware* embebido, se resuelve fácilmente la cuestión de la computación en tiempo real.

De acuerdo a esta problemática se pretende realizar, por tanto, un dispositivo *hardware* equipado con varios sensores, que permita recoger la información necesaria del entorno. Este dispositivo consistirá en un microcontrolador Arduino [Mar11] equipado con sensores adecuados para la monitorización del grupo de expadición, que tomará la información recibida de los mismos [Bel13] y la enviará a un dispositivo móvil conectado con el microcontrolador.

Se desarrollará también una aplicación móvil [Bur15] que tomará todos los datos provenientes del microcontrolador y realizará un procesamiento inteligente y en tiempo real de los mismos, para detectar posibles situaciones de riesgo como pueden ser caídas, una gran dispersión de los miembros del grupo o condiciones climatológicas adversas entre otras. Esta información será presentada a los miembros de la expedición a través de una interfaz.

Por último, se va a desarrollar una plataforma web, que servirá para mostrar un histórico de todas las expediciones que el senderista ha realizad. Se mostrarán las estadísticas de la expedición de una forma amigable. Esta plataforma web también podrá ser usada por la familia o amigos de los senderistas para visualizar en tiempo real los datos del senderista en la expedición.

Con esta propuesta se pretende cubrir el vacío que existe en dispositivos y aplicaciones para el control de grupos de senderismo ya que se tendrán en cuenta los datos recibidos de todas las personas para hacer un procesamiento conjunto y mostrar la información relevante.

El proyecto servirá a los guías de estos grupos que verán en todo momento dónde se encuentran los senderistas y su estado de salud, entre otra información, y les servirá para tomar decisiones de una forma más fácil sobre si continuar la marcha, bajar el ritmo de la misma, hacer una parada para una reagrupación, etc. Con este proyecto se mejorará la

seguridad de los senderistas, ya que ante la identificación de caídas [AB08] o empeoramiento del estado de salud se enviará un mensaje al guía para que actúe en consecuencia.

## 1.1 Trabajos previos

### 1.1.1 Aplicaciones existentes para rutas de expedición

Es notorio destacar que apenas existe alguna aplicación con una funcionalidad parecida a la que se expone en el presente proyecto. La mayoría de aplicaciones para dispositivos móviles que se encuentran tienen que ver con la gestión de rutas en la montaña y la predicción de la climatología. Ninguna de ellas se enfrenta al problema del control de un grupo de senderismo ya que están enfocadas de forma individual. Por tanto, existe un vacío en lo referente a aplicaciones que permitan mejorar la seguridad de un grupo de expedición.

**AllTrails** *AllTrails* es una aplicación móvil para iOS y Android que permite la búsqueda de nuevas rutas de expedición además de conocer a otros senderistas. *AllTrails* permite la búsqueda de rutas cercanas, a partir de la localización *Global Positioning System* (GPS) actual. Esta aplicación proporciona algunos consejos acerca de obstáculos en el camino e información sobre lugares importantes cercanos a la ruta que se está realizando. La descarga de la aplicación es gratuita, tanto en iOS como en Android pero para el acceso a contenido más avanzado, como la descarga de mapas *offline*, es necesario pagar una suscripción.

**GPX Viewer** *GPX Viewer* es una aplicación móvil y de escritorio que permite evitar problemas de orientación. Esta aplicación permite cargar rutas previamente descargadas de internet en varios formatos y visualizarlas después sobre *Google Maps*. Con esta aplicación es posible también guardar tus movimientos para crear la ruta por la que transcurres sobre un mapa. Esta ruta se podrá almacenar en el dispositivo para visualizarla posteriormente o para volver a reproducirla en cualquier otro momento.

**Primeros Auxilios - Cruz Roja** Se trata de una aplicación móvil desarrollada por la Cruz Roja con el objetivo de enfrentarse a las emergencias cotidianas. Esta aplicación proporciona consejos, información interactiva y mediante vídeos para guíar al usuario en los primeros auxilios del día a día. Es una aplicación útil para saber como reaccionar ante crisis de asma, sangrados, lipotimias y otras situaciones que pueden ocurrir en la montaña. además, es una aplicación que funciona sin cobertura y tiene una lista con contactos de urgencia para llamar a servicios de emergencias de una forma rápida y sencilla.

[2]Imagenes originales extraídas de `https://play.google.com/store/apps/details?id=com.vecturagames.android.app.gpxviewer&hl=es` y de `https://bearfoottheory.com/find-local-hiking-trail`


Figura 1.1: Aplicación móvil *GPX Viewer*. [1]

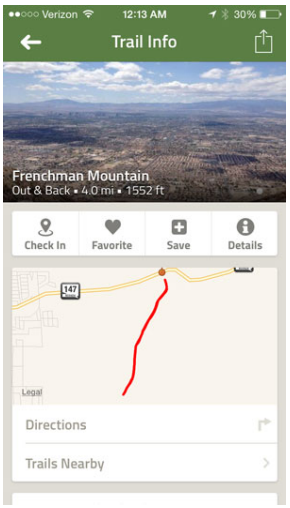



Figura 1.2: Aplicación móvil *AllTrails*. [2]

**Alpify** Esta aplicación móvil multiplataforma (iOS y Android) desarrollada por `Safe365`[3] se centra en el auxilio rápido de un accidentado en la montaña o practicando cualquier otro deporte como puede ser el ciclismo. Es una aplicación necesaria en el caso de accidentes o cambios bruscos de climatología que posee un gran botón rojo central que en el caso de ser pulsado envía la localización exacta en la que te encuentras y realiza una llamada al servicio de emergencias. En el caso de no haber cobertura, se envía un mensaje de texto por *Short Message Service* (SMS). Se le puede facilitar a la aplicación un teléfono de contacto de algun amigo o familiar. La aplicación comunicará este número de teléfono automáticamente a los servicios de emergencias para que éstos puedan ponerse en contacto con los allegados del accidentado. En el caso en el que el accidente haya sido un desmayo y al senderista no le haya dado tiempo a pulsar el botón, una vez que se haya producido la denuncia de desaparición, `Safe365` tendrá todos los datos de la ruta, algo que puede ser de utilidad para los servicios de rescate.

### 1.1.2 Dispositivos existentes para rutas de expedición

Al igual que se ha comentado en el punto anterior con las aplicaciones existentes en el mercado para rutas de expedición, no existen dispositivos con una funcionalidad igual o muy parecida a la del dispositivo que se expone en el presente proyecto. La mayoría de dispositivos en el mercado están centrados en maximizar la seguridad individual, para responder de forma rápida ante accidentes. También se pueden encontrar muchos dispositivos de monitorización individual, que se encargan de tomar información sobre el portador del mismo (información de localización, de pulso cardíaco, de altitud y de carácter social, entre otras).

[3] `https://safe365.com/es/`

[4] Imagen original extraída de `https://play.google.com/store/apps/details?id=com.cube.arc.fa&hl=es_419`

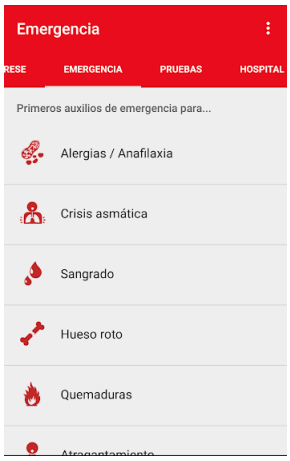



Figura 1.3: Aplicación móvil Primeros Auxilios. [4]

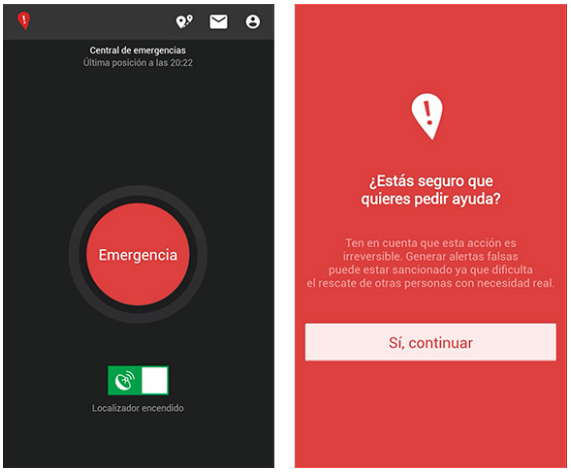



Figura 1.4: Aplicación móvil *Alpify*.

**ARVA** Un ARVA (véase Figura 1.5), o dispositivo de búsqueda de víctimas de avalanchas, es un aparato que emite de forma continua una señal emisora muy potente que se puede detectar incluso bajo una gran capa de nieve, al quedar el montañista sepultado por un alud en la montaña. Los equipos de rescate y otros compañeros de expedición llevan otros dispositivos ARVA, que si se ponen en modo de recepción, se usan para recoger la señal del ARVA del montañista sepultado y localizar con una gran exactitud al mismo.

Cuando un montañista sale de expedición con un ARVA entre su equipo, este tiene que ir siempre con el montañista y nunca en la mochila u otro objeto que no sea él mismo. Esto es debido a que ante avalanchas de nieve, es posible que la mochila se pierda y se separe del montañista por lo que la señal del ARVA indicaría la localización de la mochila y no del montañista. También es importante que el ARVA vaya debajo de nuestra ropa para que en caso de avalancha no entre en contacto directo con la humedad de la nieve y las baterías duren más. Hay que intentar colocarlo en zonas blandas del cuerpo para que en caso de caída, no nos produzca fisuras en huesos. Además, el ARVA debe estar en modo de emisión y el modo recepción tiene que estar restringido a búsqueda de compañeros.

Hoy en día las mochilas más técnicas traen incorporado un ARVA. Esto le resulta muy cómodo al montañista ya que no necesita portar ningún otro dispositivo mas allá de su chaqueta.

Inicialmente, los ARVA transmitían a una frecuencia de 2275kHz, pero en el año 1986 se estableció un estándar para que todos estos dispositivos emitiesen a una frecuencia de 457kHz. Todos los dispositivos ARVA, sea cual sea su marca, son comptabiles entre si, consiguiendo así que cualquier portador de un dispositivo ARVA sea capaz de localizar otro dispositivo cualquiera y asegurando que si un montañista se queda atrapado con su ARVA emitiendo, podrá ser encontrado por cualquier otro ARVA de cualquier marca.

[5]Imagen original extraída de `https://images.blue-tomato.com/is/image/bluetomato/301529492_front.`


Figura 1.5: Dispositivo ARVA para la búsqueda de montañistas en avalanchas. [5]

Según [Fal94] de 105 personas enterradas y rescatadas por aludes, antes de que transcurriesen 15 minutos desde que fueron sepultadas, 98 fueron encontradas con vida y 7 personas fueron encontradas muertas. Esto se corresponde con que un 93 % de las personas fueron encontradas con vida entre los minutos 0 y 15 después de su accidente. Cabe destacar que entre los minutos 15 y 45, el porcentaje de encontrar a una persona con vida se reduce hasta un 26 %, por lo que los primeros minutos después de una avalancha son cruciales. Por eso es importante portar un ARVA en tu equipo de expedición, ya que es un dispositivo que precisamente intenta minimizar el tiempo de búsqueda de desaparecidos.

Además es importante que tus compañeros de expedición porten también un ARVA y sepan usarlo ante situaciones de emergencia, ya que según este estudio un 71 % de las víctimas de aludes socorridas por compañeros de expedición sobreviven, frente a un 13 % de montañistas recogidos con vida por los cuerpos de socorro. Esto se debe principalmente a la criticidad de encontrar al montañista sepultado en los primeros 15 minutos, después de haber ocurrido el alud.

Existen dos tipos de dispositivos ARVA, los dispositivos analógicos y los digitales. Cuando un ARVA analógico se encuentra en un estado de recepción e intercepta una señal de otro ARVA que se encuentra en estado de emisión, se encarga de amplificarla y emitir un sonido sonoro "*bip*" que es más intenso cuanto más cerca se encuentre de la fuente de emisión de señales. Los dispositivos digitales convierten las señales procedentes del ARVA en estado de emisión a indicadores visuales (mediante un microprocesador interno). Estos indicadores pueden ser flechas que muestran la dirección y aumentan la intensidad de sus colores conforme nos acercamos a la fuente de la señal. Los indicadores pueden ser también numéricos, que nos indican la distancia (no necesariamente medida en metros) a la fuente de emisión. Cuanto más cerca estemos, más pequeños seran estos valores numéricos.

jpg-vKXjlekI60xiNRwUj6n0Y-rZL04/Tracker+Dts+ARVA.jpg?\protect\T1\textdollarb1\protect\T1\textdollar

**Sonda** Una sonda consiste en una vara larga (en torno a unos dos metros y medio) y plegable, cuyo principal objetivo es determinar a qué profundidad se encuentra enterrada la víctima. Es una buen complemento a un ARVA, ya que éstos nos dan una precisión bastante exacata pero con la sonda, la precisión es la máxima posible. Una vez que se ha llegado al punto en el que el ARVA da la máxima señal, hay que proceder al sondeo, realizando círculos concéntricos y con una señaración entre ellos de unos 25 centímetros. Cuando tenemos localizada a la víctima con una gran precisión, habrá que dejar clavada la sonda en ese lugar perpendicular a la pendiente. Hay que evitar pisar en la zona para no destruir las posibles bolsas de aire que se hayan creado y puedan ayudar a respirar al montañista sepultado.

Figura 1.6: Sonda de fibra de carbono. [6]

**PLB** Las PLB o balizas personales de localización son unos dispositivos creados para el rescate en la montaña y en aguas abiertas. Estos dispositivos envían una señal de emergencia cuando el afectado se encuentra en una situación de peligro de muerte o situación *Man Overboard* (MOB). Estas balizas transmiten la posición con una gran precisión a servicios de búsqueda y salvamento a través de los satélites *Low Earth Orbit Search and Rescue Satellite System* (LEOSAR) y *Geosynchronous Search and Rescue Satellite System* (GEOSAR). Los servicios de salvamento son internacionales y se fundaron por cuatro países (Estados Unidos, Canadá, Francia y Rusia), extendiéndose en la actualidad hasta 43 países. Cuando reciben una alerta, notifican a servicios de búsqueda locales, que procederán al rescate del afectado.

Un inconveniente de las PLB es que una vez activada la alerta no se puede cancelar, por lo que si se emite una alerta de forma accidental no existe una manera para comunicar el falso positivo. Además, cuando se compra una PLB, hay que registrarla (de forma gratuita) ante una autoridad nacional ya que si no se hace, la baliza es totalmente inútil. En este registro, se indican datos personales como el nombre, la dirección, un número de contacto en caso de emergencia, etc. además de datos médicos, para minimizar el tiempo de respuesta de los

[6] Imagen original extraída de `https://static.forumsport.com/img/productos/1000x1000/ARVA%20VARIOS%20MONTA%C3%91A%20SONDA%20CARBON%202.40%20COMPACT-430360.jpg`

equipos médicos en un rescate. Estos datos se deben actualizar cada dos años y si cambia el dueño de una PLB, se debe reportar para almacenar los datos del dueño actual.


Figura 1.7: Baliza PLB para la localización y rescate de accidentados. [7]

Las PLB tienen baterías de gran duración (pueden durar hasta cinco años) que permanecen en un estado inactivo hasta que se activa la PLB. Estas baterías son capaces de funcionar en condiciones extremas (pueden transmitir señales a temperaturas inferiores a los -20ºC). Sin embargo, los cambios de batería suelen ser caros y requieren enviar el dispositivo completo para realizar este cambio de batería.

***Satellite Messengers*** Los dispositivos de notificación por satélite o SEND (*Satellite Emergency Notification Devices*) [dD12] se usan para realizar una comunicación por mensajes de texto en caso de emergencia, con algún familiar o con los equipos de rescate. Al contrario que las PLB, los mensajes enviados con los dispositivos de notificación por satélite no son gratuitos. Estos dispositivos no usan la misma red de satélites que las PLB y normalmente no tienen cobertura en ciertas zonas de Alaska y en la Antártida, por lo que antes de viajar a algún área remota es necesario comprobar la cobertura en ese lugar.

Los equipos de rescate prefieren el uso de este tipo de dispositivos ya que se puede cancelar una falsa alarma y evita costes innecesarios derivados de un falso positivo. Se han desarrollado incluso aplicaciones móviles (como `Earthmate`[8]) que permiten conectar el dispositivo de notificación por satélite con el teléfono móvil para facilitar el intercambio de mensajes.

La suscripción de dispositivos de notificación por satélite no es gratuita y es el mayor gasto ya que estos dispositivos suelen tener un precio inferior a las PLB. Cada empresa que fabrica

[7] Imagen original extraída de `https://images-na.ssl-images-amazon.com/images/I/81fx0wksbtL._SL1500_.jpg`

[8] `https://buy.garmin.com/es-ES/ES/p/577212`

[9] Imagen original extraída de `https://www.rei.com/media/11b875e1-4993-478c-8473-ea475c178978?size=784x588`


Figura 1.8: Dispositivo de notificación por satélite. [9]

dispositivos de notificación por satélite tiene sus propias cuotas de suscripción, bien anuales o bien mensuales. Por ejemplo, la marca de *satellite messengers* SPOT[10], tiene una tasa anual de unos 150 dólares, además de un coste de mantenimiento de red de 25 dólares anuales. Estas tasas incluyen la capacidad para notificar emergencias, enviar mensajes personalizados, permitir a tu familia un seguimiento casi en tiempo real de tu ruta y enviarles un mensaje para decir que estás bien.

La batería de los dispositivos de notificación por satélite varía dependiendo del modelo, pero puede ir desde un par de días hasta unos 20 días, por lo que las baterías son recargables. Esto puede ser una contra, ya que hay que recordar tener las baterías siempre recargadas.

En la Tabla 1.1 se puede observar una comparación de características y precio entre los dispositivos PLB y los dispostivos SEND. Se puede observar que el precio inicial de adquisición de un dispositivo PLB es mayor que el de un dispositivo SEND, sin embargo la tasa anual que hay que pagar a la marca del dispositivo SEND para mantener la red y para poder enviar mensajes de emergencia y personalizados, hace que a la larga estos dispositivos sean más caros. Además los dispositivos SEND usan satélites distintos a las PLB y puede ser que ante situaciones en las que el montañista se encuentre bajo una vegetación muy densa o en países como Rusia, con regulaciones estrictas sobre la precisión de los dispositivos GPS, la cobertura no sea buena y el dispositivo sea inútil.

Sin embargo, cuando se usa una PLB, la batería suele durar unas 24 horas y una vez usada, es necesario cambiar la batería (cuyo coste es de unos $150) o directamente comprar una PLB nueva. Los dispositivos SEND envían la posición actual casi en tiempo real, cada diez minutos, mientras que las PLB no envían ningún tipo de señal y permanecen en un estado inactivo (con el objetivo de no gastar la batería) hasta que el montañista pulsa el

[10] https://www.findmespot.com/en/index.php?cid=103

botón de socorro. En los dos dispositivos es necesario pulsar un botón para que se produzca la alerta por lo que si el accidentado cae mareado y no puede pulsar el botón de socorro, no se producirá la alerta y el dispositivo será inútil. Aun así, como los dispositivos SEND envían la posición cada diez minutos, la familia puede ver si el accidentado se mueve o no en un cierto período de tiempo y detectar un posible accidente si el montañista se encuentra en un estado inactivo por un tiempo fuera del prudencial.

| **Tipo** | PLB | SEND |
|---|---|---|
| **Marca** | PLB básico | SPOT gen3 |
| **Precio de compra** | Alrededor de 250$ | Alrededor de 150$ |
| **Tasa anual** | No hay tasa | Alrededor de 175$ |
| **Envío de mensajes** | No disponible | Sólo mensajes predefinidos |
| **Vida de la batería** | 5 años (en un estado de reposo, sin ser utilizado) | Entre 7 y 45 días |
| **Coste máximo en 5 años** | Alrededor de 250$ | Alrededor de 1025$ |

Cuadro 1.1: Comparación entre dispositivos PLB y SEND.

**Sistemas de Radio** Cuando nos encontramos en lugares en los que el uso del teléfono móvil no es posible debido a la falta de cobertura y no poseemos dispositivos por satélite como los PLB y SEND que se han expuesto anteriormente, podemos usar como alternativa sistemas basados en comunicación por radio. Estos sistemas no tienen un rango amplio de alcance y los que si lo poseen no pueden usarse por cualquier persona, ya que es necesario poseer una autorización administrativa, la llamada «Licencia de Radioaficionado».

Los sistemas PMR son sistemas de uso libre, es decir, no es necesaria ninguna licencia para usarlos. Estos sistemas operan a ciertas frecuencias en las que no es necesaria autorización previa. Estas frecuencias oscilan entre los 446MHz y los $446,2$MHz, rango dividido en 16 canales. El alcance de estos dispositivos está entre los 5 y 10 kilómetros, pero se pueden lograr distancias mayores en terrenos llanos y sin obstáculos. Dentro de un mismo canal, se pueden disponer de 38 subtonos distintos, pudiendo entablar una conversación distinta en cada uno de los subtonos. En torno a estos sistemas se ha creado el llamado «Canal 7-7 PMR».

El Canal 7-7 PMR[12] consiste en una estrategia que busca que todos los grupos de montaña y los montañistas individuales usen el Canal 7 y el Subtono 7 (frecuencia de canal de 446.08125MHz y frecuencia de subtono de 85.4Hz) como medio de comunicación. Esta estrategia surgió al buscar a un deportista perdido en la montaña, ya que era necesario que todas las personas que participaban en la búsqueda se comunicasen en un mismo canal y subtono.

[11] Imagen original extraída de https://images-na.ssl-images-amazon.com/images/I/71Dz7lkZzVL._SX355_.jpg

[12] http://www.canal77pmr.com/


Figura 1.9: Pareja de *walkies* PMR. [11]


Figura 1.10: Anuncio para el uso del Canal 7-7 en la montaña. [13]

## 1.2 Estructura del documento

El presente TFG se compone de los siguientes capítulos:

**Capítulo 1. Motivación del trabajo**
Se trata del capítulo actual. En él se plantea la problemática existente que se pretende resolver con este TFG.

**Capítulo 2. Objetivos**
En este capítulo se presentan los objetivos a alcanzar que se deben satisfacer en el desarrollo del TFG, para alcanzar el éxito en el mismo.

**Capítulo 3. Método de trabajo**
En este capítulo se muestra la metodología de desarrollo que se ha llevado a cabo durante el presente proyecto, además de las distintas herramientas usadas para el mismo. Se presentarán también las distintas fases para la elaboración de este TFG.

**Capítulo 4. Resultados**
Este capítulo muestra los resultados obtenidos, fruto de las pruebas realizadas con el

[13] Imagen original extraída de `https://pmrjaen.files.wordpress.com/2018/02/canal77.jpg?w=900`

prototipo desarrollado.

**Capítulo 5. Conclusiones**

Capítulo 2

# Objetivos

En este capítulo se va a presentar tanto el objetivo que se pretende cumplir con el desarrollo de este proyecto, como los distintos hitos que se conseguirán para lograr el objetivo principal.

## 2.1 Objetivo general

El objetivo principal de este TFG es **la monitorización de la actividad en grupos de expedición, para velar por su seguridad**.

Para llevar a cabo este objetivo, se pretende realizar un análisis en tiempo real del estado individual de cada uno de los participantes del grupo, así como un análisis grupal de tal forma que el guía pueda tener una visión global del grupo de expedición. Como resultado de este análisis se pretende obtener una serie de alertas respondiendo a posibles situaciones de riesgo, tales como caídas o pérdidas de miembros del grupo de expedición.

## 2.2 Objetivos específicos

1) **Diseño y desarrollo de un dispositivo *hardware* equipado con distintos sensores para la monitorización de la actividad de un integrante de un grupo de expedición**

   Para realizar un análisis inteligente de los parámetros que se exponen a continuación, se pretende diseñar y desarrollar un dispositivo *hardware*, que realice un procesamiento básico de los datos que capta por los sensores.

   Este dispositivo estará constituido por un microcontrolador y un conjunto de sensores. El microcontrolador se encargará de tomar datos de los sensores, además de realizar un procesamiento básico de los datos recibidos, con el objetivo de eliminar valores no válidos que puedan proporcionar los sensores en un momento dado.

   a) Control de la posición del cuerpo, con el objetivo de detectar posibles caídas. Para ello se procesarán los datos que se recogen de sensores como el acelerómetro y el giroscopio.
   b) Geolocalizar a los usuarios mediante GPS. Se especificará un límite máximo de separación entre el guía y los miembros de la ruta de tal forma que cuando un

participante sobrepase este límite de separación, se envíe un mensaje de alerta al guía. El guía también podrá visualizar en tiempo real dónde se encuentran todos los senderistas que conforman la ruta.

c) Captación de datos sobre temperatura, lluvias y humedad, que son factores de riesgo en el estado de salud de los senderistas, ya que pueden propiciar la aparición de la deshidratación, aumentar las probabilidades de caídas e influir en el ritmo de la ruta.

d) Diseño de una PCB una vez realizado un primer prototipo del dispositivo *hardware*, para eliminar los cables de prototipado y conseguir un prototipo mucho más limpio. Se consigue un prototipo de tamaño más reducido, más manejable y sin riesgos de que los sensores dejen de funcionar por la desconexión puntual de un cable.

2) **Diseño y desarrollo de una aplicación móvil que recibirá los datos del microcontrolador, los analizará y mostrará información relevante para integrantes y guías en tiempo real**

Esta aplicación móvil se encargará de comunicarse con el dispositivo *hardware* para realizar un procesamiento mayor de los datos. Por motivos de eficiencia, se delega el procesamiento de los datos captados por el dispositivo *hardware* en un teléfono móvil, o cualquier otro dispositivo que posea un sistema operativo *Android*. Este procesamiento de los datos incluye el análisis en tiempo real de los mismos, para obtener información relevante para el grupo de expedición. No solo se realizará un análisis a nivel de integrante de la expedición, sino que también se analizará el grupo en sí mismo.

La información procedente del análisis inteligente será mostrada a los usuarios de tal forma que sea relevante para ellos. Es decir, los guías de la expedición tendrán toda la información a nivel de grupo de expedición. Esta visión global les otorga la posibilidad de anticiparse a situaciones de peligro que se pudiesen generar, como por ejemplo la dispersión excesiva de los miembros del grupo. A su vez, los integrantes de la expedición tendrán una información más individual.

La aplicación móvil tomará los datos recogidos por el dispositivo *hardware* por medio de *Bluetooth*. Además, recogerá también datos gracias a los sensores integrados en los dispositivos móviles y gracias a APIs externas. Se hace esto para comparar los valores recogidos desde el dispositivo *hardware* con estos datos captados con el dispositivo móvil y cerciorarnos de que son correctos. De esta forma, se aumenta la fiabilidad de la información final que será otorgada al usuario. También se pretende abaratar el coste del prototipo y reducir el tamaño del microcontrolador diseñado, reduciendo el número de sensores que se van a usar.

Por tanto, se pretende realizar una comunicación de la aplicación móvil con el dispositivo *hardware*, a través de la cual recibirá los datos recogidos por los sensores. El dispositivo móvil realizará una análisis inteligente de los datos y generará alertas si detecta alguna anomalía. Finalmente, la aplicación móvil enviará la información procesada a una plataforma web, para que los senderistas puedan ver de una forma amigable la información de la ruta de una forma mucho más detallada. Se pretende generar una solución escalable, que sea eficiente para grupos de expedición muy numerosos y que pueda ser usada a la vez por muchos grupos de expedición.

3) **Análisis inteligente y en tiempo real de los datos captados por los sensores del microcontrolador**

   Para conseguir el objetivo principal del presente proyecto es crítico realizar un análisis inteligente de los datos que se han captado previamente. En primer lugar, se realizará un procesamiento de los datos para manipularlos en las unidades de medida necesarias. Se desarrollarán los algoritmos que se exponen a continuación. Estos algoritmos, tomarán como entradas los datos previamente procesados.

   a) Detección de caídas. A partir de los datos obtenidos por medio del acelerómetro, se implementará un detector de caídas que generará una alerta y avisará al guía en el caso de que un integrante del grupo de expedición sufra una caída. Se pretenden minimizar tanto los falsos positivos, como los falsos negativos, para que las alertas producidas por el detector de caídas sean tenidas en cuenta.
   b) Grado de dispersión del grupo. A partir de las posiciones individuales de cada uno de los integrantes del grupo con respecto al guía, se generará un coeficiente que indique el grado de dispersión del grupo. Este coeficiente puede usarse para prevenir futuras situaciones de riesgo de pérdida de miembros, ya que el guía puede tomar la decisión de realizar una parada porque el grupo se encuentra muy disperso, entre otras decisiones.
   c) Grado de riesgo de caída. A partir de los datos de sensores de temperatura, humedad, lluvia e iluminación y otros datos como el ritmo actual al que se camina, se pretende generar un coeficiente que indique el riesgo de caída existente en la expedición. A partir de este coeficiente, el guía puede tomar la decisión de aminorar el ritmo actual para disminuir este riesgo de caída.

4) **Diseño y desarrollo de una plataforma web para la visualización de los resultados de la expedición**

   En la plataforma web se verán reflejadas las estadísticas de la ruta, así como la información que se considere relevante. La plataforma web se usará, por tanto, para que los usuarios del grupo de expedición puedan consultar los detalles de la expedición, bien siga la expedición en curso o ya haya finalizado. Se mantendrá un histórico de

expediciones de las que un senderista ha formado parte para que pueda consultar la información de todas ellas y así ver su progresión.

Aunque los senderistas posean toda la información relevante para ellos en su dispositivo móvil mientras la expedición está en curso, también existe la posibilidad de que puedan contemplar las estadísticas de la expedición en tiempo real en la plataforma web. Esto es útil sobre todo para que familiares o amigos del senderista puedan ver en vivo dónde se encuentra el senderista y si ha generado algún tipo de alerta que pueda indicar que se encuentre o se haya encontrado en una situación de peligro.

Se pretende, por tanto, realizar una plataforma web que servirá tanto a los senderistas, como al guía del grupo de expedición, para visualizar de una manera amigable la información recogida y procesada anteriormente por medio del dispositivo móvil.

| | **Recoger datos** | **Limpiar Datos** | **Análisis Inteligente** | **Visualización en tiempo real** | **Histórico de expediciones** |
|---|---|---|---|---|---|
| Microcontrolador | ✗ | ✗ | | | |
| App. Móvil | ✗ | ✗ | ✗ | ✗ | |
| App. *Web* | | | | ✗ | ✗ |

Cuadro 2.1: División de la funcionalidad del proyecto entre sus componentes.

## 2.3 Objetivos docentes

Además de los objetivos que se pretenden conseguir en el desarrollo de este TFG, se pueden describir también una serie de objetivos a cumplir en un plano más personal, relacionado con los conocimientos adquiridos a lo largo del grado en Ingeniería Informática y desarrollados en la consecución de este proyecto.

1) Aprender a diseñar un dispositivo *hardware*, basado en la plataforma Arduino. Lograr realizar la conexión del dispositivo con sensores de distinta índole, para procesar de una forma básica los datos que se captan por los sensores. También se incluye el diseño de una PCB con todas las conexiones necesarias para interconectar los sensores que se van a usar en el proyecto y el microcontrolador.
2) Desarrollar una aplicación móvil, para dispositivos con sistema operativo Android. Que la aplicación sea capaz de conectar un dispositivo *hardware* con el dispositivo móvil en el que está instalada por medio de una red de área personal basada en la tecnología *Bluetooth*.
3) Recibir datos de APIs externas para contrastar los datos recibidos por los sensores y aumentar así la fiabilidad de los mismos.
4) Ser capaz de realizar un procesamiento de los datos captados por los sensores y recibidos de APIs externas para detectar posibles riesgos en las rutas de senderismo.

5) Desarrollar una plataforma web siguiendo una arquitectura cliente-servidor, usando tecnologías en auge como son React[1] y NodeJS[2].

6) Aprender a gestionar los tiempos en un proyecto de larga duración, a preceder unas tareas ante otras asignando prioridades y a seguir una metodología de desarrollo ágil a lo largo del proyecto.

[1] https://reactjs.org
[2] https://nodejs.org

Capítulo 3

# Método de trabajo

En este capítulo se presentará la arquitectura del proyecto, qué componentes posee y cómo se han implementado esos componentes, indicando qué alternativas se han barajado y cómo se han solucionado los problemas que se han presentado a lo largo del desarrollo del proyecto. También se va a detallar la metodología que se ha empleado para el desarrollo del presente TFG, así como dar una explicación del por qué se decidió usar dicha metodología. Se usará este capítulo para discutir también los medios *software* y *hardware* usados durante el proyecto.

Por último se explicarán los recursos necesarios para la consecución del proyecto y se indicará cuál es el coste de estos recursos.

## 3.1 Arquitectura

En esta sección se va a presentar la arquitectura del sistema propuesto. `GVIDI`[1] es el nombre que toma el presente proyecto, cuyo objetivo principal es la monitorización de forma inteligente de los grupos de expedición, para detectar posibles situaciones de riesgo y combatirlas antes de que se produzcan, avisando de las mismas al guía de la expedición. Además, también se encarga de detectar situaciones de posible peligro físico para los senderistas (como pueden ser las caídas) para minimizar el posible impacto de las mismas priorizando una rápida ayuda al accidentado.

En esta sección, por tanto, se mostrarán los componentes que conforman la arquitectura y se desarrollarán las peculiaridades de cada uno de ellos, indicando qué problemática resuelven, por qué se ha implementado el componente de esa forma y qué otras alternativas se barajaron. También se hará especial hincapié en los problemas más importantes durante el diseño y la implementación y cómo se han solventado.

### 3.1.1 Visión general de la arquitectura

En la Figura 3.1 se puede observar el esquema general del proyecto completo, con un alto nivel de abstracción. A lo largo de esta sección, se desarrollarán en profundidad cada uno

[1] `GVIDI` significa guía en Esperanto, la lengua planificada internacional más hablada en el mundo. El proyecto toma este nombre porque sirve como ayuda en las expediciones.

de los componentes que conforman la arquitectura. En dicha imagen se pueden ver los tres componentes principales que se han desarrollado a lo largo del proyecto:

- **Microcontrolador**. Se trata de un dispositivo `Arduino` equipado con una serie de sensores que toman datos del entorno en el que se encuentra del portador del mismo.
- **Aplicación móvil**. Recibe los datos del microcontrolador a través de una conexión inalámbrica *Bluetooth* y realiza un procesamiento de los mismos para identificar situaciones de riesgo que serán notificadas al guía de la expedición.
- **Plataforma *web***. Permite la visualización tanto en tiempo real como de manera histórica de las expediciones que ha realizado un determinado senderista, con estadísticas detalladas acerca de las mismas.

Además, se puede observar también el uso de `Firebase`[2], una plataforma para aplicaciones (tanto *webs* como móviles) que está integrada dentro de la nube de `Google` y proporciona, entre otros, un servicio de autenticación de usuarios (*Firebase Auth*) usando únicamente código en el lado del cliente. Además, es posible autenticarse usando cuentas existentes en otras plataformas como pueden ser *Facebook*, *GitHub* o *Google*.

Para lograr una monitorización de la actividad en los grupos de expedición es necesario tomar datos del entorno de los participantes en la expedición, con el fin de obtener una vista del marco en el que se encuentran los participantes en todo momento. El microcontrolador se encarga principalmente de tomar los datos relevantes a partir de una serie de sensores que equipa para maximizar la seguridad del entorno del senderista, y enviarlos a la aplicación móvil para su procesamiento.

La monitorización inteligente es realizada por parte de la aplicación móvil, que toma los datos que llegan del microcontrolador a través de *Bluetooth* y, a partir de fuentes externas al microcontrolador, capta otros datos que también son relevantes para que el cálculo de ciertos riesgos sean más fiables.

Para que los senderistas puedan visualizar los resultados en tiempo real, se proporciona una interfaz gráfica en la aplicación móvil que posee la información más relevante a ser consultada durante la marcha. Si se quiere una mayor información (tanto en tiempo real como de forma histórica), se puede consultar la plataforma *web*.

De forma global, en `GVIDI` existen dos roles principales:

- **Guías**. Cada expedición posee un usuario con este rol y será la persona que actué como guía durante la expedición. Recibirá información detallada de todos los participantes durante la expedición, recibiendo alertas ante situaciones de peligro y posibles caídas de los participantes.

[2] `https://firebase.google.com/`

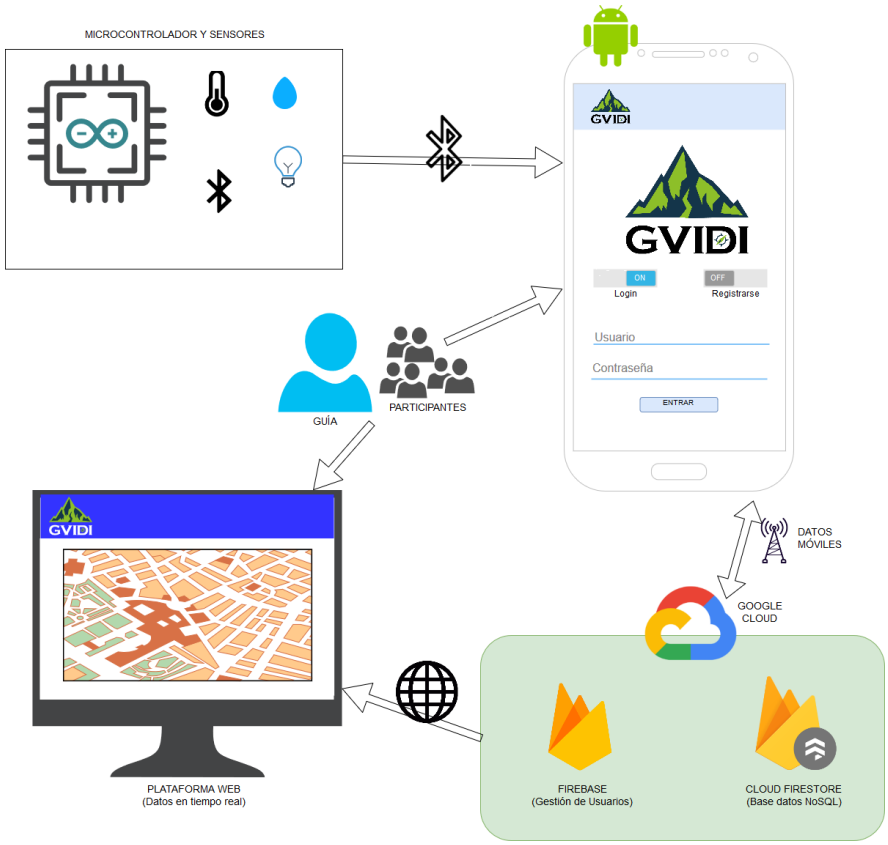



Figura 3.1: Esquema general de GVIDI.

- **Participantes**. Cada expedición puede poseer varias personas con este rol. Los participantes solo dispondrán de la información relacionada con ellos mismos. Los resultados de su monitorización serán enviados al guía de la expedición.

Los guías son los encargados de crear grupos de expedición, a los cuales se pueden unir los participantes. Una expedición está compuesta, por tanto, por un guía y un grupo de participantes. Cada integrante del grupo de expedición portará un microcontrolador en la muñeca, equipado con una serie de sensores que tomarán datos del entorno del integrante (datos de temperatura, humedad, condiciones lumínicas, lluvias, aceleración lineal y angular e información de posición). El microcontrolador se encargará de comunicar toda esta información captada a través de los sensores al dispositivo móvil del integrante, usando un módulo de comunicación *Bluetooth*. En la Tabla 3.1 se puede observar el precio de cada uno de los componentes que conforman el dispositivo *hardware*.

El dispositivo móvil tendrá instalada la aplicación GVIDI, desarrollada durante el presente TFG. Esta aplicación posee un sistema de autenticación, usando la plataforma *Firebase Auth*, anteriormente mencionada. El usuario se autenticará en la plataforma y será conducido a la pantalla principal en la que se ve información general sobre el usuario en la aplicación (esta pantalla es igual para todos los roles). Una vez en esta pantalla, las acciones que puede realizar un guía y las que puede realizar un participante de la expedición son distintas y serán

| Componente | Modelo | Cantidad | Precio por unidad |
|---|---|---|---|
| Microcontrolador | Arduino UNO | 3 | 24 € |
| Sensor GPS | NEO-6M | 3 | 11 € |
| Sensor pulso cardíaco | PulseSensor | 3 | 6 € |
| Acelerómetro y giroscopio | MPU-6050 | 3 | 1.5 € |
| Sensor de temperatura y humedad | DHT22 | 3 | 9.5 € |
| Detector de lluvia | YL-83 | 3 | 13 € |
| Fotoresistor Pack 10 | GL5528 | 3 | 3 € |
| Módulo *Bluetooth* | HC06 | 2 | 2.5 € |

Cuadro 3.1: Componentes que se han usado para construir el prototipo *hardware*.

expuestas más adelante. De igual forma, los guías serán capaces de crear y finalizar expediciones, mientras que los participantes sólo podrán buscar expediciones en curso y unirse a ellas (o abandonar una expedición a la que se encuentran unidos). Ambos roles pueden ver el estado de la expedición en un mapa, con la localización de todos los participantes de la misma.

La plataforma *web* también posee el sistema de autenticación usando *Firebase Auth*. En la plataforma *web*, los usuarios pueden ver tanto en tiempo real como a posteriori la expedición realizada (y un histórico de todas las expediciones que el usuario ha realizado, a lo largo del tiempo). La plataforma *web* proporciona información avanzada acerca de la expedición, siendo posible la visualización de estadísticas en un punto determinado de la misma. Gracias a la posibilidad de visualizar la expedición en tiempo real, un familiar o amigo del senderista puede hacer uso de la plataforma *web* para monitorizar el estado de este senderista en la expedición (ver su posición, su velocidad instantánea, la cantidad de distancia que ha recorrido y las alertas que ha generado, ya sea debido a caídas, debido a que ha superado la separación máxima con el guía en algún punto o porque ha solicitado realizar una parada).

Como se puede observar en la Figura 3.2, se puede dividir la arquitectura general en tres niveles. En el nivel más bajo se situaría el microcontrolador cuya función principal es la comunicación con los distintos sensores que conforman el dispositivo físico, para tomar datos que serán importantes en niveles posteriores. Este nivel es uno de los más importantes porque de él dependen los demás niveles. Se requiere que los datos captados sean lo más fieles a la realidad posibles, que el ruido en estos datos sea el menor posible y que se puedan enviar a una velocidad suficiente para que el análisis en etapas posteriores pueda ser realizado.

En el siguiente nivel se encuadran la aplicación móvil y la plataforma *web*. El objetivo de este nivel es el manejo de los datos que llegan del nivel previo. En este nivel se realiza un tratamiento y conversión de los datos a un formato para poder almacenarlos directamente o usarlos durante el análisis para producir resultados que serán almacenados más tarde. Los datos se transmiten por un protocolo inalámbrico desde el dispositivo físico a la aplicación móvil (en este proyecto se usa el protocolo *Bluetooth*), por lo que es posible que se produzcan

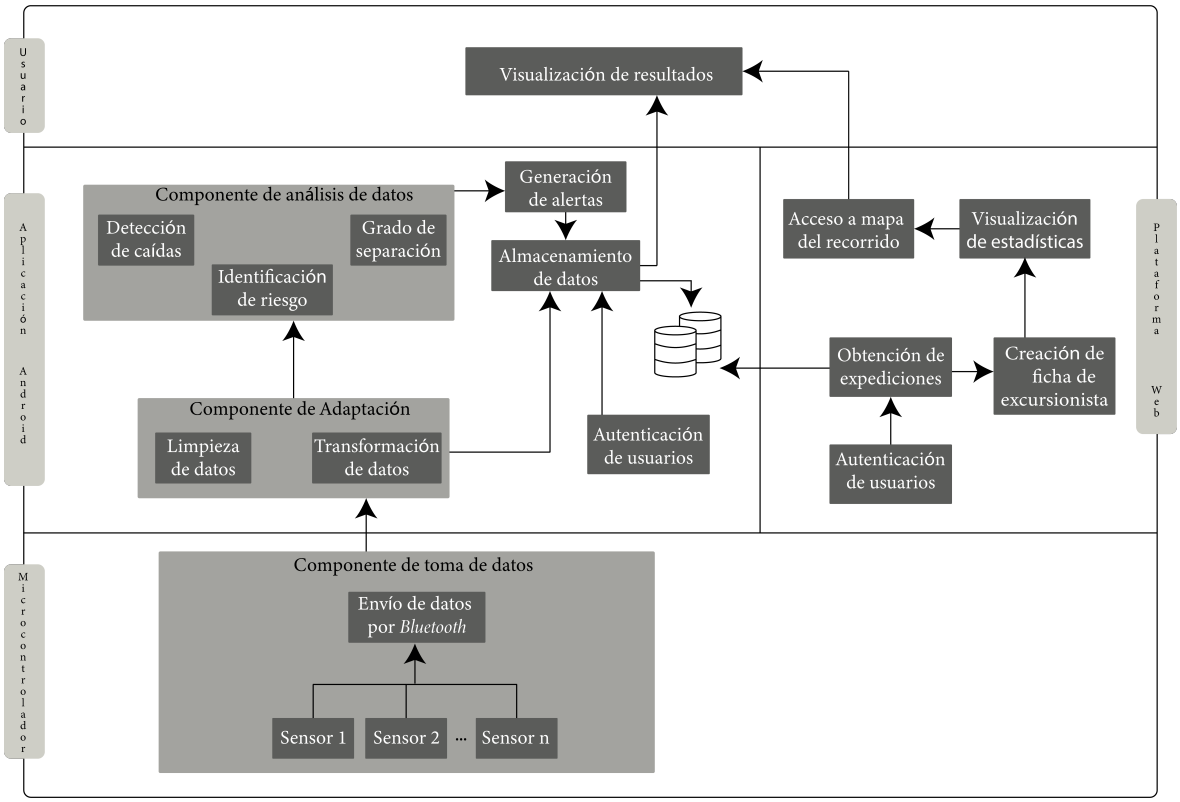



Figura 3.2: Arquitectura general, en la que se pueden observar los distintos componentes y tareas que conforman el proyecto.

pérdidas de paquetes y lleguen mensajes a la aplicación móvil incompletos o corruptos. Por esto, es necesario realizar una sanitización de los datos previa al tratamiento de estos datos. Además, los datos llegan con unidades distintas a las que requieren los algoritmos de análisis por lo que es necesario realizar un proceso de transformación de datos con el objetivo de que puedan ser usados por dichos algoritmos y puedan ser almacenados en un formato inteligible por los humanos.

Una vez que los datos han sido correctamente limpiados y transformados, éstos sirven como entrada para los algoritmos de análisis de datos. Estos algoritmos son capaces de inferir si un participante (o guía) de la expedición ha sufrido una caída, determinar el grado de separación del grupo de expedición e identificar el grado de riesgo de sufrir una caída que existe en la expedición en un momento dado, dadas las condiciones ambientales. Las salidas de estos algoritmos sirven para generar alertas si procede, o almacenar las salidas para su posterior uso o visualización.

En la plataforma *web*, los usuarios de `GVIDI` podrán acceder al histórico de expediciones en las que el usuario ha participado para ver las estadísticas detalladas de cada una de ellas. También es posible visualizar un mapa con todo el recorrido que el usuario ha realizado y con indicaciones en el mismo acerca del lugar en el que se produjeron las distintas alertas (en el caso de haber alguna). Las alertas también pueden proporcionar información adicional

del estado de la expedición en ese momento. Este histórico de expediciones engloba también la expedición que el usuario se encontrase haciendo en el momento (sólo cuando está realizando la expedición), de forma que se puede acceder a toda la información mencionada anteriormente en tiempo real.

Por último se puede observar el tercer y último nivel de la arquitectura general, que se centra en mostrar información relevante a los usuarios de `GVIDI`. Esta información se puede mostrar de dos formas distintas, a través de una interfaz de usuario en la aplicación móvil o a través de una página *web*. La primera forma está pensada para que los usuarios de la expedición puedan visualizar durante la expedición información útil para ellos. Por tanto, en la aplicación móvil se tiende a mostrar información en tiempo real y no datos generales de la expedición (el guía tiene una visión más general de la expedición pudiendo ver las alertas generadas en un periodo de tiempo, entre otras opciones). La interfaz de la *web* está pensada para su uso tanto por parte de los usuarios de la ruta como por familiares o amigos de la misma (personas autorizadas por el usuario). En la *web* se pueden visualizar tanto las estadísticas generales de la expedición (en curso o pasada) como la información específica en un punto de la expedición seleccionado.

La arquitectura del presente proyecto no se basa en ningún otro proyecto o aplicación existente, ya que no existen en el mercado alternativas que cumplan con los objetivos de `GVIDI`. Por tanto, la arquitectura se ha diseñado especialmente para resolver la problemática existente en la monitorización de la seguridad dentro de los grupos de expedición.

En las próximas secciones se describirá con un mayor detalle cada uno de los componentes que forman parte de la arquitectura, así como aspectos del diseño de dichos componentes que se consideren necesarios.

## 3.2 Microcontrolador *Arduino*

A lo largo de esta sección se presentará con detalle el diseño del microcontrolador `Arduino` usado en este proyecto. Con este dispositivo se pretende realizar la captación de datos a partir de los sensores que se encuentran equipados en el mismo. En la Tabla 3.2 se pueden observar los sensores que estarán conectados en el microcontrolador, junto con una breve explicación de para qué se usan los datos que estos sensores captan.

Para que el dispositivo sea capaz de captar datos de los sensores es necesario implementar la funcionalidad de captación por medio de código fuente usando el lenguaje de programación `Arduino`, basado en los lenguajes `C` y `C++`. A lo largo de esta sección se explicarán los detalles de implementación que se consideren relevantes.

### 3.2.1 Diseño del dispositivo *hardware*

Para el desarrollo de este proyecto se ha usado la placa `Arduino UNO Rev3`. La placa tiene conectados los siguientes componentes (ver esquema de conexión en la Figura 3.3):

| Magnitud a medir | Propósito de uso del sensor | Modelo utilizado |
|---|---|---|
| Aceleración y velocidad angular | Detección de caídas | `MPU6050` |
| Temperatura | Realización del análisis del riesgo de caída en la expedición | `DHT22` |
| Humedad | Realización del análisis del riesgo de caída en la expedición | `DHT22` |
| | Comunicación por *Bluetooth* con el dispositivo móvil | `HC-06` |
| Lluvia | Realización del análisis del riesgo de caída en la expedición | `YL-83` |
| Luz | Realización del análisis del riesgo de caída en la expedición | `GL5528` |
| Posición | Localización del participante dentro del grupo de expedición | `NEO-6M` |

Cuadro 3.2: Magnitudes y sensores que se usan en este proyecto.

- Sensor de temperatura y humedad (DHT22)
- Acelerómetro y giroscopio (MPU6050)
- Sensor de luminosidad (GL5528)
- Sensor de lluvia (YL-83)
- Módulo de *Bluetooth* (HC-06)
- Sensor de GPS (NEO-6M)
- Dos resistencias de 10KΩ
- Dos botones
- Led de color

**Uso de la memoria EEPROM**

El microcontrolador `Arduino UNO` no posee una memoria secundaria como puede tener un ordenador convencional, ni tampoco tiene la posibilidad de introducir una tarjeta *Secure Digital* (SD) (aunque existen módulos para incorporar esta funcionalidad) como tiene una `Raspberry Pi`. Por esto, cada vez que el microcontrolador se reinicia, el código que tiene cargado empieza a ejecutarse de nuevo y empezaría otra vez el proceso de calibración del acelerómetro y el giroscopio (explicado en detalle en la sección 3.2.2).

El mayor inconveniente se presenta si, por ejemplo, se reinicia el microcontrolador durante la expedición. Habría que situar el dispositivo en una superficie plana (que puede ser inexistente en ese momento) y esperar un cierto tiempo para realizar la calibración del acelerómetro y giroscopio, por lo que el grupo tendería a dispersarse. Para evitar esta situación se ha hecho uso de la memoria EEPROM disponible dentro del procesador.

Se han almacenado en memoria los datos relativos a la calibración del módulo `MPU6050` (*offsets* necesarios para calibrar la aceleración en los ejes X, Y, Z y *offsets* necesarios para calibrar la velocidad angular en los ejes X, Y, Z), de tal forma que solo hay que realizar la

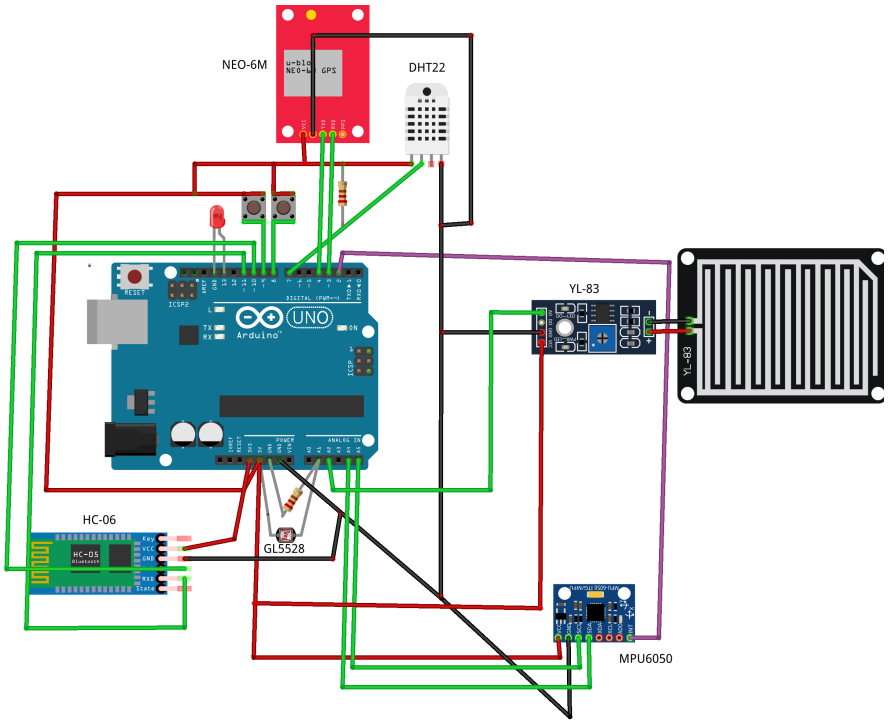



Figura 3.3: Conexiones de todos los componenetes en el microcontrolador `Arduino`.

calibración la primera vez que se usa el dispositivo (también existe la posibilidad de forzar el borrado de la memoria EEPROM dejando pulsado un botón en el dispositivo durante más de cinco segundos, como se puede observar en el código del Listado 3.1). Se usan un total de 24 *bytes* de memoria EEPROM para este cometido (los correspondientes a 6 números en punto flotante), por lo que se trata de una solución óptima a este problema.

Otras soluciones posibles serían eliminar la posibilidad de almacenar estos *offsets* y dejar que se pudieran dar situaciones potenciales de inusabilidad del dispositivo en plena ruta, o añadir un módulo externo al microcontrolador con una tarjeta SD, que implica un menor rendimiento del dispositivo (se emplea mucho tiempo manejando ficheros y el acceso a la tarjeta SD es mucho más lento si se compara con el acceso al procesador). Además la cantidad de datos que se almacenan es muy pequeña y no es necesario un gran tamaño de almacenamiento secundario.

Listado 3.1: **Borrado de la memoria EEPROM cuando se deja pulsado un botón durante más de 5 segundos.**

```
// If the button has been pressed for more than 5 secs, erase EEPROM memory
// and reset the arduino
if(millis() - reset_arduino_pressed_time >= 5000)
{
    eraseEEPROM();
    resetFunction();
}
```

Esta memoria tiene un tamaño total de 1024 *bytes* de almacenamiento y tiene unos 100.000 ciclos de lectura/escritura. Esto quiere decir que se puede escribir en la memoria EEPROM unas 100.000 veces hasta que esta se vuelva inestable. Esta cantidad de ciclos es más que suficiente para nuestro cometido, ya que simplemente se accederá a la memoria EEPROM cada vez que el dispositivo se inicia.

### 3.2.2 Toma de datos

Para hacer uso de los distintos sensores se han utilizado librerías programadas en el lenguaje C++. Para algunos sensores se han tomado librerías de código libre, adaptando o añadiendo algunas funciones de las que se hace uso en el proyecto, mientras que para usar otros sensores se ha desarrollado una librería básica. Por ejemplo, la librería usada para tomar datos del sensor de temperatura y humedad DHT22[3] ha sido modificada para añadir una función que devuelve un mensaje de error dependiendo de un código numérico (ver Listado 3.2). Esta función fue muy usada durante la primera etapa del proyecto, con un motivo principal de depuración.

Listado 3.2: **Ejemplo de funcionalidad añadida a librería de código libre del sensor DHT22.**

```
void DHT22::getErrorString(DHT22_ERROR_t& errorCode, char* errorString)
{
  switch (errorCode) {
    case DHT_ERROR_CHECKSUM:
      sprintf(errorString, "Error in checksum\n");
      break;
    case DHT_ERROR_TOOQUICK:
      sprintf(errorString, "Wait 2 seconds between each call to readData\n");
      break;
    case DHT_ERROR_NOT_PRESENT:
      sprintf(errorString, "ACK Not Present\n");
      break;
    case DHT_ERROR_ACK_TOO_LONG:
      sprintf(errorString, "ACK too long\n");
      break;
    case DHT_ERROR_DATA_TIMEOUT:
      sprintf(errorString, "Data timeout\n");
      break;
    case DHT_ERROR_SYNC_TIMEOUT:
      sprintf(errorString, "Sync timeout\n");
      break;
  }
```

[3] https://platformio.org/lib/show/115/DHT22/

### Temperatura y humedad, DHT22

Haciendo uso de la librería indicada anteriormente, la lectura de valores del sensor DHT22 es muy sencilla. Hay que tener en cuenta que el período de sondeo de este sensor es de dos segundos, es decir, realiza una nueva medición de datos de temperatura y humedad cada dos segundos. Si se intentan consultar los datos de temperatura y humedad en un tiempo menor de dos segundos después de la anterior consulta, se devolverá un error.

Figura 3.4: Imagen del sensor DHT22 [4]

Listado 3.3: **Ejemplo de toma de datos del sensor DHT22.**

```
#include <DHT22.h>
DHT22 dht22Sensor(DHT22_PIN);
void loop()
{
    DHT22_ERROR_t errorCode;
    // To avoid reading errors
    delay(2000);
    errorCode = dht22Sensor.readData();
    if (errorCode == DHT_ERROR_NONE)
    {
        float temperature = dht22Sensor.getTemperatureC();
        float humidity = dht22Sensor.getHumidity();
    }
}
```

| | Valor máximo | Valor mínimo | Precisión | Periodo de sondeo |
|---|---|---|---|---|
| **Temperatura** | 80ºC | -40ºC | ±0,5ºC | 2s |
| **Humedad** | 100 %RH[5] | 0 %RH | ±2 %RH | 2s |

Cuadro 3.3: Características principales del sensor DHT22.

### Acelerómetro y girsocopio, MPU6050

El sensor más complejo es el MPU6050 (acelerómetro y giroscopio). Para utilizar este sensor, se ha hecho uso de una librería de código libre[6]. Sin embargo, para que las medidas del MPU6050 sean fiables y se elimine el ruido en las mismas es necesario calibrar el sensor. Con este módulo es muy posible que exista un desnivel en sus componentes (ver Figura 3.6). Para eliminar esta desviación en las componentes del módulo y asegurar que no existen desniveles agregados se añaden *offsets*. A cada componente (aceleración en los ejes $X$, $Y$, $Z$ y velocidad angular en los ejes $X$, $Y$, $Z$) se le añadirá un determinado valor, de forma que el sensor en

[4] Imagen extraída de http://electr3s.com/589/sensor-dht22-digital-de-temperatura-y-humedad.jpg.jpg

[6] https://github.com/jrowberg/i2cdevlib/tree/master/Arduino/MPU6050

reposo muestre los valores de la Tabla 3.4.

El módulo en reposo debería mostrar una aceleración neta $|\mathbf{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2} = 1g = 9{,}8m/s^2$ y una velocidad angular neta de $0$ $deg/s$. El valor $16384$ en uno de los ejes indica un valor de $1g$, por lo que para convertir el valor digital que se obtiene del sensor a unidades de $m/s^2$, es necesario aplicar la fórmula:

Figura 3.5: Imagen del sensor `MPU6050` [7]

$$\text{Valor en } m/s^2 = \text{valor digital} \cdot (9{,}81/16384) \quad (3.1)$$

| | | Valores digitales de salida del sensor | | |
|---|---|---|---|---|
| | | **En reposo** | **Máximo** | **Mínimo** |
| **Aceleración** | **Eje X** | $16384 \pm 8 = 1g$ | $32767 = 2g$ | -32768 $= -2g$ |
| | **Eje Y** | $0 \pm 8 = 0g$ | $32767 = 2g$ | -32768 $= -2g$ |
| | **Eje Z** | $0 \pm 8 = 0g$ | $32767 = 2g$ | -32768 $= -2g$ |
| **Velocidad angular** | **Eje X** | $0 \pm 1 = 0deg/s$ | $32767 = 250deg/s$ | -32768 $= -250deg/s$ |
| | **Eje Y** | $0 \pm 1 = 0deg/s$ | $32767 = 250deg/s$ | -32768 $= -250deg/s$ |
| | **Eje Z** | $0 \pm 1 = 0deg/s$ | $32767 = 250deg/s$ | -32768 $= -250deg/s$ |

Cuadro 3.4: Valores digitales que mide el sensor `MPU6050` para una sensibilidad de $\pm 2g$ en la aceleración y de $\pm 250$ $deg/s$ en la velocidad angular.

**Calibración del módulo `MPU6050`.** La calibración es un proceso por el cual se toman un gran número de medidas del módulo, se descartan las primeras medidas (un total de 100 en el caso de este proyecto) para evitar ruido y se calcula el valor medio entre las medidas que se han tomado. Con este valor medio calculado se proceden a calcular los valores de *offset* (véase Listado 3.4), que serán aplicados al sensor. Este proceso se repetirá hasta que las medias de las medidas en todas las componentes del módulo estén dentro de un determinado umbral, próximo a los valores mostrados en la Tabla 3.4. Se establece un umbral debido a que es posible que los valores de las medias de las medidas nunca llegen a converger hasta llegar a los valores exactos mostrados en la tabla. Los umbrales que se ha decidido tomar por defecto son $\pm 8$ para la aceleración y $\pm 1$ para la velocidad angular.

Listado 3.4: **Cálculo de los *offsets* (en este ejemplo se han sustituido las constantes por números para un mejor entendimiento).**

[7] Imagen extraída de
`https://naylampmechatronics.com/763-large_default/modulo-mpu6050-acelerometro-giroscopio-i2c.jpg`

```
ax_offset = (16384 - mean_ax) / 8;
ay_offset = - mean_ay / 8;
az_offset = - mean_az / 8;

gx_offset = -mean_gx / 4;
gy_offset = -mean_gy / 4;
gz_offset = -mean_gz / 4;
while (1){
  int ready = 0;
  mpu6050_sensor.setXAccelOffset(ax_offset);
  mpu6050_sensor.setYAccelOffset(ay_offset);
  mpu6050_sensor.setZAccelOffset(az_offset);

  mpu6050_sensor.setXGyroOffset(gx_offset);
  mpu6050_sensor.setYGyroOffset(gy_offset);
  mpu6050_sensor.setZGyroOffset(gz_offset);

  TakeAMeasurement();

  // Check if all the offsets are ok. If only one of the offset is not ok,
  // correct the offset and take measures again
  if ((abs(16384 - mean_ax)) <= 8) ready++;
  else ax_offset = ax_offset + (16384 - mean_ax) / 8;

  if (abs(mean_ay) <= 8) ready++;
  else ay_offset = ay_offset - mean_ay / 8;

  if (abs(mean_az) <= 8) ready++;
  else az_offset = az_offset - mean_az / 8;

  if (abs(mean_gx) <= 1) ready++;
  else gx_offset = gx_offset - mean_gx / (1 + 1);

  if (abs(mean_gy) <= 1) ready++;
  else gy_offset = gy_offset - mean_gy / (1 + 1);

  if (abs(mean_gz) <= 1) ready++;
  else gz_offset = gz_offset - mean_gz / (1 + 1);

  if (ready == 6) break;
}
```

La calibración es un proceso que necesita realizarse con el microcontrolador totalmente en reposo, sobre una superficie lo más plana posible. Este proceso se realiza nada mas conectar el microcontrolador y puede llevar un tiempo arbitrario que puede llegar a ser bastante elevado (durante el desarrollo del proyecto se han podido observar valores que oscilaban entre

los 20 segundos a los 5 minutos). El motivo por el cual el proceso de calibración oscila entre unos valores tan dispares es el proceso de cálculo de *offsets*, que se puede observar en el código del Listado 3.4. Como se puede observar, este cálculo depende del *offset* calculado anteriormente y de la media de un cierto número de medidas que se han realizado con el sensor. Si el sensor no se encuentra en completo reposo (o sobre una superficie lo suficientemente plana) la media de los valores de aceleración no será todo lo próxima necesaria al valor de 1g (o $9{,}8m/s^2$). Por ejemplo, mientras la media de los valores de la acelaración en el eje Z no esté en el intervalo [-8, 8], el *offset* para la aceleración en este eje no será correcto y el proceso de calibración no finalizará. El sensor también puede realizar alguna medida errónea en un momento dado (lo que consideraríamos ruido), que afecta negativamente al proceso de calibración. Las medidas erróneas por parte del sensor no tienen una solución sencilla por lo que se asume que el sensor no realizará mediciones incorrectas. En el caso de realizarlas, el algoritmo anterior se encarga de gestionarlas ya que no finalizará hasta que el módulo esté completamente calibrado.

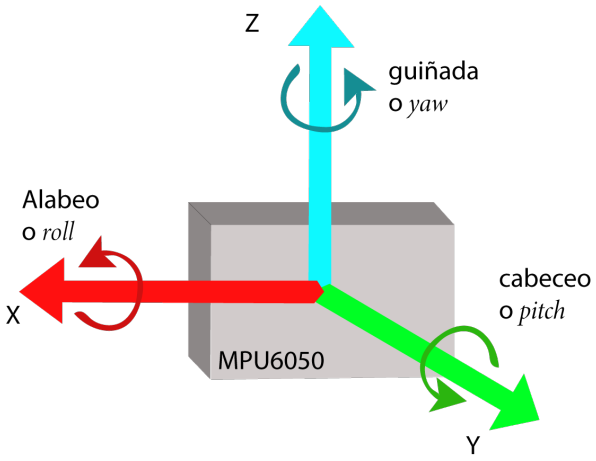



Figura 3.6: Componentes del acelerómetro y giroscopio `MPU6050`.

Una vez realizada la calibración del módulo `MPU6050`, ya puede ser usado de forma habitual. En este proyecto simplemente se solicitan los valores actuales de aceleración y velocidad angular, que serán utilizados en el análisis inteligente de los datos.

### Sensor de lluvia, `YL-83`

El uso del sensor de lluvia es muy simple. Este sensor está conectado a una entrada analógica y devuelve un valor de voltaje que va desde 0 para una placa completamente mojada hasta 1023 para una placa completamente seca. Este sensor no posee la precisión necesaria para medir la cantidad de agua acumulada, por lo que tan sólo es útil para medir la presencia o ausencia de agua. En nuestro caso la información de si está lloviendo o no es suficiente para ser usada posteriormente en el análisis del riesgo de caída en la expedición.

[8] Imagen extraída de `https://images-na.ssl-images-amazon.com/images/I/51SeX9Bk7kL._SX342_.jpg`

Figura 3.7: Imagen del sensor de lluvia `YL-83` [8]

Para hacer uso de este sensor se ha creado una librería realmente simple, que lee el valor analógico proporcionado por el sensor y devuelve un valor que indica la cantidad aproximada de lluvia que hay. La Tabla 3.5 muestra la salida de la librería para los distintos valores obtenidos del sensor.

| | **No hay lluvia** | **Lluvia suave** | **Lluvia moderada** | **Lluvia fuerte** |
|---|---|---|---|---|
| **Intervalo de valores** | $[500, 1023]$ | $(300, 500)$ | $[300, 100)$ | $[0, 100]$ |

Cuadro 3.5: Intervalos de voltaje que el sensor `YL-83` proporciona y su relación con la cantidad aproximada de lluvia.

### Sensor GPS, `NEO-6M`

Para hacer uso de este sensor se ha usado la librería `TinyGPS`[10]. La librería no es necesaria para tomar datos del sensor pero es una gran ayuda para identificar los datos que se obtienen del sensor y mostrarlos de forma entendible. Este sensor posee una antena cerámica para captar la información de los satélites y se conecta al microcontrolador a través de una conexión serial (usando dos pines, uno para transmisión y otro para recepción). Los pines usados para la conexión serial del módulo de GPS son los pines 3 y 4, como se puede ver en el esquema de conexión de la Figura 3.3.

Figura 3.8: Imagen del sensor `NEO-6M` [9]

Los datos que se obtienen de la lectura serial del sensor están en formato NMEA (véase Listado 3.6). Existen varios mensajes GPS NMEA, como $GPGGA que proporciona datos de localización y precisión, $GPGSV que proporciona información de satélites o $GPGLL que proporciona datos de latitud y longitud geográfica, entre otros. En este punto es donde la librería `TinyGPS` entra en juego, facilitando la tarea de entendimiento de los mensajes NMEA. En el Listado 3.5 se puede observar la sencillez para obtener la posición actual a partir de las sentencias NMEA usando esta librería.

Listado 3.5: **Lectura de datos del módulo GPS usando una comunicación serial.**

[9] Imagen extraída de `https://rukminim1.flixcart.com/image/832/832/joonafk0/electronic-hobby-kit/d/z/6/neo6m-gps-module-flight-controller-arduino-original-imafb2yyz9xudsfj.jpeg?q=70`

[10] `http://arduiniana.org/libraries/tinygps/`

```
// This function reads data from the GPS sensor
void readGPSData(float* latitude, float* longitude)
{
  while (ss_gps.available())
  {
    char c = ss_gps.read();   // Serial read to get the gps data
    if (tinyGPSInstance.encode(c))   // Did a new valid sentence come in?
    {
      tinyGPSInstance.f_get_position(latitude, longitude);
    }
  }
}
```

Listado 3.6: **Ejemplo de valores en formato NMEA.**

```
$GPGGA,110617.00,41XX.XXXXX,N,00831.54761,W,1,05,2.68,129.0,M,50.1,M,,*42
$GPGSA,A,3,06,09,30,07,23,,,,,,,,4.43,2.68,3.53*02
$GPGSV,3,1,11,02,48,298,24,03,05,101,24,05,17,292,20,06,71,227,30*7C
$GPGSV,3,2,11,07,47,138,33,09,64,044,28,17,01,199,,19,13,214,*7C
$GPGSV,3,3,11,23,29,054,29,29,01,335,,30,29,167,33*4E
$GPGLL,41XX.XXXXX,N,00831.54761,W,110617.00,A,A*70
$GPRMC,110618.00,A,41XX.XXXXX,N,00831.54753,W,0.078,,030118,,,A*6A
$GPVTG,,T,,M,0.043,N,0.080,K,A*2C
```

**Problema en el uso del sensor de GPS.** Ha habido un problema durante la integración del módulo de GPS relacionado con la captación de datos, que no ha hecho posible su uso. Al intentar tomar los datos de latitud y longitud del módulo, se obtenía un resultado igual que el que se obtendría si el módulo no pudiese comunicarse con ningún satélite. En un principio se pensó que era debido a una falta de cobertura, ya que es necesario esperar unos minutos con el módulo al aire libre para que pueda comenzar a tomar datos con normalidad. Después de dejar el prototipo al aire libre en distintas ocasiones y localizaciones (entre ellas Madrid, donde la captación de señal no debería ser un problema para el módulo) durante un período prolongado de tiempo, ha sido imposible captar señal GPS. Al inicio del proyecto se compraron tres módulos GPS, uno para cada uno de los prototipos que se van a crear durante el presente proyecto. Al recibirlos, uno de los módulos venía con signos evidentes de corrosión y con la antena desoldada de la placa por lo que se piensa que el motivo por el cual los dos módulos restantes no funcionan es debido a que están defectuosos.

Para resolver esta problemática se ha hecho uso del GPS del dispositivo móvil para tomar los datos relativos a la posición del guía y de los participantes en la expedición (se explicará en más detalle en la sección relacionada con la arquitectura de la aplicación móvil).

Sin embargo, el código fuente del microcontrolador contempla la toma de datos a través de un módulo GPS NEO-6M conectado al mismo (y el envío de los mismos al dispositivo móvil usando *Bluetooth*) y la PCB ha sido diseñada con las conexiones necesarias para integrar dicho módulo. Por esto, debería bastar con conectar un módulo GPS válido a la PCB y esperar a que el mismo capte señal para que la información de posición sea la dictada por el microcontrolador en vez de la dictada por el dispositivo móvil.

### 3.2.3 Creación de la PCB

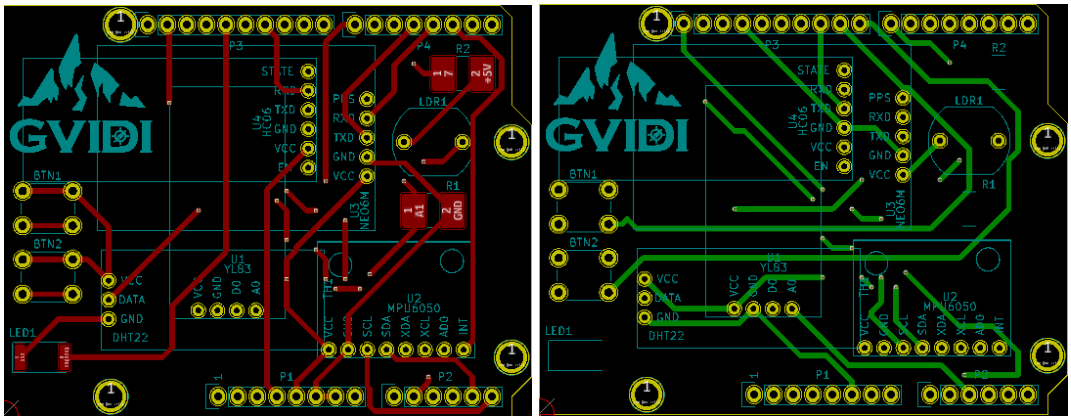
Durante la etapa inicial del proyecto, en la que se desarrolló el código fuente que usa el dispositivo para comunicarse con los distintos sensores que posee, se hizo uso de una placa *protoboard*, en la que se conectaban todos los sensores al microcontrolador por medio de cables de prototipado. De esta forma, era muy sencillo realizar cambios debidos a fallos en la conexión (si se conecta un sensor a un *pin* erróneo del microcontrolador) y añadir nuevos componentes (como es el caso de los botones y los leds) si eran requeridos en versiones posteriores del dispositivo.

Una vez se obtuvo una versión definitiva del dispositivo, se pensó en crear una PCB para mejorar la presentación del mismo. Usando una PCB, se eliminan los cables de prototipado y la placa *protoboard*, quedando un diseño mucho más profesional. Además, se elimina la posibilidad de que en plena ruta un usuario desconecte un cable del microcontrolador o de la placa *protoboard* y no sepa dónde conectarlo después. Esta situación podría dar lugar a que un sensor deje de funcionar o, en el peor de los casos, que un sensor resulte dañado si se conecta de forma errónea.

Para el diseño de la PCB se ha usado el *software* de código libre KiCad[11]. Una PCB básica posee un total de dos capas, la superior y la inferior. Ambas capas se pueden usar para el rutado entre los componentes de la misma y se comunican a través de vías, que son pequeños agujeros en la placa que comunican la capa superior con la capa inferior de la misma. En la Figura 3.9 se puede observar el diseño de ambas capas de la placa. En este diseño se encuentran los orificios necesarios para conectar la PCB con el microcontrolador, los orificios necesarios para conectar todos los sensores en la placa y además, las rutas entre el microcontrolador y los sensores. También se ha dotado al diseño de etiquetas para identificar los sensores, además del logo del proyecto para garantizar la propiedad del mismo.

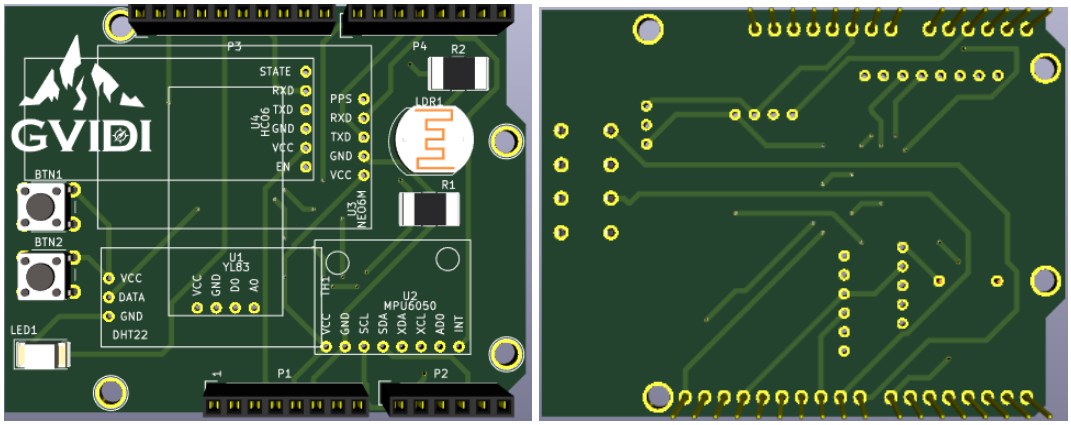
Una vez se ha completado el diseño de la PCB, es necesario crearla físicamente. La impresión de la PCB ha sido realizada por una empresa que se dedica a ello. El resultado final es el que se ve en la Figura 3.11. El ultimo paso es soldar en la placa todos los sensores y actuadores que se usan en este proyecto, además de los pines necesarios para conectar la placa en el microcontrolador. En la Figura 3.12 se puede observar la versión final del prototipo desarrollado durante este proyecto.

[11] http://kicad-pcb.org/



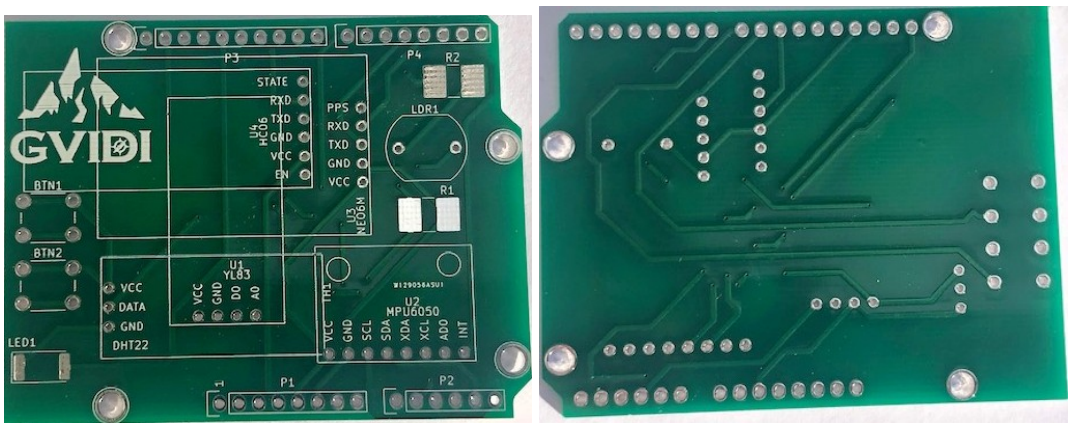

(a) Capa superior. (b) Capa inferior.

Figura 3.9: Diseño de la PCB.




(a) Capa superior. (b) Capa inferior.

Figura 3.10: Vista en 3D de la PCB.




(a) Capa superior. (b) Capa inferior.

Figura 3.11: Impresión final de la PCB.

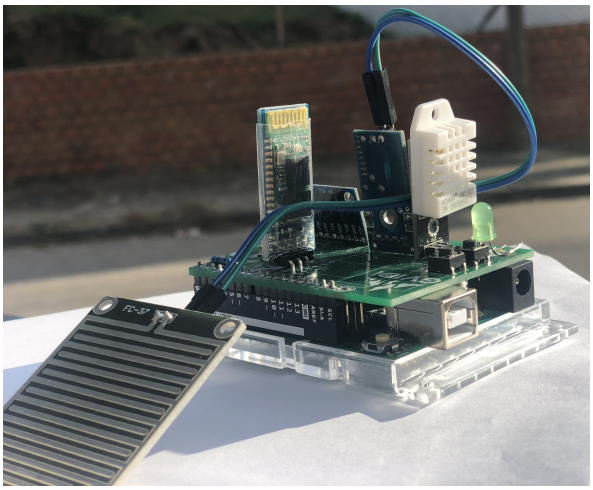


Figura 3.12: Prototipo final del microcontrolador conectado con todos los sensores que se usan.

## 3.3 Aplicación móvil

En esta sección se va a mostrar en detalle tanto el diseño como la implementación de la aplicación móvil. La aplicación móvil se encarga del tratamiento, el análisis, el almacenamiento y la visualización de los datos captados por el dispositivo físico. Se trata de la parte más compleja del proyecto por lo que se expondrán distintas alternativas que se han barajado a lo largo del mismo, y el por qué se han tomado o descartado dichas alternativas.

La aplicación móvil se ha implementado únicamente para el sistema operativo *Android*, ya que no se disponía de tiempo físico ni de recursos para implementar la aplicación móvil en otros sistemas operativos como *iOS*. Sin embargo esto no es un gran inconveniente ya que la mayoría de la población usa un dispositivo móvil *Android* (en junio del año 2018 la venta de *iPhones* suponía un total del $18{,}94\,\%$ de todos los dispositivos móviles vendidos).

### 3.3.1 Diseño de la aplicación móvil

Para el diseño de la aplicación móvil, en primer lugar, se realizaron bocetos de las distintas pantallas para tener una idea clara de la información relevante a mostrar. De esta forma la implementación se centra en conseguir la información que hay que mostrar y no se pierde tiempo implementando características que al final se desechan o que no son usadas.

La aplicación móvil se ha diseñado siguiendo el patrón de arquitectura de *software* MVC o patrón Modelo-Vista-Controlador (véase Figura 3.13). Este patrón permite realizar una separación entre la lógica de tratamiento de los datos, la interfaz de usuario y la lógica de control (o la lógica de aplicación).

- **Modelo**. Contiene la representación de los datos que gestiona el sistema y permite el acceso a sistemas de persistencia de datos, como pueden ser las bases de datos.
- **Vista**. La vista se trata de la interfaz de usuario, es decir, la parte del sistema que

permite al usuario interaccionar con el mismo de forma directa.

- **Controlador**. Este elemento actúa como intermediario entre la vista y el modelo. Recibe las peticiones por parte del usuario, solicita los datos al modelo y una vez que los tiene, los envía a la vista para que el usuario pueda visualizarlos.


Figura 3.13: Esquema general del patrón de arquitectura de *software* MVC.

### 3.3.2 Funcionalidad de la aplicación móvil

Como se ha comentado anteriormente, la aplicación móvil se ha implementado para el sistema operativo *Android*, usando `Java` como lenguaje de programación. La aplicación se compone de varias pantallas, siendo cada una de ellas una *Activity*. Una *Activity* en *Android* es un componente que contiene una interfaz o una pantalla con la cual los usuarios pueden interaccionar. Las actividades que conforman la aplicación son las siguientes:

- **Actividad *Signup***. Esta actividad se utiliza tanto para que los usuarios puedan registrarse en la plataforma, como para que los usuarios que ya están registrados en la misma puedan acceder a ella. Como se puede ver en la Figura 3.14b, tan solo es necesario un nombre de usuario, un correo electrónico y una contraseña para registrarse ya que dichos datos son los datos mínimos necesarios para realizar un registro en la plataforma.
- **Actividad *ResetPassword***. Esta actividad se utiliza como medio de recuperación de la contraseña de un usuario, si este no la recuerda. Tan solo necesita proporcionar el correo electrónico con el que se registró en la plataforma y le serán enviadas a dicho correo las instrucciones para restablecer la contraseña.
- **Actividad *JoinExpedition***. Esta actividad (ver Figura 3.14c) tan solo es visible si el usuario que se ha autenticado en el sistema tiene el rol de participante. En ella se muestran en una lista los nombres de todas las expediciones que se encuentran en curso en el momento, junto con el correo electrónico del guía de la expedición. Cuando el participante selecciona una expedición de la lista, se une a ella de forma automática.
- **Actividad *CreateExpedition***. Esta actividad tan solo es visible si el usuario que se ha autenticado en el sistema tiene el rol de guía. Gracias a la misma, los guías podrán

crear nuevas expediciones de manera muy sencilla, indicando tan sólo el nombre de la expedición (como se puede ver en la Figura 3.15b).

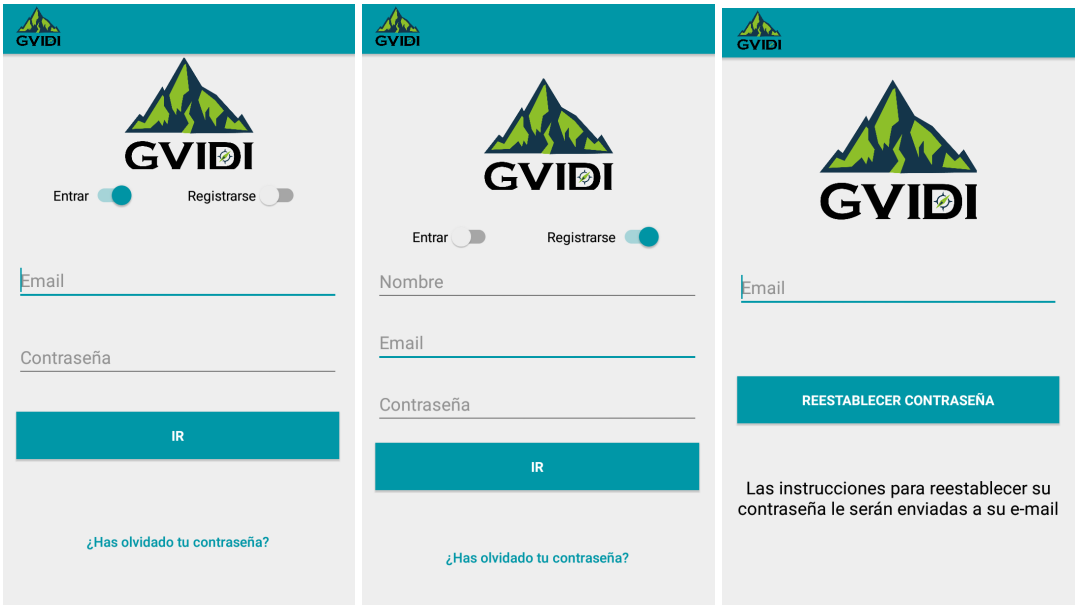
- **Actividad *ExpeditionDetails***. Esta actividad tan solo es visible si el usuario que se ha autenticado en el sistema tiene el rol de guía. En esta actividad los guías pueden ver detalles generales de la expedición como las alertas generadas en una franja de tiempo, los participantes que se encuentran a más de un cierto umbral de distancia, el número de participantes que han solicitado una parada desde la última parada realizada y el número de participantes que tienen menos de un cierto nivel de batería en sus terminales. En definitiva, gracias a esta actividad el guía puede tener una vista general de la expedición de una forma rápida.
- **Actividad *ParticipantDetails***. Al igual que las dos actividades anteriores, esta actividad es solo visible para el rol de guía. Esta actividad (ver Figura 3.16a) se muestra cuando el guía selecciona un determinado participante en el mapa de la vista principal y en ella se puede ver toda la información acerca del participante. Entre esta información destacan todas las alertas que ha generado el participante, su ritmo (tanto actual como el ritmo medio) y el nivel de batería que tiene el dispositivo móvil del participante.
- **Actividad principal**. Esta actividad es compartida tanto por los participantes como por los guías y muestra la información relevante para los usuarios (tanto participantes como guías) mientras la expedición está en curso. En esta vista podemos destacar el mapa en el cual se pueden visualizar todos los usuarios de la expedición. El guía se muestra en color azul y los participantes aparecerán en color verde, amarillo o rojo dependiendo de la distancia con respecto al guía.

### 3.3.3 Comunicación entre la aplicación móvil y el dispositivo físico

Como se ha mencionado en numerosas ocasiones anteriormente, los datos que envía el microcontrolador provenientes de los sensores que lleva equipados son enviados al dispositivo móvil a través de *Bluetooh*. Por esto es necesario que la aplicación móvil pueda dar soporte a esta comunicación, y para ello se ha implementado la solución que se puede observar en la Figura 3.17.

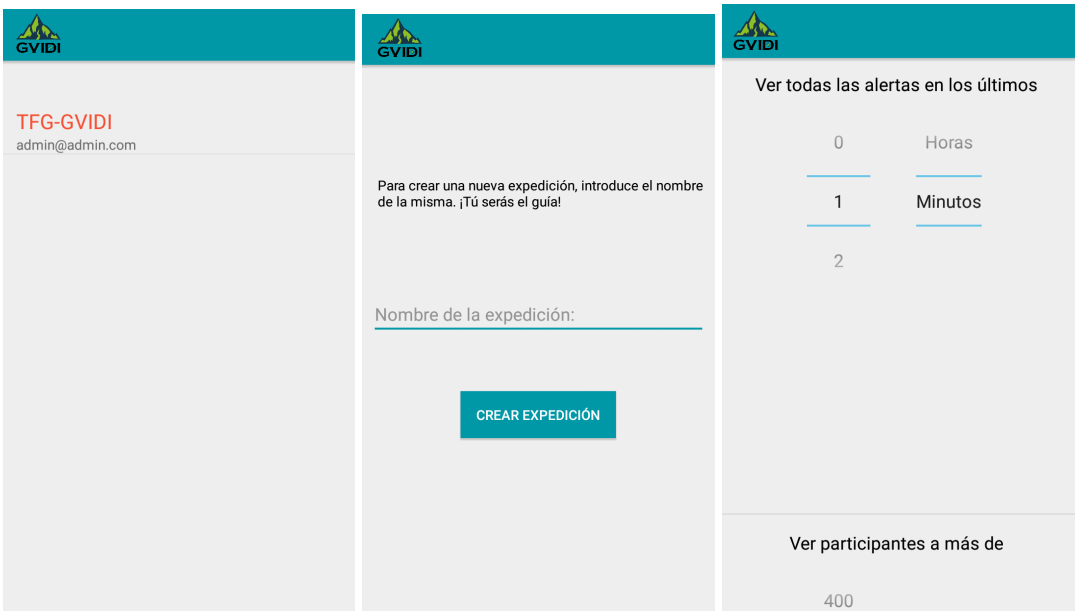
Para poder realizar una conexión usando el protocolo inalámbrico *Bluetooth* es necesario realizar una conexión usando *sockets* RFCOMM. Para crear un *socket* siempre es necesario proporcionar una tupla compuesta por una dirección y un número de puerto (como ocurre por ejemplo con los *sockets* `TCP/IP`). En los *sockets* RFCOMM, la dirección que hay que proporcionar al *socket* es la dirección MAC del dispositivo *bluetooth* con el que se quiere realizar la conexión. En este proyecto, no se usa un puerto específicamente definido y el número de puerto se gestiona de forma automática.

Para obtener la dirección MAC del dispositivo con el que se realizará la conexión *Bluetooth*



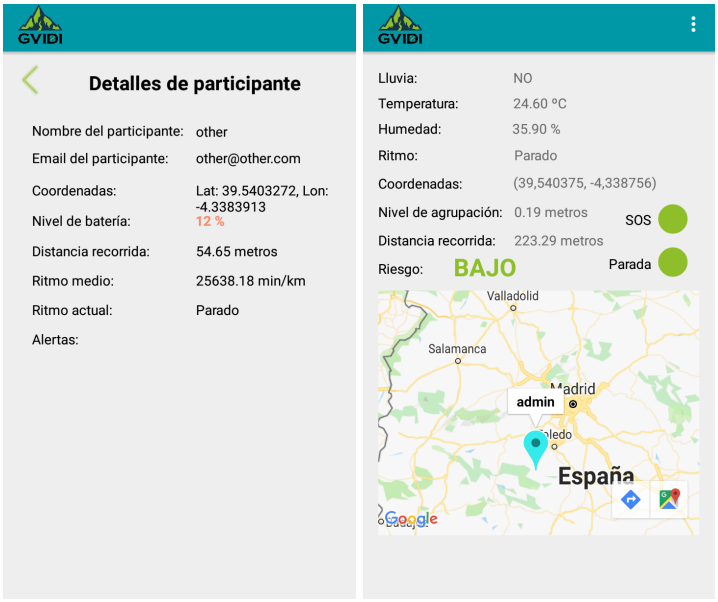

(a) Vista de *Login* (b) Vista de registro. (c) Vista de la actividad para reestablecer la contraseña.

Figura 3.14: Pantallas de inicio de sesión, registro y reestablecimiento de contraseña.




(a) Vista de la actividad para unirse a expediciones en curso. (b) Vista de la actividad para crear una nueva expedición. (c) Vista parcial de la actividad para visualizar el estado general de la expedición.

Figura 3.15: Pantallas de unión a expedición en curso, creación de expedición y vista del estado general de la expedición.




(a) Vista de la actividad de deta-(b) Vista de la actividad principal.
lles de un participante.

Figura 3.16: Pantalla de vista en detalle de un participante y pantalla principal.

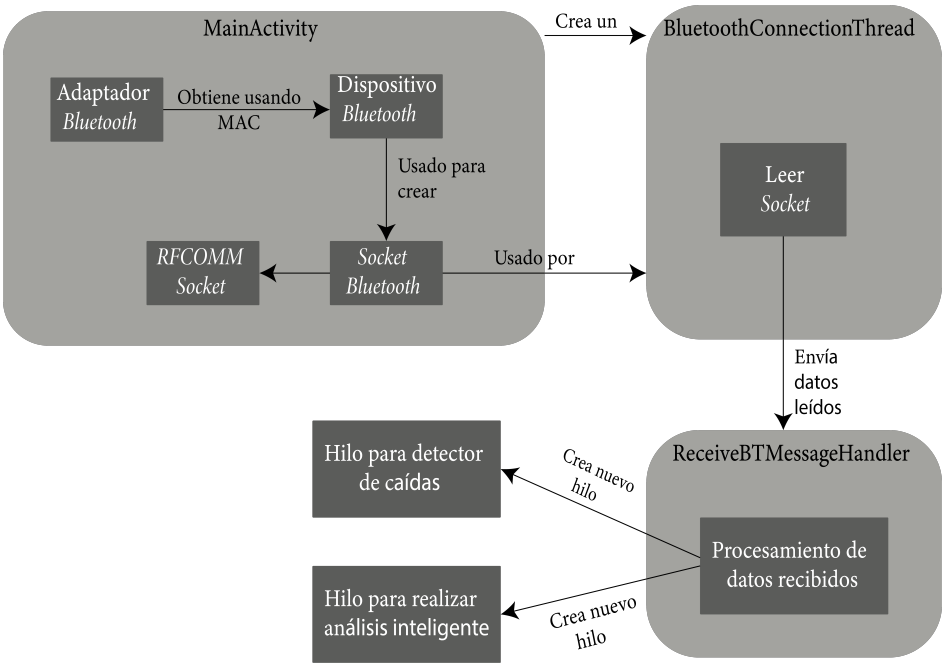



Figura 3.17: Arquitectura de la comunicación *Bluetooth* en el dispositivo móvil.

es necesario realizar una búsqueda de dispositivos. En este proyecto, la MAC se obtiene realizando una búsqueda de dispositivos *Bluetooth* en los dispositivos que han sido emparejados con el teléfono móvil (ver Listado 3.7). Una vez que se han recuperado los dispositivos emparejados se muestran al usuario para que pueda elegir el dispositivo *Bluetooth* con el que quiere realizar la conexión. Si esta conexión es fallida (porque el dispositivo que elige está

fuera del alcance o no se encuentra disponible en ese momento) el usuario tendrá que volver a elegir el dispositivo con el que se conectará.

Listado 3.7: **Búsqueda de dispositivos *Bluetooth* emparejados para obtener la dirección MAC.**

```
if (btAdapter.isEnabled() && macAddress == null) {
  // Buscar dispositivos emparejados
  Set<BluetoothDevice> pairedDevices = btAdapter.getBondedDevices();
  if (pairedDevices.size() > 0) {
    // Mostrar al usuario los dispositivos emparejados para que elija con el que
    // se quiere conectar
  }
}
```

Cuando se recupera el dispositivo *Bluetooth* con el que se va a realizar la conexión, se puede crear el *socket Bluetooth* que se ha comentado anteriormente, usando la dirección MAC del dispositivo recuperado. Si el *socket* se crea y se conecta con el dispositivo remoto de forma satisfactoria, se puede proceder a la recepción o envío de datos por medio de este *socket*. En este proyecto el dispositivo móvil sólo recibe los datos provenientes del microcontrolador y no envía nada a este último.

Para la recepción de datos, se ha creado un nuevo hilo, ya que el hilo principal de la aplicación se encarga de mostrar la información al usuario en la pantalla del teléfono y si la recepción de datos se realiza en este hilo, se bloqueará y el usuario no podrá usar la aplicación porque esta se encontrará "bloqueada" recibiendo los datos por *Bluetooth*. El nuevo hilo estará recibiendo los datos que llegan por el *socket Bluetooth* de forma continua.

Como es necesario que se reciban la mayor cantidad de datos posibles en un cierto periodo de tiempo (así lo requiere el algoritmo para detectar caídas, como se comentará más adelante), el código del hilo que lee los datos del *socket* debe ser lo más reducido posible, así leerá datos del *socket* más rápidamente. El cometido del hilo tan sólo es leer desde el *socket* y enviar los datos leídos a un manejador que los procesará mas a fondo (véase Listado 3.8).

Listado 3.8: **Hilo secundario que lee datos del *socket Bluetooth* y los envía a un manejador.**

```
while (true) {
  try {
    number_of_incoming_bytes = input_bytes.read(buffer);
    // Notificar al manejador ReceiveBTMessageHandler del nuevo mensaje que se recibe
    handler.obtainMessage(1, number_of_incoming_bytes, -1, buffer).sendToTarget();
  } catch (IOException e) {
    break;
```

```
8     }
9   }
```

Cuando el manejador de mensajes recibe los datos que se han leído del *socket*, se crean dos nuevos hilos, uno que se encargará del algoritmo para detectar caídas y otro que se encargará del resto del análisis inteligente (análisis del riesgo de caída en la expedición y análisis de distancias). Se crean dos hilos debido, principalmente, a que el algoritmo del detector de caídas necesita nuevos datos de aceleración (provenientes del acelerómetro instalado en el microcontrolador) lo más rápidamente posible y con la creación de un nuevo hilo cada vez que se leen nuevos datos del *socket Bluetooth* dedicado únicamente a proporcionar datos de aceleración al algoritmo de detección de caídas se consigue la máxima eficiencia de dicho algoritmo.

Merece la pena comentar que la creación de nuevos hilos cada vez que se leen nuevos datos del *socket* no proporciona solo ventajas, sino también inconvenientes a tener muy en cuenta. La funcionalidad para la detección de caídas se encuentra dentro de una clase que sigue un patrón *singleton* (todo esto se comentará en detalle más adelante), por lo que solo existe una instancia de esta clase. Todos los hilos que se crean comparten la misma instancia y, por tanto, hay que ser muy cuidadosos al acceder a propiedades de dicha clase cuando se quiere escribir en ellas. Identificar las secciones críticas y protegerlas con mecanismos de sincronización como los semáforos *mutex* es esencial en este caso.

**Problemas encontrados y sus soluciones en la comunicación por *Bluetooth*.**

En este apartado se van a comentar los problemas más importantes que se han presentado durante la implementación de la conexión del dispositivo móvil con el módulo de *Bluetooth* del microcontrolador, así como las soluciones implementadas para resolver dichos problemas.

En una primera implementación, se crearon dos *sockets* para realizar la conexión por *Bluetooth*. Uno de los *sockets* sería usado cuando la aplicación estaba en uso de forma normal y el otro *socket* se usaría para seguir recibiendo datos en segundo plano, cuando la aplicación no se encuentra activa o se producen cambios de pantalla. Cada uno de los *sockets* se creaba usando un puerto distinto y sólo uno de los *sockets* estaría conectado a la vez. Cuando la aplicación se iniciaba, se creaba y conectaba un *socket* de forma normal. Cuando la actividad era destruida (bien por un cambio de actividad o bien por salir de la aplicación), el *socket* conectado debía desconectarse para dar paso al *socket* que era usado en segundo plano. Sin embargo, a partir de esa primera desconexión, los *sockets* no se conectaban con éxito más y se dejaban de recibir datos.

Para resolver ese problema, se ha optado por usar tan solo un *socket Bluetooth*, que es creado en el método `onResume` de la actividad principal (es decir, cuando la actividad es

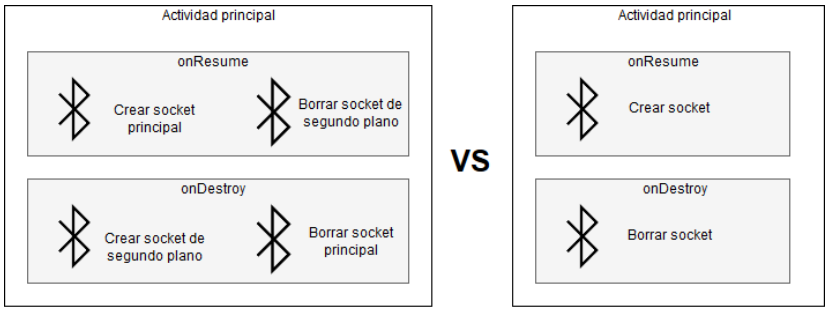



Figura 3.18: Problema en el uso de dos *sockets Bluetooth* y su solución.

creada) y que es destruido en el método `onDestroy`. De esta forma, aunque no se puedan recibir datos en segundo plano, los *sockets* se crean y se conectan al dispositivo remoto de forma correcta.

El último problema a destacar en la comunicación por *Bluetooth* es que en un principio todo el análisis inteligente era realizado por el manejador creado por el hilo secundario (el hilo que trata las lecturas del *socket Bluetooh*). Este manejador se encargaba tanto de realizar el algoritmo de detección de caídas, como del análisis del riesgo de caída en la expedición y el análisis de distancias de los participantes con respecto al guía de la misma. Esto daba como resultado que se leían datos del *socket* algo menos de una vez por segundo, claramente insuficiente para el algoritmo de detección de caídas. Una frecuencia de lectura del *socket* tan baja significa que muchos datos de aceleración del usuario no eran leídos (como se comentará más adelante la detección de caídas está motivada por la superación de unos ciertos umbrales por parte de los valores de aceleración. Si no se realiza la lectura del valor de aceleración que supera esos umbrales, no será posible detectar una caída) por lo que las caídas no se inferían correctamente.

Para resolver este problema, se ha hecho uso de la solución comentada anteriormente. Cada vez que se realiza una lectura del *socket*, se crean dos hilos alternativos (uno que se encarga de la detección de caídas y otro que se encarga del resto del análisis inteligente). De esta forma, se ha logrado multiplicar por cuatro la frecuencia de lectura del *socket*, ya que ahora se realizan unas cuatro lecturas por segundo. Esto ha permitido que la detección de caídas sea mucho más precisa ya que es más difícil que la información asociada a una caída (un determinado valor de aceleración que supera unos ciertos umbrales establecidos) no sea leída (aún así, el algoritmo de detección de caídas también contempla la pérdida de información crítica puntual, como se comentará más adelante).

### 3.3.4 Autenticación de usuarios en `Firebase` y almacenamiento de datos en `Firestore`

`Firebase` es una plataforma para aplicaciones (tanto *webs* como móviles) que está integrada dentro de la nube de `Google` y proporciona, entre otros, un servicio de autenticación de

usuarios (*Firebase Auth*) usando únicamente código en el lado del cliente. En este proyecto se ha decidido usar `Firebase` para homogeneizar la autenticación de usuarios tanto en la plataforma *web* realizada como en la aplicación móvil, evitando reescribir código de acceso a un sistema de almacenamiento de usuarios propio a medida en dos lenguajes distintos (`Java` en la aplicación móvil y `Javascript` en el *backend* de la plataforma *web*).

Además, cabe destacar que `Firebase Auth` proporciona un sistema de autenticación altamente usado y probado por lo que la seguridad en la autenticación es mayor que usando un sistema de autenticación propio. La seguridad en la autenticación es crítica, ya que de ella dependen los datos del usuario que usa la aplicación.

Hay que tener en cuenta que, de acuerdo a la Ley Orgánica de Protección de Datos (LOPD), los datos se recogen de manera lícita, leal y transparente. En este proyecto, el usuario está al tanto de todos los datos que se recogen y los acepta para su monitorización (por lo que los datos tienen un fin legítimo y explícito). No se recogen más datos de los estrictamente necesarios y tampoco se recogen datos de los que el usuario no es consciente. Una de las razones principales por las que se ha usado tanto `Firebase`, como `Firestore`, es que la LOPD exige que los datos personales sean tratados con la adecuada seguridad (y estas plataformas lo cumplen).

Para poder usar `Firebase` en este proyecto, ha sido necesario crear un nuevo proyecto en la *web* de `Firebase` (`https://console.firebase.google.com`). Para ello solo es necesario indicar un nombre de proyecto y una vez creado, añadir al mismo una aplicación *Android* que en este caso es la aplicación que se ha desarrollado. En la Figura 3.19 se puede observar la información de la aplicación *Android*, que será necesario incluir en la misma para poder realizar conexiones con la API de `Firebase`.

Listado 3.9: **Inicio de sesión usando `Firabase Auth`.**

```
FirebaseAuth auth = FirebaseAuth.getInstance();
auth.signInWithEmailAndPassword(email, password).addOnCompleteListener(
  SignupActivity.this, new OnCompleteListener<AuthResult>() {
    @Override
    public void onComplete(@NonNull Task<AuthResult> task) {
      if (task.isSuccessful()) {
        startActivity(new Intent(SignupActivity.this, MainActivity.class));
        finish();
      } else {
        Toast.makeText(SignupActivity.this, "Authentication has failed", Toast.LENGTH_SHORT).show();
      }
    }
  }
);
```

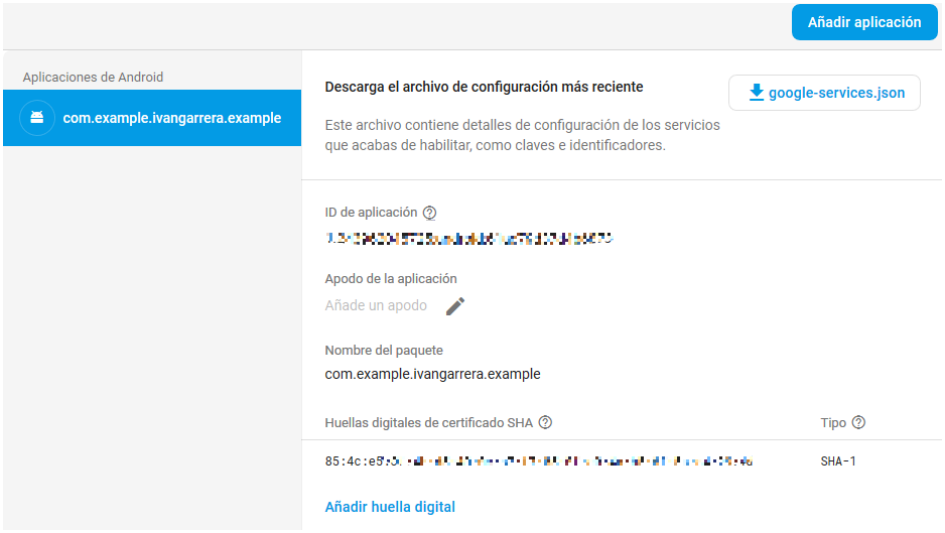



Figura 3.19: Credenciales necesarias para poder acceder a la API de `Firebase` en la aplicación *Android*.

En el Listado 3.9 se puede observar la facilidad en el inicio de sesión, usando `Firebase Auth`. Todas las llamadas a la API de `Firebase` son llamadas asíncronas, es decir, no bloquean el hilo que realiza la llamada hasta que no terminan su trabajo. En este caso, se procede a realizar la autenticación de forma asíncrona (por lo que mientras la autenticación no se completa, el programa seguirá ejecutando las siguientes instrucciones) y el código del método `onComplete` se ejecutará cuando se haya completado la autenticación. Dentro de este *callback* se encontrará la lógica que decide qué hacer si la autenticación ha tenido éxito o qué hacer si no lo ha tenido.

`Firestore` es una base de datos de documentos `NoSQL` rápida, completamente administrada, sin servidor y que ofrece servicios *cloud* para almacenar, sincronizar y consultar datos desde aplicaciones tanto móviles como *web*. Para hacer uso de `Firestore`, no se necesita un servidor intermedio que gestione el acceso a los datos, sino que se usa a través de llamadas a una API definida. Los datos se consultan y envían a `Firestore` serializados en *JavaScript Object Notation* (JSON).

En el Listado 3.10 se puede observar el formato de los datos que proporciona la aplicación móvil y se almacenan en `Firestore`. Estos datos pertenecen a una expedición, que como se puede observar está formada por un guía y una serie de participantes. La información que almacena cada participante es exactamente la misma que la que almacena el guía. Como se puede ver, toda la información almacenada es la necesaria para llevar a cabo los objetivos del proyecto.

Listado 3.10: **Formato de los datos almacenados en `Firestore` para una expedición.**

```
1 {
2   "Duration": int,
```

```
    "EndTime": long,
    "ExpName": string,
    "Guide": {
      "Alerts": [],
      "BatteryLevel": int,
      "Humidity": [],
      "Lat": string,
      "Lon": string,
      "PreviousLocations": [
        {
          "Lat": string,
          "Lon": string,
          "Timestamp": long
        }
      ],
      "Temperature": [],
      "TotalDistance": double,
      "id": string
    },
    "InProgress": boolean,
    "InitTime": long,
    "LastStop": long,
    "Participants": []
}
```

Como se puede ver en el Listado 3.11, la obtención de datos desde la base de datos *cloud* de `Firestore` se realiza a través de llamadas a la API asíncronas (al igual que ocurría con `Firestore`). El método `onComplete` será invocado cuando los datos que se han solicitado estén listos para su uso. Mientras esto no ocurra, la aplicación seguirá ejecutándose de manera normal. El uso de llamadas asíncronas aumenta el rendimiento de la aplicación, puesto que no hay que esperar sin poder ejecutar más código hasta que las llamadas completen su trabajo. Sin embargo, hay que tener en cuenta que los datos que devuelve la llamada a un método asíncrono podrían no estar disponibles cuando se requieren en el código más adelante (en el caso de que sean requeridos).

Listado 3.11: **Obtención de todos los participantes de una determinada expedición.**

```
DocumentReference documentReference = database.collection(DATABASE_NAME).document(expedition_name);
documentReference.get().addOnCompleteListener(new OnCompleteListener<DocumentSnapshot>() {
  @Override
  public void onComplete(@NonNull Task<DocumentSnapshot> task) {
    if (task.isSuccessful()) {
      Map<String, Object> expedition = task.getResult().getData();
      ArrayList<Map<String, Object>> participants =
              (ArrayList<Map<String, Object>>) expedition.get("Participants");
```

```
10      }
11    }
12  });
```

En la Figura 3.20 se puede observar un diagrama de secuencia que resume cómo se realizan en este proyecto las llamadas a la API de `Firestore`. Las llamadas son iniciadas por el usuario en la vista, de forma directa (si pulsa un botón que devuelva las alertas de un participante, por ejemplo) o de forma indirecta (cuando se carga una actividad se cargan de forma automática ciertos datos que se solicitan a la base de datos) y, a través del controlador, se solicitan los datos al modelo. El modelo se encarga de solicitar los datos a `Firestore` y cuando son devueltos, los transforma al formato necesario.

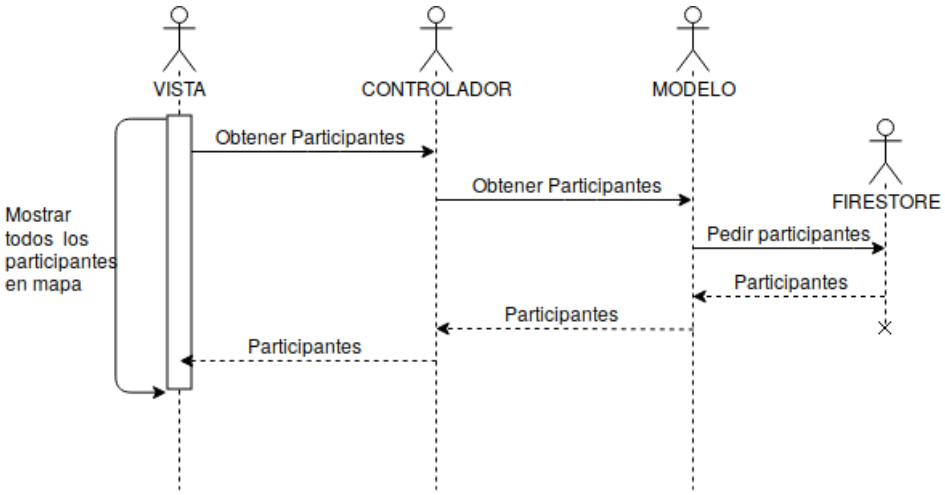



Figura 3.20: Diagrama de secuencia en llamadas a la base de datos en `Firestore`.

## 3.4 Detección automática de situaciones anómalas

Como se ha explicado anteriormente la mayoría de los datos son recogidos a través de los sensores equipados en el microcontrolador y enviados al dipositivo móvil a través de *Bluetooth*. Sin embargo, otros datos como los de posición, los toma el dispositivo móvil (véase Listado 3.12). El dispositivo móvil se encarga de realizar este análisis inteligente.

Los datos que llegan a través de *Bluetooth* tienen que ser sometidos a un **proceso de depuración**, antes de que puedan ser utilizados por los algoritmos de análisis inteligente. El proceso de depuración de datos es necesario porque en ocasiones se pueden leer datos del *socket Bluetooth* incompletos o que están corruptos. Los datos que se envían por *Bluetooth* desde el microcontrolador siguen la estructura que se puede observar en la Figura 3.21, por lo que cuando son recibidos en la aplicación móvil hay que asegurarse de que siguen esa misma estructura. Si cuando se realiza el proceso de depuración de datos, el mensaje no es correcto (bien porque no tenga el formato adecuado o bien porque los datos se hayan corrompido) se

desecha por completo el mensaje ya que inferir por *software* un posible valor para un cierto dato no es adecuado en este caso (se envían los datos con una alta frecuencia, por lo que es preferible desechar un mensaje y recibir rápidamente el siguiente que intentar inferir un valor para un dato determinado).

```
ax: 16480
ay: 140
az: -184
gx: -5
gy: -3
gz: 38
Alert: false
Stop: false
Temperature:  21.90
Humidity:  30.60
lat: 1000.00000
lon: 1000.00000
Light: 340
Rain: 1020
```

Figura 3.21: Formato de un mensaje de datos que se envía a través de *Bluetooth*.

Listado 3.12: **Obtener datos de localización usando el dispositivo móvil.**

```
LocationServices.getFusedLocationProviderClient(this).requestLocationUpdates(mLocationRequest,
  new LocationCallback() {
    @Override
    public void onLocationResult(LocationResult locationResult) {
      super.onLocationResult(locationResult);
      for (Location location : locationResult.getLocations()) {
        // Gestionar el uso de la localizacion
      }
    }
  }, null);
```

El algoritmo que comprueba si los datos recibidos tienen el formato correcto se puede ver en el Listado 3.13. En este algoritmo se comprueba si el mensaje tiene la cantidad de datos que debería tener y si siguen la estructura de la Figura 3.21. En el caso de ser así, el mensaje se acepta y los datos del mensaje se usan en los algoritmos de análisis inteligente. En caso contrario, el mensaje se descarta.

Listado 3.13: **Proceso de comprobación del formato de los datos recibidos por el *socket Bluetooth*.**

```
private boolean checkCorrectFormat(String data) {
  String header_format[] = {"ax", "ay", "az", "gx", "gy", "gz", "Alert", "Stop",
          "Temperature", "Humidity", "lat", "lon", "Light", "Rain"};
```

```
    String[] lines = data.split("\\r?\\n");

    // Check if the length of the received packet is correct
    if (lines.length == header_format.length) {
      for (int index = 0; index < lines.length; index++) {
        String[] word = lines[index].split(":");
        if (!header_format[index].equals(word[0])) {
          return false;
        }
      }
      return true;
    } else {
      return false;
    }
  }
```

### 3.4.1 Algoritmo de detección de caídas

El algoritmo de detección de caídas usa los datos relacionados con la aceleración del usuario para, a partir de un análisis de los mismos, inferir una posible caída. Si se produce una caída, se registrará una alerta con la marca de tiempo del instante en el que se produce la caída y los datos de localización del usuario en el momento de la caída. De esta forma, el guía visualizará la alerta y podrá socorrer al participante de una forma más rápida.

El algoritmo toma como variables de entrada los valores de aceleración en los ejes $X$, $Y$, $Z$. A partir de estos valores, como se comentará a continuación, se produce un evento que produce transiciones en el estado en el que se encuentra el participante.

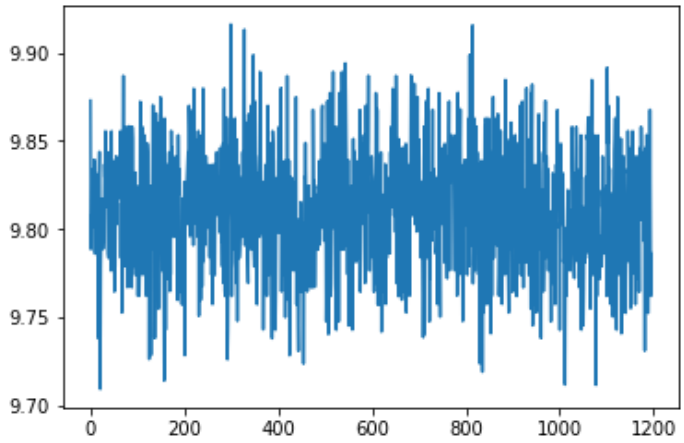



Figura 3.22: Variación de la aceleración lineal cuando el individuo se encuentra en reposo total. La unidad de los datos de aceleración son $m/s^2$.

Si se estudia como varía la aceleración lineal de un usuario durante una caída, se pueden

observar cuatro estados principales:

- **Estado normal**. En este estado, el usuario aún no ha sufrido la caída y se encuentra realizando una actividad diaria de forma normal. En este estado, la aceleración neta no es muy extrema ni muy distinta de la aceleración en reposo (la aceleración en reposo son $9{,}8\ m/s^2$, como se puede ver en la Figura 3.22).
- **Estado de caída libre**. Este estado es muy breve en el tiempo ya que comprende los instantes de tiempo desde que el usuario comienza a caer hasta que impacta con el suelo. Este estado está caracterizado por una aceleración neta muy baja, cercana a $0m/s^2$. La aceleración de la gravedad que actúa sobre nuestro cuerpo tiene un valor de $9{,}8\ m/s^2$ y mientras nuestro cuerpo está cayendo, ejerce una fuerza en sentido contrario de módulo igual (en condiciones ideales sin rozamiento) a la de la gravedad. En realidad la aceleración neta no será cero debido a que existen rozamientos, principalemente.
- **Estado de impacto**. La transición del estado de caída libre al estado de impacto se da en el momento en el que el cuerpo del individuo impacta contra el suelo o contra cualquier otra superficie. Este estado está caracterizado por una aceleración neta muy alta. Cuando el cuerpo de un individuo colisiona contra una superficie, las fuerzas impulsivas se caracterizan por su alto módulo y su breve periodo de acción. La fuerza que ejerce una persona contra el suelo tiene el mismo módulo que la fuerza que ejerce el suelo contra la persona, pero sentido contrario.
- **Estado de reposo**. En este estado un individuo que ha sufrido una caída se encuentra en reposo en el suelo durante un período de tiempo variable. El estado se caracteriza por que la aceleración neta durante este período de tiempo apenas varía y está en torno a los $9{,}8\ m/s^2$.

En la Figura 3.24 se puede observar un ejemplo real de una simulación de una caída. En esta imagen se ven de forma clara los estados que se acaban de comentar y las transiciones entre ellos. De un estado de actividad normal se pasa al estado de caída libre y de este al estado de impacto para después permanecer en el estado de reposo un cierto tiempo. Cabe destacar que los estados de caída libre y de impacto son muy breves en el tiempo, como se puede ver en la imagen. Además, esta gráfica es muy similar a la que se generaría practicando una actividad cotidiana como puede ser un salto.

El algoritmo de detección de caídas que se ha implementado en este proyecto es un algoritmo de detección de caídas basado en umbrales, a partir de los datos de aceleración lineal. En este algoritmo se establecen dos umbrales, uno inferior y otro superior, que deberán ser traspasados para realizar la transición de estados. Para llevar a cabo la detección de caídas se ha diseñado una máquina de estados que tiene un total de cuatro estados:


Figura 3.23: Fuerzas que actúan sobre el cuerpo en caída libre (de forma ideal).

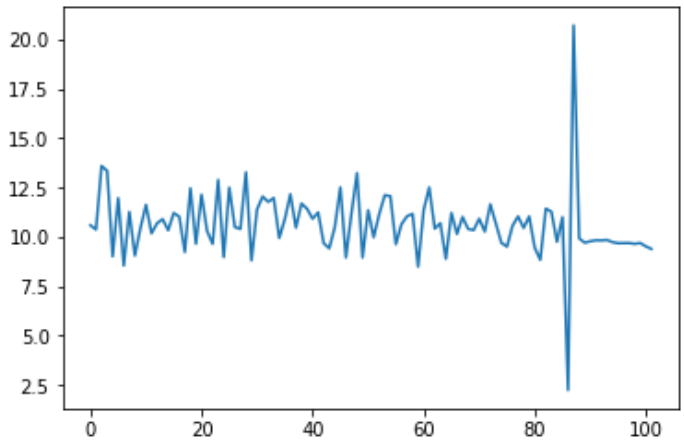



Figura 3.24: Variación de la aceleración lineal durante una caída. La unidad de los datos de aceleración son $m/s^2$.

- Estado normal
- Estado de caída libre
- Estado de impacto
- Estado de caída

Estos estados se corresponden con los comentados anteriormente. En la Figura 3.25 se puede ver un diagrama de estados del algoritmo de detección de caídas. El algoritmo está a la espera de recibir una serie de eventos, que son producidos por los datos de aceleración que llegan al dispositivo móvil (una descripción detallada de los eventos se puede encontrar en la Tabla 3.6). Dependiendo del valor de los datos de aceleración se generan los eventos, utilizados para transicionar entre estados. De forma habitual, el usuario se encontrará en el estado normal. Si se supera un cierto umbral inferior, se generará un evento `LTT` y se transicionará al estado de caída libre y si, desde este estado, se supera un umbral superior se generará un evento `HTT` y se transicionará al estado de impacto. En el estado de impacto se tomarán datos

durante un cierto período de tiempo y se compararán los resultados obtenidos con los que deberían obtenerse en un estado de reposo (véase Figura 3.22). Si ambos resultados coinciden, se generará un evento `MIT` y una caída será inferida.

| Evento | Descripción | Valores que lo generan |
|---|---|---|
| IT | Evento que indica que los valores de aceleración se encuentran entre los dos umbrales | Aceleración $>$ umbral inferior y Aceleración $<$ umbral superior |
| LTT | Evento que indica que los valores de aceleración están por debajo del umbral inferior | Aceleración $\leq$ umbral inferior |
| HTT | Evento que indica que los valores de aceleración están por encima del umbral superior | Aceleración $\geq$ umbral superior |
| MIT | Evento que indica que los valores durante un periodo de tiempo se ajustan a la situación de reposo | Aceleración $\approx 9,8$ |
| MNIT | Evento que indica que los valores durante un periodo de tiempo no se ajustan a la situación de reposo | Aceleración distinta de $9,8$ |

Cuadro 3.6: Descripción de los los eventos que se pueden generar en el algoritmo de detección de caídas y los valores de aceleración que los generan.

Una consideración importante que se ha tenido en cuenta en el algoritmo de detección de caídas es el posible ruido existente en los datos. Es posible que se esté recibiendo información de caída libre y justo después se reciba un evento de datos dentro de los dos umbrales que haría al algoritmo transicionar al estado de normalidad (esto se puede dar simplemente por ruido en los datos o incluso porque un hilo que gestiona datos anteriores de normalidad se ejecute más lentamente que un hilo creado posteriormente con datos de caída libre). Se contempla que si en los siguientes instantes se recibe un evento de superación del umbral superior, la información relativa al estado de normalidad es ruido y se transiciona de forma directa al estado de impacto.

Listado 3.14: **Creación de eventos a partir de los datos de aceleración y transición a otro estado.**

```
1 FD_Event event_occurred;
2 if (force < 0.65 * G) {
3   event_occurred = FD_Event.FD_DataLessThreshold;
4 } else if (force > 2.5 * G) {
5   event_occurred = FD_Event.FD_DataMoreThreshold;
6 } else {
7   event_occurred = FD_Event.FD_DataInsideThreshold;
8 }
```

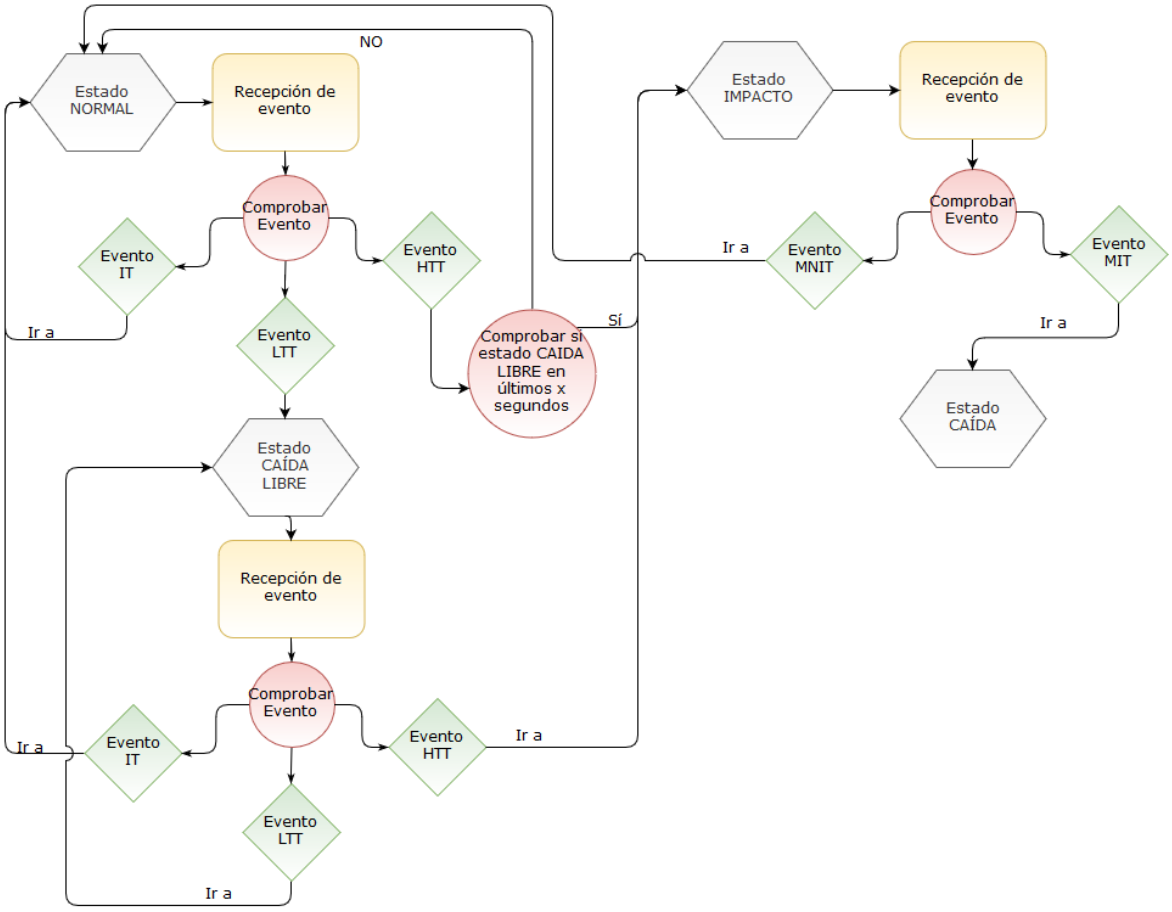



Figura 3.25: Diagrama de estados del algoritmo de detección de caídas.

```
10 FallDetector fallDetector = FallDetector.getInstance();
11 fallDetector.makeTransition(event_occurred, force);
```

En el Listado 3.14 se puede observar el código asociado a la creación de un cierto evento a partir de los datos de aceleración. El valor de aceleración que se compara con los umbrales es el resultado de aplicar la fórmula:

$$|\vec{a}| = \sqrt{a_x^2 + a_y^2 + a_z^2} \tag{3.2}$$

Los valores de aceleración que se han establecido como umbrales son $2{,}5g$ (o $24{,}5\ m/s^2$) para el umbral superior y $0{,}65g$ o ($\approx 6{,}4\ m/s^2$) para el umbral inferior. Los valores se han establecido tanto empíricamente (a base de distintas pruebas de caídas simuladas) y gracias a otros trabajos en algoritmos de detección de caídas basados en umbrales como pueden ser

los encontrados en [Mao17], [Lim14] y [Bou07].

El algoritmo que implementa el diagrama de la Figura 3.25, está basado en una máquina de estados. La máquina de estados sigue un patrón de diseño *singleton*, de forma que sólo hay una instancia de la máquina de estados en todo el código. Como todos los hilos gestionan la misma instancia de la máquina de estados (no tiene sentido tener varios objetos, pues el estado en el que el usuario se encuentra es único y global), la transición entre estados se debe hacer en términos de exclusión mutua. Se puede observar en el Listado 3.15 como el hilo que se dispone a hacer una transición de estados tiene que tomar el control del semáforo `mutex` que gobierna la sincronización en los accesos.

Listado 3.15: **Exclusión mutua en la transición entre estados, en el algoritmo de detección de caídas.**

```
public void makeTransition(FD_Event event, double data) {
  try {
    mutex.acquire(1);
    // Logica de la transicion
  } catch (Exception ex) {
    // Control de errores
  } finally {
    mutex.release(1);
  }
}
```

Como se puede ver en el Listado 3.16, la matriz de transiciones almacena todas las posibles transiciones de la maquina de estados. Cada entrada tiene el siguiente formato:

(Estado en el que se está, Evento que se produce), Estado al que se transiciona

Listado 3.16: **Máquina de estados que gestiona las transiciones entre estados, a partir de eventos generados.**

```
transition_matrix.put(Pair.create(FD_State.FD_NORMAL, FD_Event.FD_DataInsideThreshold), FD_State.
    FD_NORMAL);
transition_matrix.put(Pair.create(FD_State.FD_NORMAL, FD_Event.FD_DataMoreThreshold), FD_State.
    FD_NORMAL);
transition_matrix.put(Pair.create(FD_State.FD_NORMAL, FD_Event.FD_DataLessThreshold), FD_State.
    FD_FREE_FALL);

transition_matrix.put(Pair.create(FD_State.FD_FREE_FALL, FD_Event.FD_DataInsideThreshold), FD_State.
    FD_NORMAL);
transition_matrix.put(Pair.create(FD_State.FD_FREE_FALL, FD_Event.FD_DataLessThreshold), FD_State.
    FD_FREE_FALL);
transition_matrix.put(Pair.create(FD_State.FD_FREE_FALL, FD_Event.FD_DataMoreThreshold), FD_State.
    FD_IMPACT);
```

```
9   ...
```

### 3.4.2 Grado de dispersión del grupo

El grado de dispersión del grupo es un posible indicador de riesgo en la expedición. Un alto grado de dispersión significa que el grupo de expedición está muy separado, por lo que una posible alerta de un participante demasiado lejos puede tardar más en ser atendida, con los riesgos que esto conlleva. El guía puede usar este valor para decidir realizar una parada de reagrupación, o al menos reducir el ritmo de marcha hasta que todos los usuarios están agrupados. Un alto grado de dispersión en el grupo puede indicar también síntomas de fatiga en los participantes que se encuentran mas rezagados. Esta fatiga puede acarrear otros problemas como la deshidratación o posibles caídas, por lo que es recomendable tenerla controlada.

El grado de dispersión del grupo se mide como la media de las distancias entre el guía y los participantes:

- Sea $g$ el guía de la expedición.
- Sea $\mathcal{P}$ el conjunto de todos los participantes de la expedición.
- Sea $n$ el tamaño del conjunto $\mathcal{P}$.
- Sea $distancia(a, b)$ la distancia de ***Harvesine***[12] existente entre dos usuarios $a$ y $b$, con coordenadas $(Lat_a, Lon_a)$ y $(Lat_b, Lon_b)$ en una circunferencia de radio $r$:

$$distancia(a, b) = 2 \cdot r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{Lat_b - Lat_a}{2}\right) + \cos\left(Lat_a\right) \cdot \cos\left(Lat_b\right) \cdot \sin^2\left(\frac{Lon_b - Lon_a}{2}\right)}\right) \quad (3.3)$$

Entonces:

$$\text{Grado de dispersión} = \frac{1}{n} \cdot \sum_{i=1}^{n} distancia(g, p), \forall p \in \mathcal{P} \quad (3.4)$$

[12] La distancia de *Harvesine* se explica más a fondo en la Sección 3.5.2

Póngase como ejemplo un grupo de expedición en el que participan cuatro usuarios (tres de ellos actúan como participantes y uno como guía). Se suponen las siguientes distancias de separación (calculadas usando la fórmula de *Harvesine*) entre el guía y los participantes:

- Distancia entre guía y participante $1 = 0{,}3km$
- Distancia entre guía y participante $2 = 0{,}1km$
- Distancia entre guía y participante $3 = 0{,}8km$

El grado de dispersión del grupo será $\frac{1}{3} \cdot (0{,}3 + 0{,}1 + 0{,}8) = \frac{1}{3} \cdot 1{,}2 = 0{,}4km$

### 3.4.3 Análisis del riesgo de caída en la expedición de cada integrante de la misma

Realizar un análisis del riesgo de caída existente en un cierto instante de tiempo resulta una gran ayuda tanto para los participantes como para el guía. Gracias a este análisis, los participantes pueden visualizar el riesgo al que están expuestos en un determinado momento por lo que pueden poner los medios de su parte para intentar minimizar ese riesgo por ejemplo, aminorando la velocidad, haciendo uso de una linterna si las condiciones de luz son desfavorables o poniéndose al resguardo de la lluvia en el caso de haber una tormenta. Del mismo modo, el riesgo de caída es también muy útil para el guía, que en función de éste puede decidir realizar una parada para la reagrupación total del grupo, y así tener controlados a todos los participantes del mismo. El riesgo de caída de un integrante en la expedición depende de varios factores, como por ejemplo la cantidad de luz que existe en el entorno en el que se encuentran, la humedad que podría dificultar la estabilidad del terreno y las condiciones climatológicas adversas, como la lluvia, que hacen que el terreno por el que se mueven los usuarios presente un mayor riesgo de caída. Todos ellos, debidamente combinados, nos pueden reportar información realmente útil para determinar el riesgo de caída durante la expedición.

Para realizar el análisis del riesgo de caída en el presente proyecto, se ha decidido hacer uso de la **lógica difusa**. La lógica difusa [Gon11] proporciona un mecanismo de inferencia que permite simular los procedimientos de razonamiento humano en sistemas basados en el conocimiento. El cálculo del riesgo de caída es un problema en el que existe incertidumbre y vaguedad en los datos ya que no es posible afirmar con certeza si un usuario en una ruta de expedición va a sufrir una caída o no. Por lo tanto, este modelo matemático es el apropiado para lidiar con esta problemática.

La lógica difusa se trata de un modelo matemático adecuado para tratar la incertidumbre y la vaguedad de los datos, dos situaciones que se plantean en este proyecto. Existe incertidumbre ya que no se puede asegurar con precisión si un usuario se caerá en una cierta situación. Existe vaguedad en los datos (sobre todo con los datos de humedad y luz) ya que no se puede precisar si un cierto valor de humedad o luz pertenece a un conjunto (por ejemplo, no tiene

sentido inferir que una humedad del 74 % es una humedad media y una humedad del 75 % es una humedad alta).

| Variable de Entrada | Dominio de Definición | Conjunto Difuso | Rango de Valores |
|---|---|---|---|
| Lluvia | [0 - 1023] | Sí | [0 - 562] |
| | | No | [462 - 1023] |
| Luz | [0 - 1023] | Muy Oscuro | [0 - 200] |
| | | Oscuro | [100 - 400] |
| | | Media | [300 - 600] |
| | | Media-alta | [500 - 900] |
| | | Máxima | [800 - 1023] |
| Humedad | [0 -100] | Muy baja | [0 -20] |
| | | Baja | [10 - 40] |
| | | Media | [30 - 60] |
| | | Alta | [50 - 80] |
| | | Muy alta | [70 - 100] |

Cuadro 3.7: Variables de entrada y conjuntos difusos de dichas variables, del algoritmo de riesgo de caída.

| Variable de Salida | Dominio de definición | Conjunto Difuso | Rango de Valores |
|---|---|---|---|
| Riesgo | [0 - 1] | Bajo | [0 - 0.4] |
| | | Medio | [0.3 - 0.8] |
| | | Alto | [0.7 - 1] |

Cuadro 3.8: Variable de salida y conjuntos difusos de dicha variable, del algoritmo de riesgo de caída.

Los conjuntos difusos permiten a sus elementos tener un grado de pertenencia a los mismos. La lógica difusa emplea valores continuos entre 0 (pertenencia nula a un conjunto difuso) y 1 (pertenencia total a un conjunto difuso). Mediante el proceso de *fuzzificación*, los valores de las distintas variables de entrada se asocian con los conjuntos difusos definidos (véase Tabla 3.7). Una vez se ha realizado la *fuzzificación*, se aplican reglas lingüísticas para obtener una salida (véase Tabla 3.8). Esta salida se puede *defuzzificar* (para obtener un valor discreto) o tratarse de forma *fuzzificada* (en lenguaje natural). En este proyecto, la salida se trata de forma *fuzzificada*, ya que los usuarios de la expedición sólo necesitan saber el riesgo de caída mediante una etiqueta lingüística (alto, medio o bajo), sin necesidad de conocer un valor numérico preciso.

La función de pertenencia, asociada a cada variable, define el grado de pertenencia de un cierto valor de entrada a un conjunto difuso. El valor de entrada se puede *fuzzificar* asignando la etiqueta lingüística asociada a un conjunto difuso. Una vez que se tienen *fuzzificados* los

valores de todas las variables de entrada, comienza el proceso de inferencia haciendo uso del conjunto de reglas definidas, que siguen el formato:

*Si variable de entrada pertenece a un conjunto $x$ → variable de salida pertenece al conjunto $y$*

Cabe destacar que la sentencia condicional anterior se puede extender mediante el uso de operadores lógicos (*and*, *or* y *not* lógicos) para encadenar varias condiciones en una misma regla. Las reglas que se han diseñado en este proyecto se pueden observar a continuación:

- **R1.** Si lluvia es sí ∧ (luz es media-alta ∨ luz es máxima) ∧ (humedad es muy baja ∨ humedad es baja ∨ humedad es media) → riesgo es bajo
- **R2.** Si lluvia es sí ∧ (luz es media-alta ∨ luz es máxima) ∧ (humedad es alta ∨ humedad muy alta) → riesgo es medio
- **R3.** Si lluvia es sí ∧ luz es media ∧ (humedad es muy baja ∨ humedad es baja ∨ humedad es media) → riesgo es medio
- **R4.** Si lluvia es sí ∧ (luz es oscuro ∨ luz es muy oscuro) → riesgo es alto
- **R5.** Si lluvia es sí ∧ luz es media ∧ (humedad es alta ∨ humedad muy alta) → riesgo es alto
- **R6.** Si lluvia es no ∧ (luz es media-alta ∨ luz es máxima) → riesgo es bajo
- **R7.** Si lluvia es no ∧ luz es media ∧ (humedad es muy baja ∨ humedad es baja ∨ humedad es media) → riesgo es bajo
- **R8.** Si lluvia es no ∧ luz es media ∧ (humedad es alta ∨ humedad muy alta) → riesgo es medio
- **R9.** Si lluvia es no ∧ luz es oscuro ∧ (humedad es muy baja ∨ humedad es baja ∨ humedad es media) → riesgo es medio
- **R10.** Si lluvia es no ∧ luz es oscuro ∧ (humedad es alta ∨ humedad muy alta) → riesgo es alto
- **R11.** Si lluvia es no ∧ luz es muy oscuro ∧ (humedad es muy baja ∨ humedad es baja) → riesgo es medio
- **R12.** Si lluvia es no ∧ luz es muy oscuro ∧ (humedad es media ∨ humedad es alta ∨ humedad muy alta) → riesgo es alto

En el Listado 3.17 se puede observar la correspondencia de alguna de las anteriores reglas en código `Java`.

Para implementar la lógica difusa en la aplicación *Android*, se ha usado la librería `jfuzzylogic`[13]. Esta librería usa varios tipos de *defuzzificadores*, de reglas de agregación, de reglas

[13] `http://jfuzzylogic.sourceforge.net/html/index.html`

de implicación y de operadores de conexión entre reglas, de forma que se adapta a la situación en la que se vaya a usar de forma muy fácil.

En este proyecto se usa el máximo como regla de agregación y el *defuzzificador centroid*.

En las Figuras 3.26 y 3.27 se pueden observar entre qué valores (del dominio de definición de la variable) se encuentran los distintos conjuntos difusos.

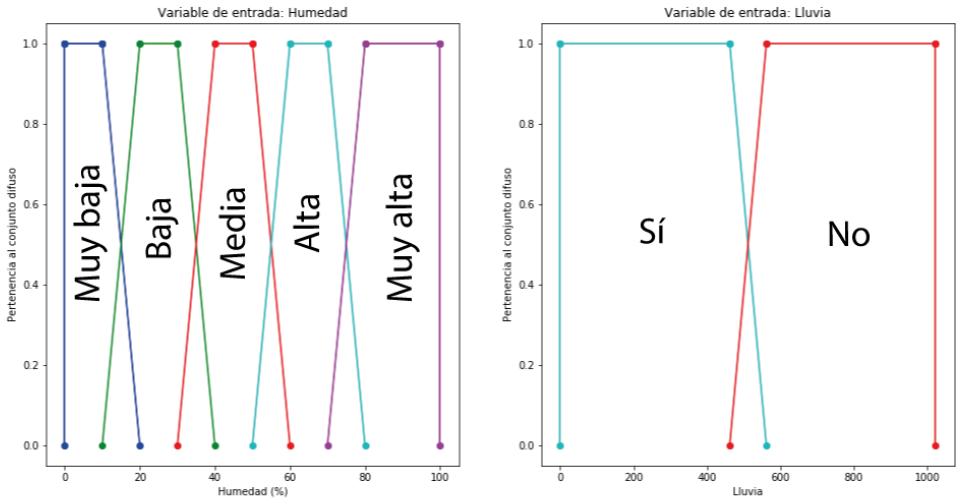



Figura 3.26: Valores de los conjuntos difusos, de las variables de entrada de humedad y lluvia.

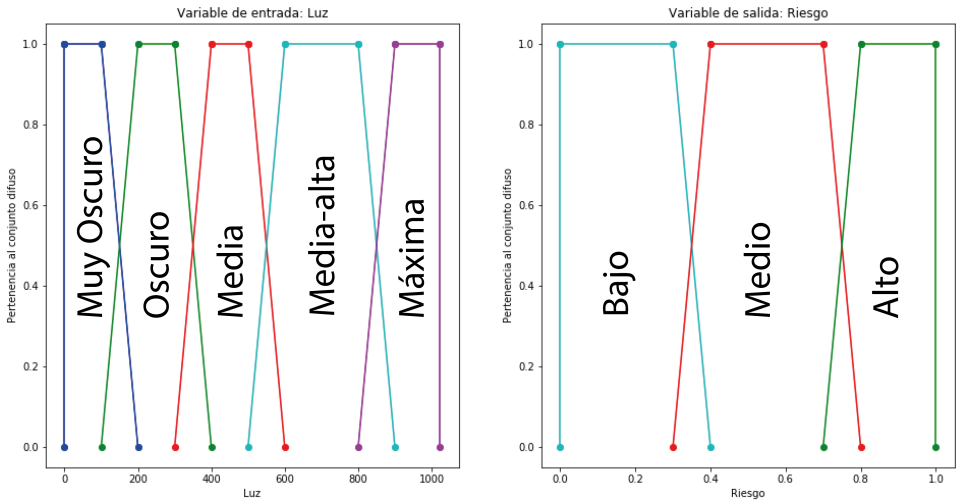



Figura 3.27: Valores de los conjuntos difusos, de la variable de entrada de luz y la variable de salida de riesgo.

Listado 3.17: **Algunas de las reglas difusas definidas para obtener el grado de pertenencia de una variable de salida a los conjuntos difusos.**

```
1 ruleBlock.addRule(Rule.parse("if rain is yes and (light is midhigh or light is maximum) and (
      humidity is veryLow or humidity is low or humidity is mid) then risk is low", engine));
2 ruleBlock.addRule(Rule.parse("if rain is yes and (light is midhigh or light is maximum) and (
      humidity is high or humidity is veryHigh) then risk is mid", engine));
3 ruleBlock.addRule(Rule.parse("if rain is yes and light is mid and (humidity is veryLow or humidity
      is low or humidity is mid) then risk is mid", engine));
4 ruleBlock.addRule(Rule.parse("if (humidity is high or humidity is veryHigh) and rain is yes then
      risk is high", engine));
5 ...
```

Entre las ventajas del uso de la lógica difusa en este proyecto destacan:

- Mayor interpretabilidad de los datos de salida para los usuarios, ya que visualizan etiquetas lingüísticas que son más familiares que un valor numérico de riesgo de caída.
- Una regla difusa equivale a múltiples reglas normales, por lo que puede cubrir una gran cantidad de casos.
- Si en algún momento se quiere cambiar el comportamiento del algoritmo, solo es necesario redefinir las reglas (añadiendo nuevas reglas o modificando o eliminando las reglas existentes) o variar los dominios de definición de las variables de entrada y salida.
- La inclusión de nuevos sensores que formen parte del análisis del riesgo de caída es sencilla. Tan sólo es necesario crear una nueva variable de entrada y añadir nuevas reglas que la incluyan.

## 3.5 Plataforma *web*

En esta sección se muestra el diseño y la implementación de la plataforma *web*, encargada de la visualización (tanto en tiempo real como de forma histórica) de la información relacionada con las expediciones. La plataforma *web* sigue una arquitectura cliente-servidor, en la que el *backend* se ha desarrollado usando `node-js` y el *frontend* ha sido desarrollado usando `react`.

### 3.5.1 Diseño de la plataforma *web*

Como se ha comentado anteriormente, el *frontend* de la plataforma *web* se ha desarrollado usando `React`, una librería basada en componentes y escrita en `Javascript`. `React` es una librería que se utiliza para crear la interfaz de usuario de la página *web* a través de componentes. Un componente es un elemento visual que forma parte de la interfaz, que tiene su propio estado e implementa su propia lógica de renderizado, por lo que una misma vista se puede componer por varios componentes que se podrán mostrar en pantalla de formas distintas, de

acuerdo a la lógica de renderizado de cada uno de los componentes. Para desarrollar el *frontend*, además de `React` se ha usado `Bootstrap` con el objetivo de realizar un diseño *responsive* que se adapte a distintos tamaños de pantalla (ordenador, móvil o *tablet* entre otros).

La arquitectura del sistema *web* se puede observar en la Figura 3.28. El usuario interacciona con el sistema a través de la interfaz de usuario que éste proporciona. Dependiendo de la acción, se generarán peticiones HTTP (`GET` y `POST`) a una API de funciones expuestas en el servidor `Express`. Estas funciones se encargarán de las tareas de gestión de usuarios (autenticación y deautenticación de los mismos) y gestión de las expediciones (obtener las expediciones de un usuario, obtener información detallada acerca de una expedición en concreto, obtener las alertas de un participante, etc.). Una vez que la información que ha sido solicitada por el *frontend* a través de la petición HTTP está lista, el *backend* responde a la petición con dicha información.

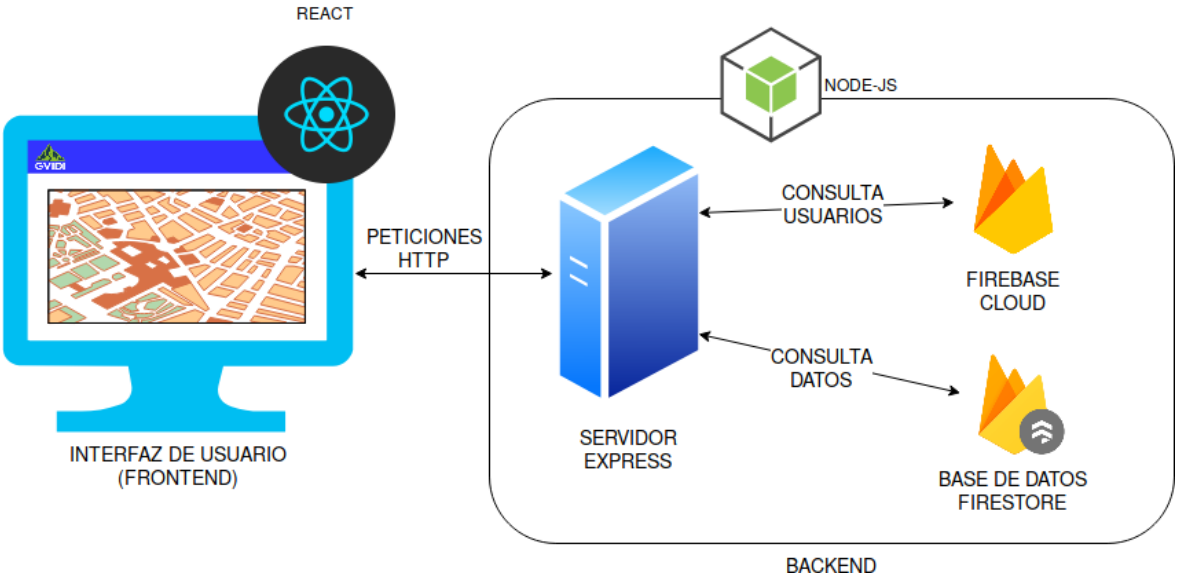



Figura 3.28: Arquitectura de la plataforma *web*.

En las Figuras 3.29 a 3.32 se puede observar la interfaz de usuario de la plataforma *web*. Cabe destacar la visualización sobre el mapa en tiempo real de la expedición. Sobre este mapa también se mostrarán las alertas (de caídas, pérdida de miembro del grupo o excesiva distancia con respecto al guía) si las hubiera. Si se hace *click* sobre alguna de las alertas o sobre cualquier otro punto en la ruta sobre el mapa, debajo del mismo se mostrará una tarjeta con información detallada de la expedición en ese punto.

Se realizó una primera versión de la plataforma *web* en la que toda la lógica de gestión de usuarios y recuperación de datos de expediciones se hacía en el *frontend*, implementada en los propios componentes `React`. El componente `React` de inicio de sesión, gestionaba por ejemplo la autenticación de un usuario, recuperando los datos de la vista, realizando la conexión con el sistema de autenticación de `Firebase` y alterando el estado del componente

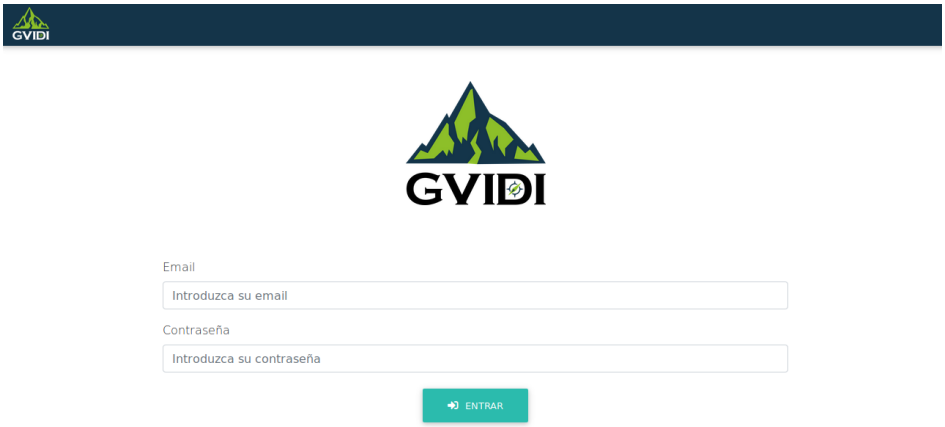



Figura 3.29: Página de *Login* de la *web*.


Figura 3.30: Página que contiene el histórico de rutas de un usuario.


Figura 3.31: Tarjetas con información general acerca de la ruta.


Figura 3.32: Mapa en tiempo real de la expedición con información detallada del punto en el que se haga *click*.

con el resultado de la autenticación.

Se obtuvo una plataforma *web* totalmente funcional, que cumplía con todos los objetivos propuestos pero claramente insegura. Las credenciales para la conexión con las APIs de `Firebase` y `Firestore` las gestionaba el *frontend* por lo que cualquier usuario podría visualizarlas desde el propio navegador. La obtención de dichas credenciales por parte de terceras personas podría hacer que se borrasen usuarios de las bases de datos y se alterasen los valores de las expediciones, entre otras acciones maliciosas.

Para remediarlo se decidió añadir el *backend* usando `NodeJS`. Este *backend* implementa una serie de controladores con funciones, con las que el *frontend* interactúa a través de consultas HTTP. La lógica de conexión con `Firestore` y `Firebase`, reside en el servidor por lo que el usuario no es capaz de consultar ningún tipo de credencial desde su navegador. Con esta solución, si un usuario quiere iniciar sesión, enviará una petición al servidor con los datos de autenticación para que sea este quien le autentique. El servidor simplemente responderá con un valor que indicará si la autenticación ha sido satisfactoria.

### 3.5.2 Implementación de la plataforma *web*

Para la implementación de la plataforma *web* se ha usado `Javascript` como lenguaje de programación tanto en el *frontend* (con `React` como librería escrita en `Javascript`) como en el *backend* (con `Node` como *framework* escrito en `Javascript`). También se ha hecho uso del lenguaje de marcas HTML para definir el esqueleto de las distintas páginas *web* que conforman el *frontend*. Aunque no se han programado estilos propios usando *Cascading Style Sheets* (CSS), se ha usado `Bootstrap` para definir los estilos de la vista, mediante uso de plantillas de diseño predefinidas que permiten un prototipado rápido de la *web*.

### Servidor `Express`

Para implementar la lógica del servidor se ha hecho uso de `Express`, un *framework* usado para implementar aplicaciones *web* en la parte del servidor usando `Node`. Este *framework* proporciona una serie de características que permiten construir aplicaciones de una única página, multipágina e híbridas de una forma sencilla. La instalación de `Express` se realiza a través del gestor de paquetes de `node` (*node package manager* (NPM)). En el Listado 3.18 se puede observar la creación del servidor `Express`. Simplemente es necesario inicializar el módulo `Express`, establecer el puerto en el que el servidor escuchará las peticiones entrantes y establecer las rutas que el servidor gestionará (véase Listado 3.19), con el controlador para cada una de las rutas (véase Listado 3.20), que será la función que se ejecutará cuando se haga una petición a dicha ruta (ya sea una petición `GET` o una petición `POST`).

Listado 3.18: **Creación del servidor `Express` de la plataforma *web*.**

```
const express = require('express');
const http = require('http');

const app = express();
// El puerto por defecto del servidor es el 4000
app.set('port', process.env.PORT || 4000);

// Establecer las rutas a la api de usuarios (para interactuar con usuarios)
app.use('/api/users', require('./routes/UserRoutes'));
// Establecer las rutas a la api de expediciones
app.use('/api/expeditions', require('./routes/ExpeditionRoutes'));

var server_created = http.createServer(app).listen(app.settings.port, function() {
        console.log ('Server listening on port ' + app.settings.port);
});
```

Listado 3.19: **Rutas que gestiona el servidor al manejar peticiones a la API de expediciones.**

```
const express = require('express');
const expeditionRouter = express.Router();

const expeditionController = require('../controllers/ExpeditionController');

expeditionRouter.get('/allexpeditions', expeditionController.GetExpeditionsFromUser);
expeditionRouter.get('/expedition', expeditionController.GetExpedition);
...

module.exports = expeditionRouter;
```

Listado 3.20: **Controlador de expediciones, que gestiona la lógica de la petición.**

```
const firebase = require('firebase');

const expeditionController = {};

// Funcion que proporciona todas las expediciones de las que un usuario ha formado parte
expeditionController.GetExpeditionsFromUser = (req, res) => {
  const db = firebase.firestore();
  const expeditionsRef = db.collection('Expeditions');
  const current_user = firebase.auth().currentUser;
  let expeditions_array = [];

  if (current_user) {
    ...
    res.json(expeditions_array);
  }
};

module.exports = expeditionController;
```

Por tanto, como se puede ver en la Figura 3.33 la lógica del *backend* se puede resumir en el servidor de aplicación `Express` que gestiona las peticiones entrantes y salientes, la API que queda definida por el conjunto de rutas a las que se puede acceder desde el exterior (desde el *frontend* o incluso desde otra aplicación *web*, siempre y cuando se exponga el servidor *backend* a Internet) y por los controladores que implementan la lógica de la petición, es decir, de qué forma actuar ante las peticiones a la API.

### Autenticación de usuarios en `Firebase` y almacenamiento de datos en `Firestore`

Al igual que ocurría con la aplicación móvil, se ha hecho uso de `Firebase` para el proceso de autenticación de usuarios y de `Firestore` como base de datos en tiempo real `NoSQL`. En este punto del proyecto es en el que realmente se reflejan los puntos fuertes de haber elegido estos servicios para gestionar la autenticación y gestión de datos ya que se ha ahorrado una cantidad de tiempo considerable con esta elección. Se han omitido las fases de diseño e implementación de una solución propia para la gestión de datos, además de las dificultades relacionadas con cerciorarse de que existen políticas de seguridad adecuadas para acceder a los datos.

`Firebase` y `Firestore` proporcionan APIs en varios lenguajes de programación (entre los que se incluyen `Java` para la aplicación móvil y `Javascript` para la plataforma *web*) lo que hace que el sistema de autenticación y de gestión de datos quede unificado en todo el proyecto.

Para poder usar `Firebase` en la plataforma *web* es necesario usar las mismas credenciales

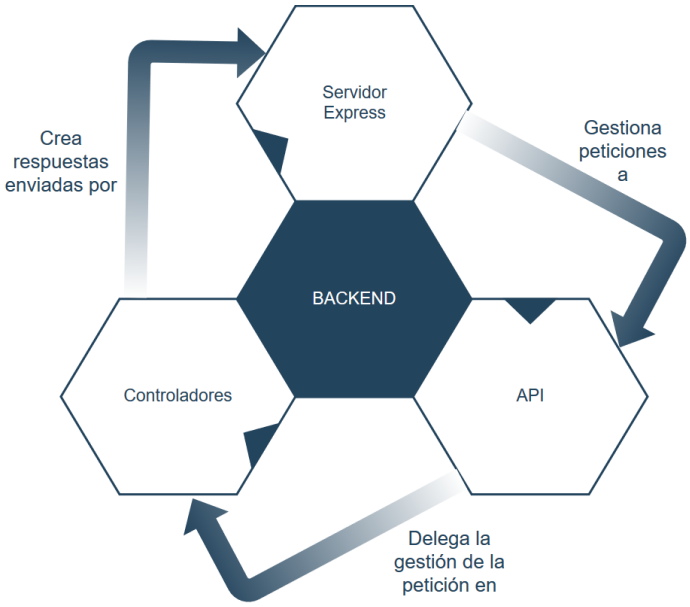



Figura 3.33: Esquema del *backend* de la plataforma *web*.

que las que se usaron para acceder a la API de la aplicación móvil. En el Listado 3.21 se puede observar la sencillez en la conexión entre la plataforma *web* y los servicios de `Firebase`. Una vez que se tienen las credenciales de la API, solo es necesario importar el paquete de `Firebase` (que se puede instalar a través de NPM) e inicializar la aplicación. Este paquete contiene el *Software Development Kit* (SDK) para poder hacer uso de `Firebase` en el proyecto *web*.

Listado 3.21: **Creación de la conexión con `Firebase` en la aplicación *web*.**

```
const firebase = require('firebase')

const config = {
    apiKey: CLAVE_DE_LA_API,
    authDomain: 'ejemplo.firebaseapp.com',
    databaseURL: 'https://ejemplo.firebaseio.com"',
    storageBucket: 'ejemplo.appspot.com',
    projectId: 'ejemplo'
};

firebase.initializeApp(config);
```

Como `Firestore` es un servicio de almacenamiento de datos en tiempo real integrado dentro de `Firebase`, no es necesario usar credenciales distintas para acceder a los servicios de almacenamiento de datos. Una vez que se ha habilitado el uso de `Firestore` como sistema de gestión de datos desde la consola del proyecto de `Firebase`[14] (véase Figura 3.34), se pueden realizar peticiones a la base de datos de una forma sencilla, como se puede ver en el Listado

[14] `https://console.firebase.google.com`

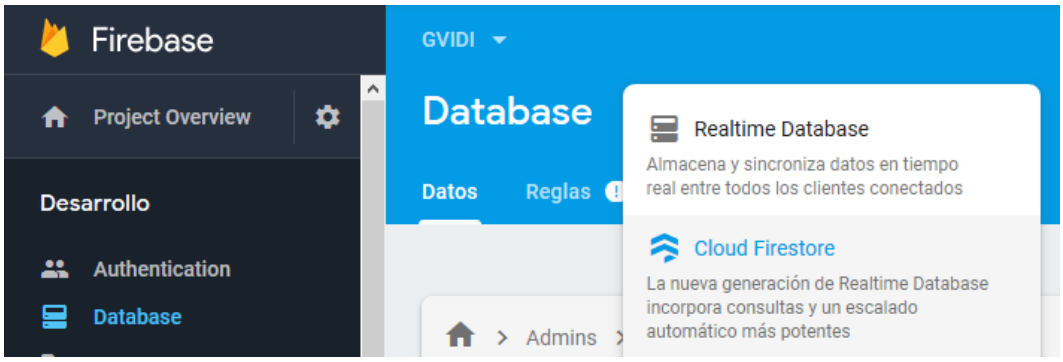



Figura 3.34: Habilitar `Firestore` desde el proyecto de `Firebase`.

3.22.

Listado 3.22: **Consulta de los datos de todas las expediciones usando `Firestore` y `Javascript`.**

```
1 const db = firebase.firestore();
2 const expeditionsRef = db.collection('Expeditions');

4 expeditionsRef.get().then(querySnapshot => {
5   querySnapshot.forEach(doc => {
6     const data = doc.data();
7   });
8 });
```

### Mapa del recorrido

Para poder visualizar el recorrido que realiza un usuario en la expedición, se ha decidido añadir un mapa sobre el cual se irá dibujando, en forma de líneas, el camino que se realiza en la expedición (véase Figura 3.35). Además sobre el mapa se añadirán otros indicadores visuales, en forma de marcadores, para identificar las alertas que se han producido. `Google Maps` es la alternativa más contemplada en lo que al uso de mapas se refiere, ya que posee una API para desarrolladores que permite explotar de diversas formas el uso de los mapas (en este proyecto se usará principalmente la creación de rutas o caminos sobre un mapa y la inserción y customización de marcadores en el mismo).

Como se puede observar en la sección A.6.2 del antiguo capítulo de antecedentes, añadido en el presente proyecto en forma de anexo, lo único necesario para usar la API de `Google Maps` en este proyecto es una «API *Key*»[15]. En el Listado 3.23 se puede observar la creación del mapa usando `React`. Además del mapa, también se crean los marcadores de las alertas de la ruta y la línea poligonal que dará lugar a la ruta por la que la expedición transcurre. Para que el usuario identifique de forma visual el tipo de alerta en la *web*, se cambia la imagen del

[15] `https://developers.google.com/maps/documentation/javascript/get-api-key`


Figura 3.35: Mapa durante el recorrido de la expedición, en el que se pueden consultar detalles y alertas de dicha expedición.

marcador dependiendo de la alerta.

Listado 3.23: **Creación del mapa en la plataforma *web*, incluyendo la ruta y los marcadores de alertas**.

```
1  googleMapsAPI(API_CONFIG).then(googleMaps => {
2    // Crear un nuevo mapa y establecer el punto central y el zoom
3    let map = new googleMaps.Map(self.refs.map, { center: { lat: lat, lng: lng }, zoom: 16 });
4
5     // Cada alerta es un marcador en el mapa
6    self.state.alerts.forEach(alert => {
7      let myLatlng = new googleMaps.LatLng(alert.Latitude, alert.Longitude);
8      let image = undefined;
9      // Depende del tipo de alerta, el marcador tendra un aspecto u otro
10     if (alert.AlertType === 'STOP') {
11       image = { url: window.location.origin + '/marker_stop_32.png', size: new googleMaps.Size(32,
             64), origin: new googleMaps.Point(0, 0) }
12     else if (alert.AlertType === 'SOS') {
13       image = { url: window.location.origin + '/marker_sos_32.png', size: new googleMaps.Size(32,
             64), origin: new googleMaps.Point(0, 0) }
14     }
15     let marker = new googleMaps.Marker({ position: myLatlng, animation: googleMaps.Animation.DROP,
           title: 'Alerta', icon: image });
16     // Listener que se ejecutara cuando se hace click en un marcador
17     marker.addListener('click', args => {
18        ...
19     });
20     marker.setMap(map);
21   });
```

```
23    // Crear la linea de la ruta
24    let poli = new googleMaps.Polyline({
25      path: self.state.locations,
26      geodesic: true,
27      strokeColor: '#FF0000',
28      strokeOpacity: 1.0,
29      strokeWeight: 2
30    });

32    poli.setMap(map);

34    // Listener que se ejecutara cuando se hace click en la ruta
35    poli.addListener('click', args => {
36      ...
37    });
38  });
```

Cuando el usuario pulsa un marcador, debajo del mapa aparece una tarjeta con información detallada acerca de la alerta. De igual forma, cuando el usuario pulsa sobre un punto en la ruta del mapa, se busca el instante más cercano al punto seleccionado en el cual se tienen datos de la expedición y se muestra una tarjeta debajo del mapa con información detallada acerca de la expedición en ese instante.

La aplicación móvil almacena una nueva entrada con datos relativos a la posición del usuario en la expedición cada veinte segundos, por lo que es muy probable que las coordenadas elegidas por el usuario cuando pulsa un punto en la ruta del mapa no dispongan de información para ese mismo instante. Lo que se hace en este caso es buscar el punto más cercano a las coordenadas elegidas por el usuario del que se dispone información y mostrar la información de ese punto. Esto no es ningún problema, ya que si se pulsa un punto en la ruta durante el transcurso normal de la expedición se verá la información del punto más cercano a las coordenadas seleccionadas del que se disponen datos y como el período de envío de datos de localización es tan alto (cada veinte segundos) el usuario no notará una falta de información en ningún momento. Si se produce una alerta las coordenadas que se muestran si que son las exactas de la alerta, ya que en caso de caída se requiere la máxima precisión posible para identificar el lugar de la caída.

En el Listado 3.24 se puede observar el algoritmo que implementa la funcionalidad que se acaba de explicar y en el Listado 3.25 se encuentra el algoritmo necesario para calcular la distancia entre dos puntos. Este algoritmo está basado en la fórmula de *Harvesine*, que calcula la distancia ortodrómica o mínima distancia entre dos puntos en la superficie terrestre (véase Figura 3.36), sabiendo sus longitudes y latitudes. La fórmula de *Harvesine* es la siguiente:

$$d = 2 \cdot r \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cdot \cos(\phi_2) \cdot \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)}\right) \quad (3.5)$$

donde $\phi_1, \lambda_1$ se refiere a la latitud y longitud del primer punto, $\phi_2, \lambda_2$ se refiere a la latitud y longitud del segundo punto y $r$ se refiere al radio de la circunferencia que engloba los puntos (en este caso, el radio terrestre).

Listado 3.24: **Obtención de la localización más cercana a un punto dado**.

```
expeditionController.GetMinLocation = (req, res) => {
  const db = firebase.firestore();
  const expeditionsRef = db.collection('Expeditions');

  let expeditionName = req.query.expName;
  let lat = req.query.lat;
  let lon = req.query.lon;
  let min_distance = Infinity;
  let min_location = null;

  expeditionsRef.get().then(querySnapshot => {
    querySnapshot.forEach(doc => {
      const data = doc.data();
      if (data.ExpName === expeditionName) {
        data.Guide.PreviousLocations.forEach(location => {
          let distance = getDistanceBetweenTwoPoints(lat, lon, parseFloat(location.Lat), parseFloat(
              location.Lon));
          if (distance < min_distance) {
            min_distance = distance;
            min_location = location;
          }
        });
      }
    });
  });
};
```

Listado 3.25: **Obtención de la distancia entre dos puntos usando la formula de *Harvesine*.**

```
function getDistanceBetweenTwoPoints(lat_first, lon_first, lat_second, lon_second) {
  lat_first = toRadians(Math.abs(lat_first));
  lon_first = toRadians(Math.abs(lon_first));
```

[16]Imagen extraída de https://upload.wikimedia.org/wikipedia/commons/thumb/c/c2/Ortodroma.svg/375px-Ortodroma.svg.png

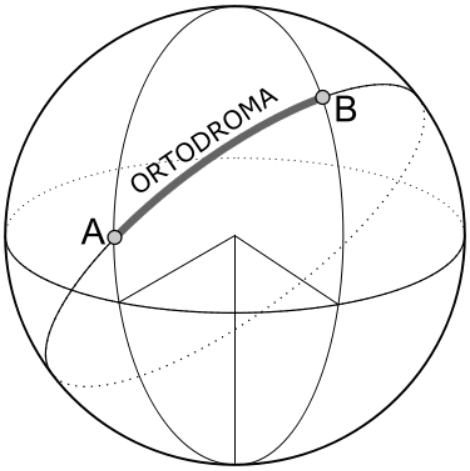



Figura 3.36: Distancia ortodrómica entre dos puntos sobre la superficie de una esfera. [16]

```
4     lat_second = toRadians(Math.abs(lat_second));
5     lon_second = toRadians(Math.abs(lon_second));

7     let lat_distance = lat_first - lat_second;
8     let lon_distance = lon_first - lon_second;

10    let a = Math.pow(Math.sin(lat_distance / 2), 2) + Math.cos(lat_second) * Math.cos(lat_first) *
          Math.pow(Math.sin(lon_distance / 2), 2);
11    let c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

13    // El valor 6373.0 es el radio de la tierra
14    return 6373.0 * c;
15  }
```

## 3.6 Organización del proyecto en paquetes de trabajo

Para la consecución del presente proyecto se definen un total de cuatro PT, cada uno de los cuales está dividido en una serie de Tareas (T). En el diagrama de Gantt de la Figura 3.37, se puede observar la descomposición temporal del proyecto en paquetes de trabajo y tareas.

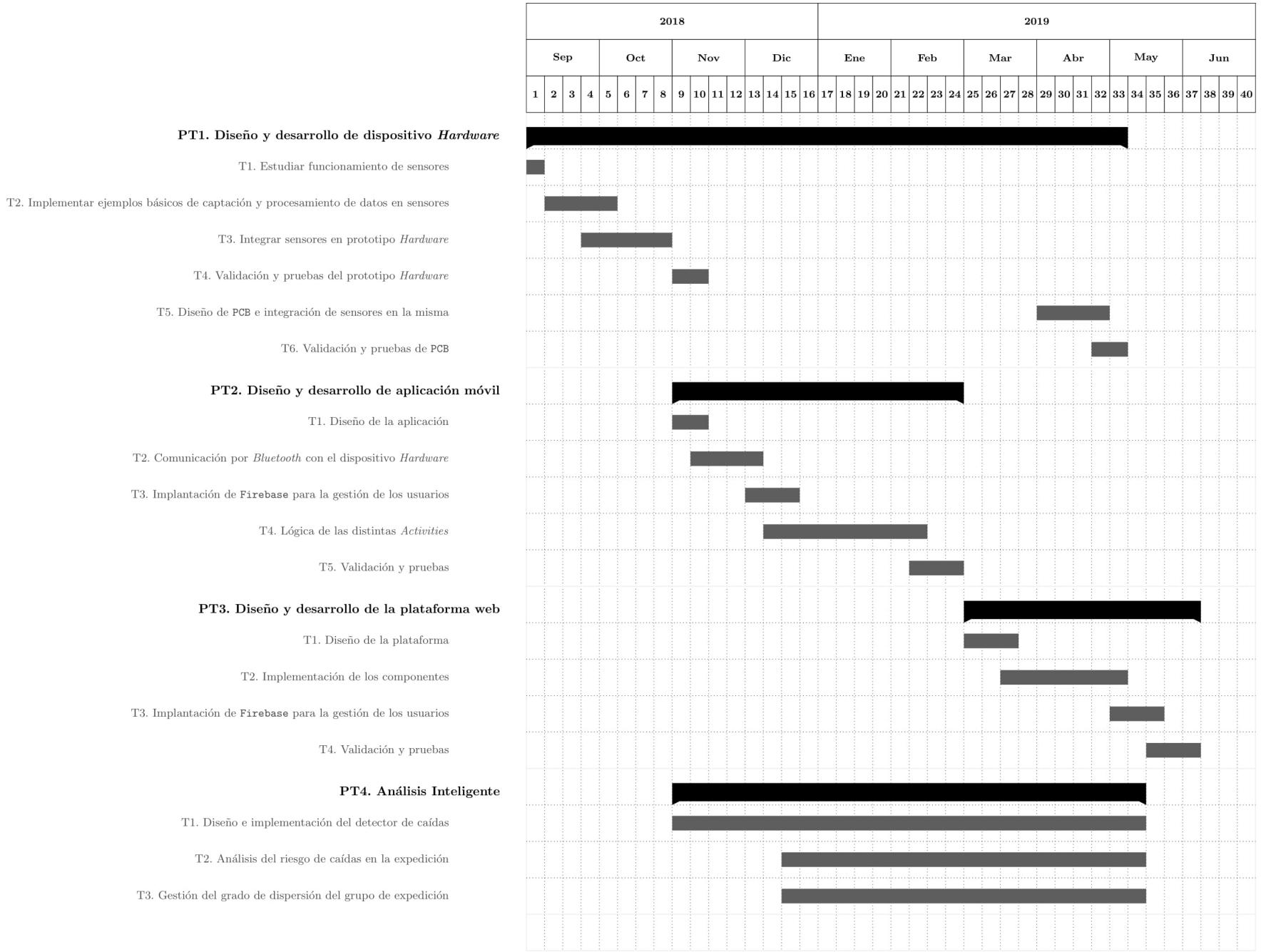



Figura 3.37: Diagrama de Gantt.

### 3.6.1 PT1. Diseño y desarrollo de dispositivo *Hardware* (Semanas 1 a 33)

Este paquete de trabajo engloba todo lo relacionado con el dispositivo físico, con el principal objetivo de obtener un prototipo del dispositivo *hardware* capaz de tomar los datos de los sensores necesarios y enviarlos a un dispositivo móvil, por medio de *Bluetooth*. El trabajo en las tareas que conforman este paquete comenzó en la primera semana del proyecto y comprendió hasta la semana 33. Se obtuvo una versión del prototipo válida en la semana 10, pero semanas mas tarde se diseño la PCB en la que se integraron los sensores del dispositivo, mejorando el diseño del prototipo final.

El PT1 se compone de las siguientes tareas:

- **T1 (Semana 1). Estudiar funcionamiento de sensores.** Esta tarea está diseñada para conocer de forma más profunda los sensores que se van a manejar en el proyecto. Es una tarea importante ya que conocer el funcionamiento de los sensores hace más sencilla la interpretación de los valores de los mismos, así como la identificación de un posible comportamiento anómalo, en un momento dado.
- **T2 (Semanas 2 a 5). Implementar ejemplos básicos de captación y procesamiento de datos en sensores.** A lo largo de esta tarea se realizaron varios ejemplos de código en los que se utilizaban los sensores de forma individual (captación de datos en el caso de sensores de temperatura, humedad y acelaración entre otros, y emisión de datos en el caso del módulo *Bluetooth*). En esta tarea se realizó la búsqueda de librerías existentes que gestionasen los sensores o se implementaron librerías propias para los sensores que no tuviesen librerías disponibles.
- **T3 (Semanas 4 a 8). Integrar sensores en prototipo *Hardware***. Durante esta tarea se tomaron todos los ejemplos individuales de uso de sensores de la tarea anterior y se creó un solo *sketch* que integraba todos los sensores.
- **T4 (Semanas 9 y 10). Validación y pruebas del prototipo *Hardware***. Esta tarea sirvió para identificar los fallos en el prototipo *hardware* y corregirlos, obteniendo una versión estable del mismo.
- **T5 (Semanas 29 a 32). Diseño de PCB e integración de sensores en la misma.** Durante esta tarea se diseñó y se construyó la PCB (por una empresa que se dedica a la impresión de PCBs) y se procedió a soldar todos los sensores en la misma.
- **T6 (Semanas 31 a 33). Validación y pruebas de PCB.** A lo largo de esta tarea se realizaron pruebas para cerciorarse de la correcta conexión de los sensores en la PCB, del correcto diseño de la PCB y del correcto funcionamiento del prototipo *hardware*.

El resultado obtenido del PT1 es la creación de un dispositivo *hardware* equipado con sus correspondientes sensores que sea capaz de tomar datos de los mismos y comunicarlos a un dispositivo móvil a través de *Bluetooth*.

### 3.6.2 PT2. Diseño y desarrollo de aplicación móvil (Semanas 9 a 24)

Este paquete de trabajo tiene como responsabilidad el diseño y desarrollo de una aplicación móvil que tome datos de un microcontrolador a través de *Bluetooth* y muestre información de manera amigable a los usuario de la misma.

El PT2 está compuesto por las siguientes tareas:

- **T1 (Semanas 9 y 10). Diseño de la aplicación.** A lo largo de esta tarea se realizaron los bocetos que darían lugar a las distintas pantallas de las que se compone la aplicación. Durante esta tarea también se llevó a cabo el diseño de la arquitectura MVC de la aplicación móvil.
- **T2 (Semanas 10 a 13). Comunicación por *Bluetooth* con el dispositivo *Hardware*.** Durante esta tarea se llevó a cabo la conexión de la aplicación móvil con el dispositivo físico, usando el protocolo de comunicación inalámbrico *Bluetooth*. Se desarrollaron distintas implementaciones de esta conexión hasta que se obtuvo una implementación que cumplía con una frecuencia de lectura de datos suficiente para realizar la detección de caídas.
- **T3 (Semanas 13 a 15). Implantación de `Firebase` para la gestión de los usuarios.** En esta tarea se realizó la creación y autenticación de usuarios en la aplicación móvil. También se realizó la conexión con la base de datos en tiempo real `Firestore` para almacenar los datos relacionados con las expediciones.
- **T4 (Semanas 14 a 22). Lógica de las distintas *Activities*.** A lo largo de esta tarea se realizó la implementación de las actividades que conforman la aplicación móvil. Esta implementación no incluye el análisis inteligente de los datos, que comprende un paquete de trabajo completo.
- **T5 (Semanas 22 a 24). Validación y pruebas.** Durante esta tarea se llevaron a cabo una serie de pruebas de la aplicación móvil, para identificar los fallos de implementación en la misma y corregirlos.

El resultado obtenido del PT2 es una aplicación móvil para el sistema operativo *Android*, que tome datos de un microcontrolador a través de *Bluetooth* y muestre información de manera amigable a los usuario de la misma.

### 3.6.3 PT3. Diseño y desarrollo de la plataforma *web* (Semanas 25 a 37)

Este paquete de trabajo tiene como responsabilidad el diseño y desarrollo de una plataforma *web* que se encargue de mostrar información al usuario en tiempo real de la expedición que está realizando. De la misma manera, se encargará de mostrar un histórico de expediciones realizadas por el usuario, con los datos de las mismas.

El PT3 se compone de las siguientes tareas:

- **T1 (Semanas 25 a 27). Diseño de la plataforma.** Durante esta tarea se llevó a cabo la realización de bocetos que darían lugar a las distintas páginas de las que se compone la plataforma *web*. En esta tarea también se definió la arquitectura que se ha usado al implementar la plataforma.
- **T2 (Semanas 27 a 33). Implementación de los componentes.** Esta tarea está diseñada para llevar a cabo la codificación de los distintos componentes `React` que conforman el *frontend* de la plataforma.
- **T3 (Semanas 33 a 35). Implantación de `Firebase` para la gestión de los usuarios.** A lo largo de esta tarea se llevó a cabo la autenticación y gestión de los usuarios en la plataforma *web*. En esta tarea también se desarrolló el servidor `Express` y los distintos controladores del *backend*.
- **T4 (Semanas 35 a 37). Validación y pruebas.** Durante esta tarea se llevaron a cabo una serie de pruebas de la plataforma *web*, para identificar los fallos de implementación en la misma y corregirlos.

El resultado obtenido del PT3 es una plataforma *web*, usando `React` y `Node` como bibliotecas y *frameworks* de desarrollo, que sirve para visualizar (en tiempo real y de forma histórica) información avanzada acerca de una expedición de la que el usuario ha tomado o esta tomando parte.

### 3.6.4 PT4. Análisis Inteligente(Semanas 9 a 34)

Este paquete de trabajo tiene como responsabilidad la realización de un análisis inteligente a partir de los datos recogidos por los sensores del dispositivo físico (y otros datos captados gracias al dispositivo móvil). Este análisis inteligente es realizado por la aplicación móvil.

El PT4 se compone de las siguientes tareas:

- **T1 (Semanas 9 a 34). Diseño e implementación del detector de caídas.** Esta tarea sirvió para diseñar y desarrollar un algoritmo de detección de caídas usando datos de aceleración, siguiendo un enfoque basado en límites o umbrales. Durante esta tarea se realizaron pruebas con distintos valores de umbrales para conseguir minimizar los falsos positivos y falsos negativos.
- **T2 (Semanas 15 a 34). Análisis del riesgo de caídas en la expedición.** A lo largo de esta tarea se realizó un algoritmo que utiliza lógica difusa para, a través de unos valores de entrada proporcionados por los sensores del dispositivo físico, inferir un grado de riesgo de caída en la expedición.
- **T3 (Semanas 15 a 34). Gestión del grado de dispersión del grupo de expedición.** Durante esta tarea se realizó un algoritmo que analiza el grado de dispersión de los

usuarios de la expedición con respecto al guía, para prevenir posibles situaciones de riesgo en la expedición.

Los resultados obtenidos del PT4 han sido una serie de algoritmos implementados en la aplicación móvil con el objetivo común de mejorar la seguridad de los usuarios en una expedición y ayudar al guía proporcionándole una mayor visión de la expedición en todo momento.

## 3.7 Metodología de trabajo

Para el desarrollo del proyecto, se ha optado por seguir una metodología basada en un desarrollo iterativo e incremental [Cab16b] ya que esta metodología se adaptaba perfectamente a este proyecto. Para llevarlo a cabo, se han seguido una secuencia de pasos no lineales haciendo que cada poco tiempo se tenga una versión operativa del producto final. En este TFG es muy importante esta característica, ya que era necesario tener una versión operativa del *hardware* usado y el *firmware* para la comunicación con el dispositivo físico antes de comenzar con otras fases como el análisis inteligente. Las posteriores iteraciones se han usado para, prioritariamente, corregir fallos de iteraciones anteriores. En nuevas iteraciones también se han añadido funcionalidades nuevas, aumentando la calidad del producto final. Al final de cada iteración se ha mantenido una reunión con el director del proyecto, que ha identificado fallos en el trabajo realizado hasta el momento, ha propuesto mejoras sobre el trabajo existente y nuevas funcionalidades a añadir.

El desarrollo del proyecto ha estado motivado por una metodología ágil [Cab16a]. Cada iteración se ha dividido en una lista de tareas, a las que se les ha dado una prioridad antes de comenzar la iteración. Estas tareas han sido de corta duración (de una a cuatro semanas, a lo sumo), motivadas por la entrega de *software* funcional y que aporta valor. La comunicación con el director del proyecto ha sido continua durante la ejecución de las tareas. Durante los primeros meses del proyecto se mantuvieron reuniones presenciales con el director del mismo cada semana (o a lo sumo cada dos semanas). Durante los meses finales las reuniones fueron a través de videoconferencia cada dos semanas o cuando se encontraba alguna dificultad en la consecución del proyecto. Se ha usado un método basado en un tablero *Kanban* para gestionar el tiempo y visualizar en todo momento qué tareas de la iteración se han realizado y cuáles faltan por realizar (ver Figura 3.38).

También se ha usado `git` como sistema de control de versiones, para mantener un registro detallado de qué cambios se realizan en cada uno de los archivos del repositorio creado para este proyecto. El repositorio `git` se ha alojado en `GitHub`, como se puede observar en la Figura 3.39.


Figura 3.38: Tablero de `Trello`[17] usado durante el TFG.

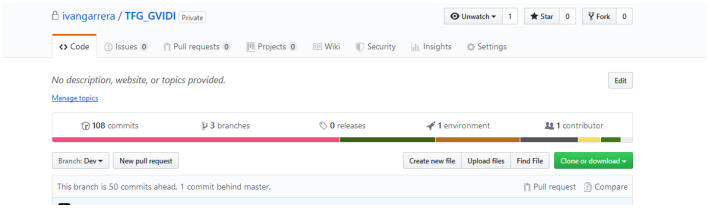



Figura 3.39: Repositorio `git` del proyecto, alojado en `GitHub`.[18]

### 3.7.1 Iteraciones

Este proyecto está compuesto por un total de diez iteraciones. Las primeras iteraciones se centran en el análisis del alcance del proyecto, la recogida de requisitos y la búsqueda del *hardware* necesario para construir un prototipo del dispositivo *hardware*. A continuación, las iteraciones se centran en la mejora del prototipo *hardware*, el diseño y desarrollo de la aplicación móvil y la redacción del capítulo de antecedentes del presente TFG. Las últimas iteraciones se centran en las pruebas de la aplicación móvil, el diseño y creación de la PCB para mejorar el dispositivo físico, el diseño, desarrollo y pruebas de la plataforma *web* y la redacción de esta memoria. A continuación se van a detallar cada una de las diez iteraciones que se han realizado:

#### Iteración 1

En esta iteración se llevaron a cabo las tareas relacionadas con la recogida de requisitos, para definir qué detalles específicos se van a llevar a cabo durante la consecución del proyecto. Una vez recogidos los requisitos se definió el alcance del proyecto, para tener una idea del tiempo y los recursos que se emplearían. Durante esta iteración también se llevó a cabo la realización del anteproyecto.

Los requisitos que se obtuvieron fueron:

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| Documentación | | - Recogida de requisitos<br>- Definición del alcance del proyecto<br>- Realización del anteproyecto | 01/08/ 2018 | 31/08/ 2018 | 30h |

Cuadro 3.9: Descripción resumida de la primera iteración.

- El dispositivo *hardware* será capaz de obtener datos de los sensores necesarios y los enviará a un dispositivo móvil por *Bluetooth*.
- El dispositivo *hardware* debe estar equipado con botones físicos para generar alertas.
- Se diseñará una PCB para mejorar el prototipo *hardware*.
- La aplicación móvil se realizará usando el sistema operativo `Android`.
- La aplicación móvil será capaz de recibir los datos provenientes del microcontrolador por medio de *Bluetooth*.
- La aplicación móvil debe proporcionar información distinta y relevante para guías y para participantes de las distintas expediciones.
- Tanto el guía como los participantes deben poder visualizar en un mapa su situación en el grupo de expedición.
- El dispositivo móvil será capaz de usar la información proveniente del microcontrolador para detectar caídas en los miembros y guía del grupo de expedición.
- El dispositivo móvil debe inferir un grado de peligrosidad usando la información de los sensores, que el guía y los participantes pueden usar para maximizar su precaución.
- El dispositivo móvil le proporcionará al guía información sobre la distancia a la que se encuentran los participantes de la expedición, así como información sobre el grado de dispersión del grupo.
- La autenticación en la aplicación móvil y en la plataforma *web* se realizará usando `Firebase`, para maximizar la seguridad durante el proceso de autenticación.
- La plataforma *web* tendrá un diseño *responsive*, que se adaptará al tamaño de pantalla del dispositivo dónde se visualice.
- La plataforma *web* mantendrá un histórico de todas las expediciones en las que ha tomado parte un determinado participante o guía, con información relevante sobre ellas.
- Debe ser posible que una persona autorizada por el participante en la expedición pueda seguir en tiempo real el transcurso de la misma.

**Iteración 2**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT1 | T1,T2,T3 | - Compra de componentes que conformarían el dispositivo físico<br>- Búsqueda o realización de librerías para los sensores | 02/09/ 2018 | 28/10/ 2018 | 35h |

Cuadro 3.10: Descripción resumida de la segunda iteración.

Durante esta iteración se realizó una búsqueda de todos los sensores que se necesitarían en el proyecto (véase Tabla 3.1) y se procedió a la compra de los mismos gracias al grupo de investigación AIR[19]. Después de estudiar distintas alternativas en lo que a los microcontroladores se refiere (por ejemplo `Raspberry Pi`, `STM32` o `Arduino`), finalmente se eligió el último de los nombrados por la rapidez de prototipación, la gran disponibilidad de librerías para muchos sensores y las necesidades básicas de este proyecto que no requerían un microcontrolador de mayores prestaciones.

En esta iteración también se buscaron librerías existentes para los sensores que se iban a utilizar. En muchos de los casos fue necesario adaptar estas librerías para este proyecto en concreto o, en algunos casos, crear por completo una nueva librería.

**Iteración 3**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT2 | T1 | - Capítulo de antecedentes<br>- Ensamblado de sensores en microcontrolador<br>- Diseño de la app móvil | 05/11/ 2018 | 16/11/ 2018 | 35h |
| Documentación | | | | | |

Cuadro 3.11: Descripción resumida de la tercera iteración.

A lo largo de esta iteración se realizó el capítulo de antecedentes del presente proyecto. Se investigaron las tecnologías que se han usado en el proyecto, alternativas a estas tecnologías

[19] `http://air.esi.uclm.es/`

y se buscó información acerca de otros proyectos o sistemas con una funcionalidad igual o parecida a la que se expone en el presente proyecto. También se realizaron bocetos del diseño de la aplicación móvil.

Durante esta iteración también se procedió a la integración de todos los sensores para generar un primer prototipo muy básico del dispositivo *hardware*, con el objetivo de poder comenzar en la siguiente iteración con la aplicación móvil.

**Iteración 4**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT1<br><br>PT2 | T3, T4<br><br>T2 | - Implementar comunicación *Bluetooth* en la app móvil<br>- Finalizar el prototipo del dispositivo físico | 19/11/ 2018 | 30/11/ 2018 | 25h |

Cuadro 3.12: Descripción resumida de la cuarta iteración.

Durante esta iteración se implementó parte de la funcionalidad de la aplicación móvil. Se realizaron los primeros pasos para comunicar el dispositivo *hardware* con la aplicación móvil usando *Bluetooth*. Se siguió mejorando el prototipo *hardware* y la integración de algunos sensores.

**Iteración 5**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT2 | T3, T4 | - Integración de `Firebase` en la app móvil<br>- Primera versión de un algoritmo de detección de caídas<br>- Separación de información por roles | 04/12/ 2018 | 08/02/ 2019 | 60h |
| PT4 | T1 | | | | |

Cuadro 3.13: Descripción resumida de la quinta iteración.

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT1 | T5 | | | | |
| PT2 | T4 | | | | |
| PT3 | T1 | | | | |
| PT4 | T1, T2,T3 | - Segunda versión del algoritmo de detección de caídas<br>- Análisis del riesgo de caída en la expedición<br>- Diseño de app *web*<br>- Diseño y fabricación de PCB | 11/02/ 2019 | 22/03/ 2019 | 60h |

Cuadro 3.14: Descripción resumida de la sexta iteración.

En esta iteración se realizó el proceso de autenticación de usuarios en el dispositivo móvil, se mejoró la comunicación por *Bluetooth* entre el dispositivo *hardware* y el dispositivo móvil y se comenzó con el análisis inteligente de la información recibida, implementando una primera aproximación de un detector de caídas. Durante esta iteración también se realizó la separación de la información que se iba a mostrar dependiendo del rol del usuario.

**Iteración 6**

Durante esta iteración se mejoró el detector de caídas existente y se prosiguió con la lógica de la aplicación móvil y el análisis inteligente de la información (se implementaron las alertas por distancia en la expedición, se determinó el nivel de agrupación de la expedición y se realizó un primer algoritmo sencillo para determinar el riesgo en la ruta).

En esta iteración se realizó también el prototipo de diseño de la aplicación *web* y se investigaron las tecnologías que se iban a usar tanto en *frontend* como en *backend* (principalmente `React` y `NodeJS` respectivamente) y cómo se podrían combinar para realizar la plataforma *web*.

En esta iteración se realizó también el diseño de la PCB que se ha usado en el prototipo *hardware* y se pidió la fabricación de la misma a una empresa externa.

**Iteración 7**

Durante esta iteración se soldaron los sensores y componentes *hardware* necesarios en la PCB y se probó el correcto funcionamiento de la misma. Se hizo uso de la memoria EEPROM del microcontrolador para almacenar información cuya persistencia en el tiempo era necesaria.

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT1 | T5, T6 | | | | |
| PT2 | T5 | | | | |
| PT3 | T2 | - Ensamblado de la PCB<br>- Versión estable de app móvil<br>- Componentes de la vista de la *web*<br>- Creación del servidor *backend* de la *web* | 25/03/ 2019 | 12/04/ 2019 | 40h |

Cuadro 3.15: Descripción resumida de la séptima iteración.

También se corrigieron fallos en la aplicación móvil y se completó la funcionalidad que faltaba en la misma, dando como resultado una primera versión completa y estable de la aplicación móvil.

En esta iteración se crearon visualmente los componentes *React* que se iban a usar en la aplicación *web* y se integró el *backend* y el *frontend* de la misma.

**Iteración 8**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT3 | T2, T3 | - Lógica de la aplicación *web* | 22/04/ 2019 | 03/05/ 2019 | 40h |

Cuadro 3.16: Descripción resumida de la octava iteración.

Durante esta iteración se integró el sistema de autenticación en la plataforma *web*, de tal forma que la *web* y la aplicación móvil quedan unificadas bajo el mismo sistema de autenticación. Se añade la lógica a los componentes *React* creados en la iteración anterior. A lo largo de esta iteración se añade la posibilidad de que los familiares o amigos del participante en la ruta de expedición puedan visualizar en tiempo real el estado de dicho participante en un mapa.

**Iteración 9**

En esta iteración se corrigieron fallos en la plataforma *web*, dando como resultado una primera versión completa de la *web*. A lo largo de la iteración, se realizaron pruebas reales del dispositivo, realizando rutas con una duración en torno a una hora. Gracias a estas pruebas,

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| PT3 | T4 | - Versión operativa de la plataforma *web*<br>- Pruebas reales del proyecto | 20/05/ 2019 | 21/06/ 2019 | 90h |
| PT4 | T1, T2, T3 | | | | |

Cuadro 3.17: Descripción resumida de la novena iteración.

se detectaron más fallos (en la aplicación móvil y en la *web*, principalmente) que deberán ser corregidos.

**Iteración 10**

| Paquetes de Trabajo | Tareas a las que afecta | Objetivos alcanzados | Fecha Inicio | Fecha Fin | Tiempo Estimado |
|---|---|---|---|---|---|
| Documentación | | - Correción de errores en el proyecto<br>- Documentación del proyecto | 08/05/ 2019 | 17/05/ 2019 | 90h |
| PT1 | T6 | | | | |
| PT2 | T5 | | | | |
| PT3 | T4 | | | | |

Cuadro 3.18: Descripción resumida de la décima iteración.

En esta última iteración, se corrigieron los fallos detectados durante las pruebas reales del dispositivo y se procedieron a realizar más pruebas para probar que el sistema funcionaba correctamente. Con la realización de estas pruebas se genera una versión final de `GVIDI`.

## 3.8 Características *hardware* y *software* del desarrollo

A lo largo de esta sección se van a exponer los componentes *hardware* y *software* que se han empleado para la consecución de este proyecto.

### 3.8.1 Medios *hardware*

Los componentes *hardware* que se han usado a lo largo de el proyecto son los siguientes:

- **Equipo de trabajo**. Ordenador portátil `Lenovo Idea G50-80`, con la siguientes características principales:
  - Procesador `Intel Core i7-5500U`.

  - Memoria *Random Access Memory* (RAM) de 8GB de capacidad.
  - 1TB de almacenamiento HDD y 500GB de almacenamiento SSD.
  - Tarjeta gráfica `AMD Radeon HD 8690M`.
  - *Dual boot* (`Microsoft Windows 10` y `Ubuntu 18.04`)

- **Dispositivo móvil**, necesario para realizar las pruebas de la aplicación `Android` del proyecto. Tiene las siguientes características:
  - Marca y modelo: `Samsung Galaxy J3` (2016).
  - Procesador: `Spreadtrum SC7731` *quad-core* a 1,5 `GHz`.
  - 8 `GB` de almacenamiento interno.
  - 1,5 `GB` de memoria RAM.
  - Sistema operativo: `Android` versión 5.1.1
- **Microcontrolador**. `Arduino Nano`, basado en el microprocesador `ATmega328`.
- **Sensor de GPS**, modelo `NEO-6M`.
- **Sensor de pulso cardíaco**, modelo `PULSESENSOR`.
- **Acelerómetro y giroscopio**, modelo `MPU6050`.
- **Sensor de temperatura y humedad**, modelo `DHT22`.
- **Sensor de lluvia**, modelo `YL-83`.
- **Sensor de luminosidad**, modelo `GL5528`.
- **Módulo de *Bluetooth***, modelo `HC-06`.

### 3.8.2 Medios *software*

A continuación se muestran las herramientas *software* usadas durante este proyecto:

#### Sistemas Operativos

- **Windows 10**. Es el sistema operativo en el que se ha desarrollado la aplicación móvil, el código fuente que gestiona los distintos sensores integrados en el dispositivo físico y además, el sistema operativo usado para escribir este documento.
- **Ubuntu 18.04**. Este sistema operativo se ha usado para desarrollar la plataforma *web*. Se escogió este sistema operativo porque la instalación y configuración de `node` en `Windows` es más compleja.
- **Android 5.1.1**. La aplicación móvil ha sido desarrollada y probada sobre un dispositivo móvil cuyo sistema operativo era el aquí indicado.

**Herramientas de desarrollo**

- **Atom**. Este editor de código fuente se ha usado para llevar a cabo el desarrollo de la plataforma *web*, sobre un sistema operativo basado en `Linux`.
- **Android Studio**. Es el entorno de desarrollo integrado oficial para desarrollar aplicaciones `Android`, por lo que se ha decidido usarlo para desarrollar la aplicación móvil del presente proyecto.
- **Arduino `IDE`**. Este entorno de desarrollo hace que la carga del *software* desarrollado para el microcontrolador `Arduino` se lleve a cabo de forma muy sencilla. Es la principal razón por la que se ha decidido usarlo en este proyecto.
- **Git**. Se trata del *software* de control de versiones usado a lo largo del proyecto. Se ha usado para gestionar los distintos cambios en el código fuente y en la documentación.
- **Npm** es el sistema de gestión de paquetes que usa `NodeJs`. Ha permitido facilitar la instalación de los paquetes externos necesarios durante el desarrollo de la plataforma *web*.
- **Nodemon** es una herramienta *software* que ha ayudado durante el desarrollo de la plataforma *web* reiniciando el servidor cada vez que se producen cambios en el código fuente, de forma automática.

**Tecnologías usadas**

- **Javascript** ha sido el lenguaje elegido para el desarrollo *web*, tanto en el *frontend* mediante el uso de `React` como en el *backend* con el uso de `Node`.
- **Java** ha sido el lenguaje elegido para el desarrollo de la aplicación `Android`.
- **C++** ha sido usado durante el desarrollo del código para el microcontrolador `Arduino`. Se han desarrollado en este lenguaje algunas librerías para gestionar los sensores usados. El código fuente de la solución que se carga en el microcontrolador no es exactamente `C++` (es código `Arduino`), pero está basado en este lenguaje.
- **Bootstrap**. Esta biblioteca ha sido la elegida para gestionar el estilo de la plataforma *web*, debido a que se puede crear un diseño*responsive* de la *web* de una forma sencilla.

**Documentación**

- **Texmaker** ha sido el editor de texto LaTeXque se ha utilizado para la confección del presente documento.
- **draw.io** se trata de una *web* en la que se pueden crear diagramas *online* de forma gratuita. Ha sido la principal herramienta de creación de imágenes y diagramas en este proyecto.

- **Adobe Illustrator** también ha sido usado para llevar a cabo los diagramas más complejos del proyecto.

Capítulo 4

# Resultados

En este capítulo se realizará un análisis para mostrar el cumplimiento de los objetivos específicos que se propusieron en el Capítulo 2.

## 4.1 Desarrollo de un dispositivo *hardware* equipado con distintos sensores.

El primer objetivo específico de este proyecto era el diseño y desarrollo de un dispositivo *hardware*, que equipase varios sensores necesarios para monitorizar un grupo de expedición. En la Figura 4.1 se puede observar cómo el microcontrolador capta los datos de los sensores que equipa, que serán enviados al dispositivo móvil a través del módulo *Bluetooth* conectado al dispositivo.

En las Figuras 3.11 y 3.12 se puede observar la PCB que ha sido diseñada especialmente para este proyecto y todos los componentes soldados en esta placa.

## 4.2 Desarrollo de una aplicación móvil.

Se puede observar el cumplimiento del objetivo de diseño y desarrollo de una aplicación móvil que recibe datos del microcontrolador a partir del protocolo *Bluetooth* en las Figuras 3.14, 3.15 y 3.16 de la Sección 3.3.2, en las que se aprecian las distintas pantallas que conforman la aplicación móvil, con información relevante tanto para los participantes como para el guía de la expedición. La mayor parte de la información se capta a través de *Bluetooth*, proveniente de los sensores del microcontrolador. La aplicación móvil cumple con toda la funcionalidad que se esperaba desde el comienzo de este proyecto.

## 4.3 Detección inteligente de situaciones anómalas en una expedición.

Uno de los objetivos principales era el análisis inteligente y en tiempo real de los datos que se captan a través de los sensores que el microcontrolador equipa. Este objetivo engloba tres subobjetivos:

- **Detección de caídas.**
- **Análisis del grado de dispersión del grupo de expedición.**

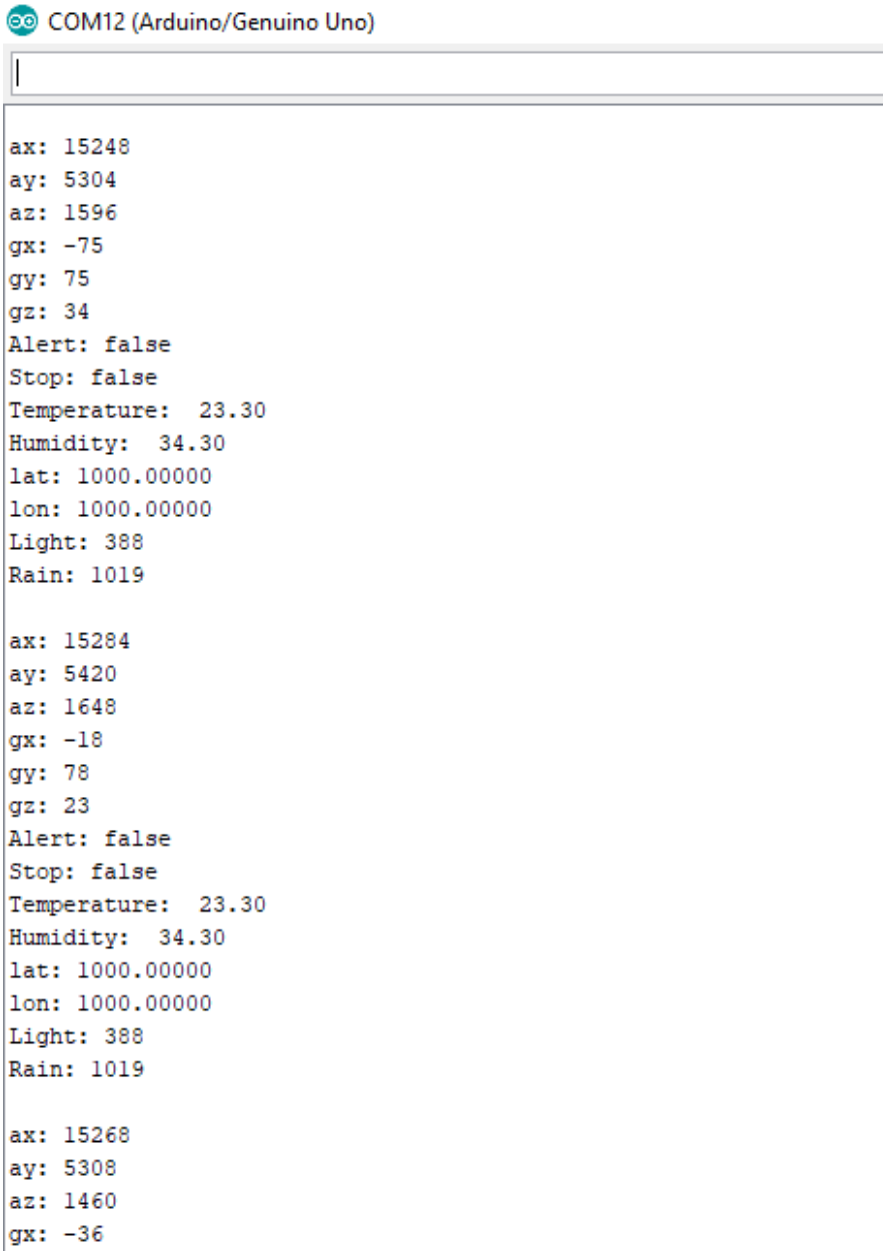



Figura 4.1: Datos de los sensores equipados en el microcontrolador `Arduino`.

- **Análisis del riesgo de caída.**

A continuación se muestran los resultados que demuestran el cumplimiento de los subobjetivos anteriores:

### 4.3.1 Cumplimiento del objetivo de detección de caídas.

La detección de caídas parecía ser un objetivo muy factible al inicio del proyecto, utilizando los datos de aceleración lineal y velocidad angular. Pronto se eliminaron los datos de velocidad angular del algoritmo de detección de caídas ya que el dispositivo físico iba a ser portado en la muñeca o en el brazo del usuario de la expedición y los datos de velocidad an-

gular con el microcontrolador en esa situación no eran de utilidad porque el usuario durante una caída puede mover el brazo y la mano de distintas formas.

Por tanto, el algoritmo de detección de caídas de este proyecto está muy limitado debido a que solo infiere la caída basándose en los datos de aceleración lineal. Además, el procesamiento de los datos de aceleración para inferir las caídas son realizados por la aplicación móvil, a partir de datos que recibe por *Bluetooth* desde el microcontrolador. Este hecho se trata de otra limitación, ya que la frecuencia de captación de datos del sensor por parte del microcontrolador es mayor que la frecuencia de envío de los datos por *Bluetooth*, es decir, se leen más datos del sensor de los que se envían por *Bluetooth*. Al ser *Bluetooth* un protocolo inalámbrico, la aplicación móvil en ocasiones recibe datos corruptos, que tiene que descartar, por lo que existen más pérdidas de valores leídos (cabe destacar que el número de paquetes corruptos es muy bajo, pero es necesario comentar que existe dicha situación).

Como última situación desfavorable hay que señalar que la aplicación móvil (como se ha comentado en el capítulo anterior) crea un nuevo hilo por cada paquete de datos que recibe por medio de *Bluetooth*. Esta medida es necesaria, para que la frecuencia de recepción de datos del *socket Bluetooth* sea mayor y pueda realizarse el algoritmo de detección de caídas. Sin embargo, se pueden dar situaciones en las que un paquete con datos de aceleración lineal de un instante posterior sea procesado antes que un paquete con datos de un instante anterior (debido al planificador del sistema operativo del dispositivo móvil). Se podría, por tanto, dar una situación en la que se reciban datos de normalidad (de instantes anteriores) después de procesar datos de caída libre (de instantes posteriores) y la detección de una posible caída fallaría. Esta situación ha sido contemplada y se ha mitigado (como se explicó en el capítulo anterior) aunque es una posible fuente de errores porque no se conoce con certeza cuándo se ejecutará un hilo.

Para medir la precisión del detector de caídas se han realizado un total de cinco caídas de distintas maneras:

- Caída hacia delante.
- Caída hacia detrás.
- Caída lateral.

Además, aunque en este proyecto se pensó que el dispositivo físico debería estar situado en la muñeca del usuario de la expedición, se han realizado las mismas pruebas con el dispositivo situado en la cintura. Se ha hecho de esta forma porque un usuario durante una caída puede realizar movimientos del brazo y la muñeca que proporcionen valores de aceleración lineal que no tengan que ver con una caída, por lo que el algoritmo de detección de caídas basado en límites en la aceleración lineal no inferiría una caída.

Los resultados de las pruebas se pueden observar en la Tabla 4.1. Solo un **33.33 % de todas las caídas fueron detectadas**, cuando el dispositivo era portado en la muñeca del

usuario. Si el dispositivo era portado en la cintura del usuario, este porcentaje **subía hasta el 66.66 %**. El tipo de caída que fue detectado con mayor éxito fue la caída hacia detrás, ya que la forma en la que se cae se asemeja más a la ideal. Además, en este tipo de caída los brazos suelen acompañar al movimiento que hace el cuerpo al contrario que en la caída hacia delante en la que los brazos realizan movimientos más bruscos (para intentar plantar las manos en el suelo y minimizar los daños que la caída pudiese producir) que conllevan variaciones en la aceleración lineal y desviaciones con respecto a la caída ideal.

El dispositivo se ha probado en situaciones reales, durante expediciones en la naturaleza de distinta dificultad y en distintos terrenos como el asfalto, pistas de tierra y senderos con mucha piedra y desnivel. No se ha detectado ninguna caída realizando dichas expediciones por lo que se han producido un **0 % de falsos positivos**. Esto mejora mucho los resultados del detector de caídas ya que si el guía recibe una alerta de caída puede estar casi convencido de que se trata de una alerta real y no un falso positivo.

Cabe desatacar también que el microcontrolador está equipado con dos botones físicos y uno de ellos genera una alerta por caída. De esta forma, si un usuario se encuentra consciente después de una caída puede pulsar este botón del dispositivo para generar la alerta por caída.

| | Caídas detectadas con el dispositivo en la muñeca | Caídas detectadas con el dispositivo en la cintura |
|---|---|---|
| **Caída hacia delante** | 1 de 5 | 3 de 5 |
| **Caída hacia detrás** | 3 de 5 | 4 de 5 |
| **Caída lateral** | 2 de 5 | 3 de 5 |

Cuadro 4.1: Precisión del algoritmo de detección de caídas, ante distintas pruebas.

### 4.3.2 Cumplimiento del objetivo de análisis del grado de dispersión en el grupo de expedición.

Para demostrar el correcto funcionamiento del algoritmo de análisis del grado de dispersión del grupo de expedición, se va a plantear una situación hipotética en la que existen un total de cuatro usuarios (un guía y tres participantes). Se compararán los resultados obtenidos por el algoritmo desarrollado en el presente proyecto, los resultados obtenidos por `Google Maps` y los obtenidos de forma manual. Los usuarios se encuentran en la ciudad de Ciudad Real (como se puede observar en la Figura 4.3).

Las coordenadas de los usuarios de esta expedición se pueden encontrar en la Tabla 4.2.

Se han usado dichos valores de entrada en el algoritmo implementado en este proyecto (véase Listado 4.1) para obtener el grado de dispersión del grupo. Los resultados de salida del algoritmo se pueden observar en la Figura 4.2.

| Usuario | Latitud en grados | Longitud en grados | Latitud en radianes | Longitud en radianes |
|---|---|---|---|---|
| Guía | 38.989 | -3.929 | 0.680486 | -0.06857 |
| Participante 1 | 38.981 | -3.925 | 0.6803467 | -0.068504 |
| Participante 2 | 38.982 | -3.930 | 0.6803642 | -0.068591 |
| Participante 3 | 38.980 | -3.934 | 0.6803293 | -0.68661 |

Cuadro 4.2: Coordenadas en grados y en radianes de los usuarios de la expedición hipotética planteada.

Listado 4.1: **Test de análisis del grado de dispersión en una situación hipotética planteada.**

```
1  public void TestDispersionGrupo() {
2    double lat_guide = 38.989; double lon_guide = -3.929;
3    double lat_p1 = 38.981; double lon_p1 = -3.925;
4    double lat_p2 = 38.982; double lon_p2 = -3.930;
5    double lat_p3 = 38.980; double lon_p3 = -3.934;

7    double d1 = getDistanceBetweenTwoPoints(Math.toRadians(lat_guide), Math.toRadians(lon_guide),
8              Math.toRadians(lat_p1), Math.toRadians(lon_p1));
9    double d2 = getDistanceBetweenTwoPoints(Math.toRadians(lat_guide), Math.toRadians(lon_guide),
10             Math.toRadians(lat_p2), Math.toRadians(lon_p2));
11   double d3 = getDistanceBetweenTwoPoints(Math.toRadians(lat_guide), Math.toRadians(lon_guide),
12             Math.toRadians(lat_p3), Math.toRadians(lon_p3));

14   double dispersion = (0.333333333) * (d1+d2+d3);

16   Log.d("GVIDI", "Distancia entre gu a y participante 1 = " + String.valueOf(d1));
17   Log.d("GVIDI", "Distancia entre gu a y participante 2 = " + String.valueOf(d2));
18   Log.d("GVIDI", "Distancia entre gu a y participante 3 = " + String.valueOf(d3));
19   Log.d("GVIDI", "Dispersi n del grupo = " + String.valueOf(dispersion));
20 }
```

```
Distancia entre guía y participante 1 = 0.9546824285282473
Distancia entre guía y participante 2 = 0.7833945137256746
Distancia entre guía y participante 3 = 1.0904240767455835
Dispersión del grupo = 0.9428336720570014
```

Figura 4.2: Salida del algoritmo que calcula el grado de dispersión de la expedición, para una situación hipotética planteada.

Las distancias que calcula `Google Maps` se pueden observar en la figura 4.3.

Las distancias calculadas de forma manual, usando la fórmula de *Harvesine* (que se encuentra en la Sección 3.5.2) son las siguientes:

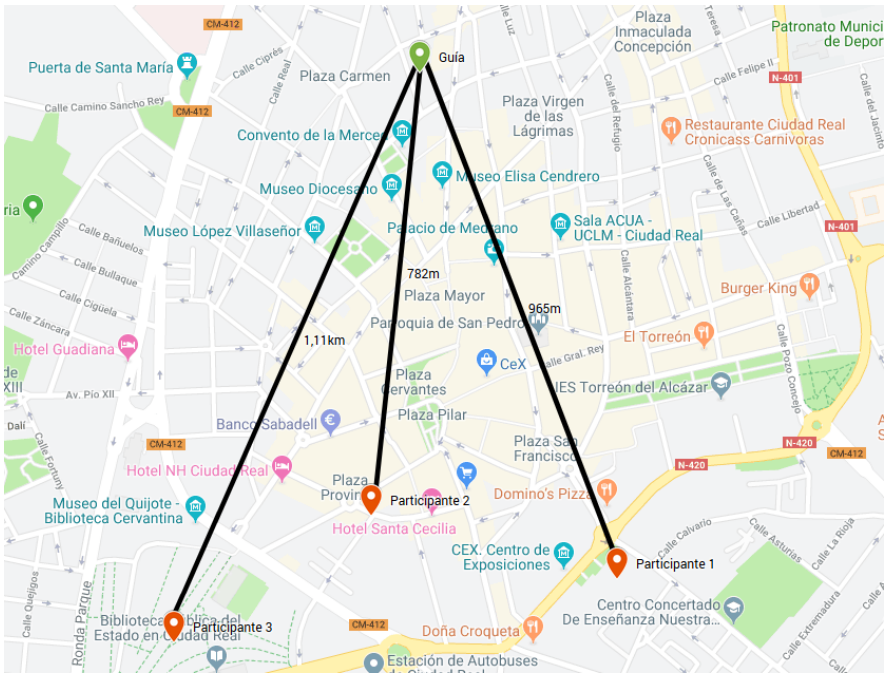



Figura 4.3: Localización de los usuarios de la expedición hipotética planteada.

$$\text{distancia(guía, participante1)} = 2 \cdot 6373 \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{0{,}6803467-0{,}680486}{2}\right) + \cos(0{,}680486) \cdot \cos(0{,}6803467) \cdot \sin^2\left(\frac{-0{,}068504+0{,}06857}{2}\right)}\right) \approx 1{,}10$$

$$\text{distancia(guía, participante2)} = 2 \cdot 6373 \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{0{,}6803642-0{,}680486}{2}\right) + \cos(0{,}680486) \cdot \cos(0{,}6803642) \cdot \sin^2\left(\frac{-0{,}068591+0{,}06857}{2}\right)}\right) \approx 0{,}797$$

$$\text{distancia(guía, participante3)} = 2 \cdot 6373 \cdot \arcsin\left(\sqrt{\sin^2\left(\frac{0{,}6803293-0{,}680486}{2}\right) + \cos(0{,}680486) \cdot \cos(0{,}6803293) \cdot \sin^2\left(\frac{-0{,}068661+0{,}06857}{2}\right)}\right) \approx 1{,}32$$

En la Tabla 4.3 se puede observar la comparativa entre los distintos grados de separación obtenidos en esta expedición hipotética usando el algoritmo desarrollado en este proyecto, la plataforma `Google Maps` y aplicando la fórmula descrita en la Sección 3.5.2. Se puede observar que los resultados del algoritmo propuesto y de `Google Maps` son prácticamente idénticos. La desviación existente con el cálculo manual radica principalmente en la precisión de los decimales que se usan, ya que las operaciones trigonométricas utilizadas (seno, arcoseno y coseno) son muy sensibles a cambios en los decimales.

### 4.3.3 Cumplimiento del objetivo de análisis del riesgo de caída.

Al igual que en el apartado anterior, se van a realizar un total de tres *tests* para demostrar que la salida del algoritmo de análisis del riesgo de caída, basado en el empleo de lógica difusa, es la esperada para unos ciertos valores de entrada. Cabe recordar que las variables de entrada de este algoritmo son:

| | Algoritmo del proyecto | De forma manual | Google Maps |
|---|---|---|---|
| Distancia(guía, participante1) | 0.955km | 1.1km | 0.965km |
| Distancia(guía, participante2) | 0.783km | 0.797km | 0.782km |
| Distancia(guía, participante3) | 1.09km | 1.32km | 1.11km |
| Grado de separación | 0.943km | 1.072km | 0.952km |

Cuadro 4.3: Comparativa del grado de separación obtenido de diversas maneras.

- La humedad que existe en la expedición.
- La presencia o ausencia de lluvia.
- La cantidad de luz que hay en un momento dado en la expedición.

Como única variable de salida se encuentra el riesgo de caída en un preciso instante.

***Test* 1.**

En la Tabla 4.4 se encuentran los valores que tomarán las distintas variables de entrada en el *Test* 1. También se puede observar a qué conjunto difuso pertenecen dichos valores.

| | Humedad | Lluvia | Luz |
|---|---|---|---|
| **Valor** | 90 % | 200 | 250 |
| **Conjunto difuso** | Muy alta | Sí | Oscuro |

Cuadro 4.4: Valores y conjuntos difusos de las variables de entrada del *Test* 1.

Para este primer *test*, se activaría la siguiente regla (del conjunto de reglas definidas en este proyecto, que se pueden consultar en la Sección 3.4.3:

- **R4.** Si lluvia es sí $\wedge$ (luz es oscuro $\vee$ luz es muy oscuro) $\rightarrow$ riesgo es **alto**

Si la cantidad de luz que hay en un momento determinado es baja, aumenta la dificultad para ver obstáculos en el camino y por tanto aumenta también el riesgo de caída. Si además está lloviendo, el terreno puede estar resbaladizo por lo que el riesgo de caída también aumenta. Ante una situación con una visibilidad baja y lluvia el riesgo de caída debe ser alto. Por lo tanto, la salida del algoritmo difuso para este *test* debería ser un riesgo alto de caída.

La detección del riesgo de caída se realiza usando únicamente tres variables de entrada por limitaciones económicas. Sin embargo este enfoque usando lógica difusa hace que sea muy sencillo añadir nuevos sensores que ayuden a realizar un análisis del riesgo de caída en un futuro, como pueden ser aquellos relacionados con la pendiente y dificultad del terreno.

***Test* 2.**

En la Tabla 4.5 se encuentran los valores que tomarán las distintas variables de entrada en el *Test* 2. También se puede observar a qué conjunto difuso pertenecen dichos valores.

| | Humedad | Lluvia | Luz |
|---|---|---|---|
| **Valor** | 40 % | 512 | 700 |
| **Conjunto difuso** | Media | 50 % Sí, 50 % No | Día |

Cuadro 4.5: Valores y conjuntos difusos de las variables de entrada del *Test* 2.

En este segundo *test*, se activarán las siguientes reglas:

- **R1.** Si lluvia es sí $\wedge$ (luz es media-alta $\vee$ luz es máxima) $\wedge$ (humedad es muy baja $\vee$ humedad es baja $\vee$ humedad es media) $\rightarrow$ riesgo es **bajo**
- **R6.** Si lluvia es no $\wedge$ (luz es media-alta $\vee$ luz es máxima) $\rightarrow$ riesgo es **bajo**

En este caso en vez de activarse tan sólo una regla se activan un total de dos reglas, ya que existe un 50 % de posibilidades de que esté lloviendo. En este proyecto, se usa el producto algebraico como método de conjunción entre reglas. Esto es, las distintas funciones de pertenencia a los conjuntos difusos se multiplican por el grado de cumplimiento de la condición y por último se combinan. En este caso, como la salida de ambas reglas es un riesgo bajo, el resultado del algoritmo debería ser un riesgo bajo de caída.

En esta situación es lógico el valor del riesgo bajo de caída porque independientemente de si llueve o no, la cantidad de luz que existe es suficiente para detectar posibles obstáculos en el camino y la humedad es aceptable para un riesgo de caída bajo.

***Test* 3.**

En la Tabla 4.6 se encuentran los valores que tomarán las distintas variables de entrada en el *Test* 3. También se puede observar a qué conjunto difuso pertenecen dichos valores.

| | Humedad | Lluvia | Luz |
|---|---|---|---|
| **Valor** | 90 % | 100 | 1000 |
| **Conjunto difuso** | Muy alta | Sí | Mucha luz |

Cuadro 4.6: Valores y conjuntos difusos de las variables de entrada del *Test* 3.

Para este último *test*, se activaría la siguiente regla:

- **R2.** Si lluvia es sí $\wedge$ (luz es media-alta $\vee$ luz es máxima) $\wedge$ (humedad es alta $\vee$ humedad muy alta) $\rightarrow$ riesgo es **medio**

En este caso, además de estar lloviendo la humedad es alta. Este hecho aumenta el riesgo de caída porque la estabilidad el terreno puede no ser la idónea. Sin embargo, como las condiciones de iluminación exterior son correctas, el riesgo de caída no llega a ser alto. Por lo tanto, la salida del algoritmo difuso para este *test* debería ser un riesgo medio de caída.

**Resultados del algoritmo de análisis del riesgo de caída.**

En el Listado 4.2 se puede observar el código empleado para llevar a cabo los *tests* propuestos anteriormente.

Listado 4.2: **Cálculo del riesgo de caída en una expedición, para unos valores de humedad, luz y lluvia dados.**

```
public void TestAnalisisRiesgo() {
  FuzzyRiskAnalysis.Build();
  // Los parametros del metodo CalculateRisk son (humedad, lluvia, luz)
  String value = FuzzyRiskAnalysis.CalculateRisk(90.0, 200.0, 250.0);
  printResult(1, value);
  value = FuzzyRiskAnalysis.CalculateRisk(40.0, 512.0, 700.0);
  printResult(2, value);
  value = FuzzyRiskAnalysis.CalculateRisk(90.0, 100.0, 1000.0);
  printResult(3, value);
}
```

La salida del algoritmo que calcula el riesgo de caída en la expedición para los *test* propuestos anteriormente se puede ver en la Figura 4.4.

```
Test 1 -> El riesgo de caída es: alto
Test 2 -> El riesgo de caída es: bajo
Test 3 -> El riesgo de caída es: medio
```

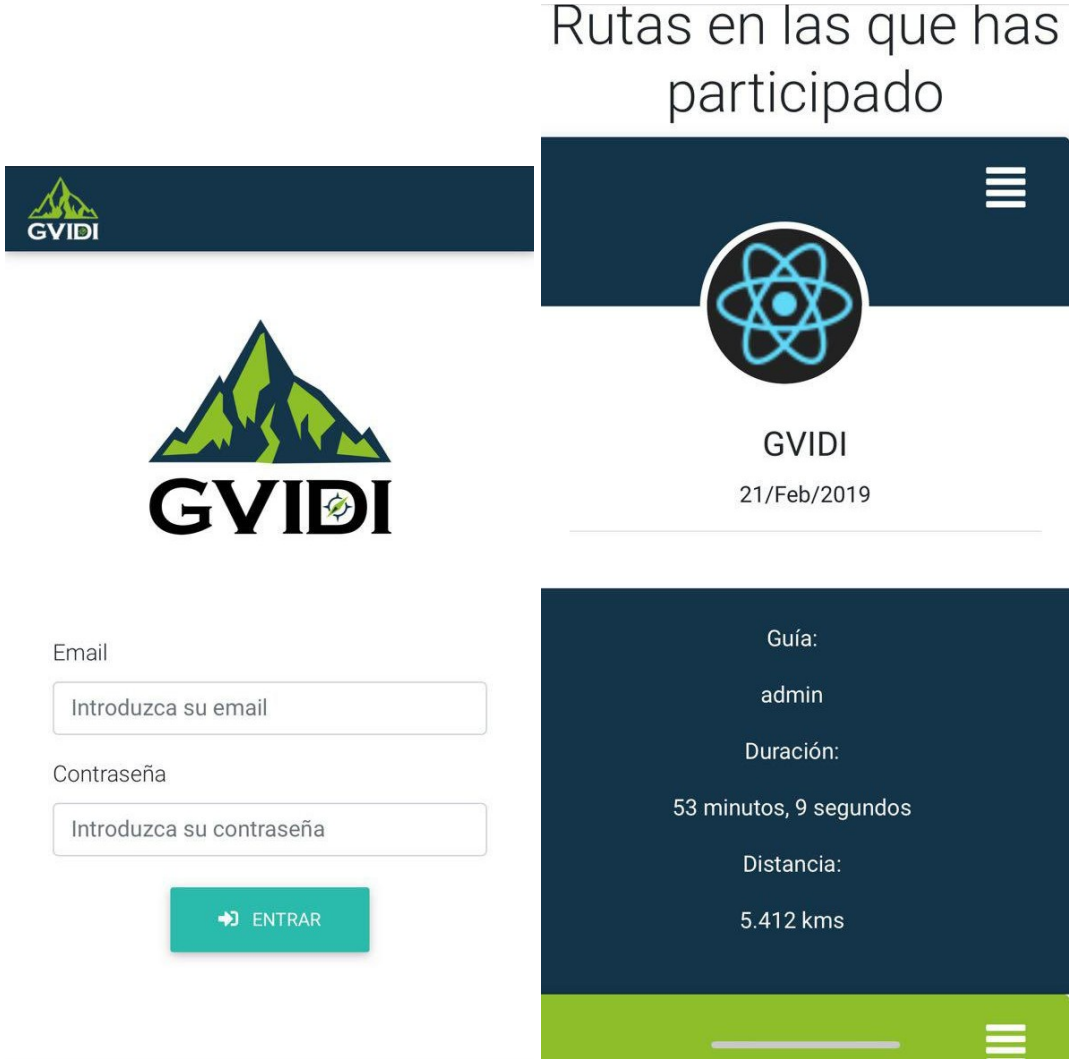
Figura 4.4: Salida del algoritmo que calcula el riesgo de caída en la expedición, para los *tests* aquí propuestos.

## 4.4 Desarrollo de una plataforma web

En este proyecto se ha desarrollado una plataforma *web* que permite a los usuarios (y a familiares y amigos si los usuarios lo permiten) visualizar información sobre la expedición que están realizando en tiempo real. Esta información ha sido procesada y almacenada en una base de datos *cloud* por parte de la aplicación móvil ya comentada.

En las Figuras 3.29, 3.30, 3.31 y 3.32 de la Sección 3.5 se puede observar la interfaz de usuario de la plataforma *web*, vista desde un ordenador portátil con un tamaño de pantalla de 15,6 pulgadas. La *web* tiene un diseño *responsive*, ya que se adapta a distintos dispositivos con tamaños de pantalla diferentes. Esto se puede observar en las Figuras 4.5 y 4.6 en las que se ve la interfaz gráfica adaptada a la pantalla de un `iPhone X`, de 5,8 pulgadas. En este caso, también se han cumplido con todos los objetivos propuestos al inicio del proyecto.

No solo es posible visualizar una expedición en tiempo real sino que también se pueden analizar expediciones realizadas anteriormente, como se puede observar en la Figura




(a) Página de inicio de sesión de la aplicación *web*, vista desde un dispositivo móvil.

(b) Página que contiene el histórico de rutas de un usuario, vista desde un dispositivo móvil.

Figura 4.5: Páginas de *login* e histórico de rutas desde un dispositivo móvil `iPhone X`

3.30.

## 4.5 Competencias específicas de intensificación trabajadas durante este proyecto

A lo largo de este proyecto se han trabajado competencias específicas de la intensificación que he cursado: **Ingeniería de Computadores**. Estas competencias son las siguientes:

- **Capacidad de diseñar y construir sistemas digitales, incluyendo computadores, sistemas basados en microprocesador y sistemas de comunicaciones.** En este proyecto se ha desarrollado un dispositivo *hardware* consistente en un microcontrolador `Arduino` equipado con una serie de sensores. Además, se ha diseñado una `PCB` para eliminar el uso de cables de prototipado y crear un dispositivo con un aspecto más profesional. La comunicación entre el microcontrolador y la aplicación móvil desarrollada

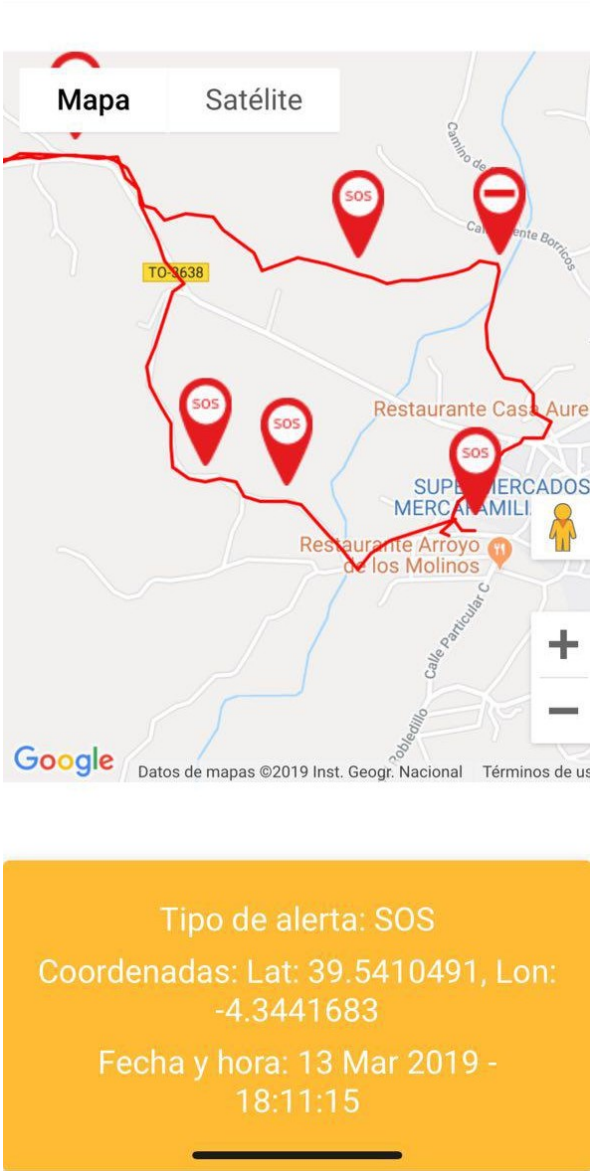



Figura 4.6: Página de información específica acerca de una expedición, vista desde un dispositivo móvil.

se lleva a cabo usando el protocolo de comunicaciones inalámbrico *Bluetooth*.

- **Capacidad de desarrollar procesadores específicos y sistemas empotrados, así como desarrollar y optimizar el *software* de dichos sistemas.** Durante este TFG se ha desarrollado *software* específico para interaccionar con el dispositivo físico desarrollado. Este *software* incluye tanto el código fuente que es ejecutado directamente por el microcontrolador como las librerías que se han usado para captar datos de los sensores equipados en el dispositivo.
- **Capacidad de analizar y evaluar arquitecturas de computadores, incluyendo plataformas paralelas y distribuidas, así como desarrollar y optimizar *software* para las mismas.** En este proyecto se ha planteado una arquitectura distribuida entre los dis-

tintos dispositivos físicos que portan los usuarios de la expedición, la aplicación móvil desarrollada que se encuentra instalada en los dispositivos móviles de los usuarios y la plataforma *web* que permite la visualización tanto en tiempo real como de forma histórica de los datos relativos con la expedición.

- **Capacidad de diseñar e implementar *software* de sistema y de comunicaciones.** Se ha desarrollado *software* específico para tratar con algunos de los sensores que el dispositivo equipa. De igual forma, se ha desarrollado *software* que permite la comunicación usando el protocolo *Bluetooth* entre el dispositivo móvil y el microcontrolador. La comunicación de datos entre la aplicación móvil y la plataforma *web* se realiza a través de una base de datos *cloud* `NoSQL`. La aplicación móvil almacena en la base de datos la información procesada en tiempo real y la plataforma *web* realiza consultas a la misma para obtener los datos almacenados y mostrarlos al usuario.
- **Capacidad de analizar, evaluar y seleccionar las plataformas *hardware* y *software* más adecuadas para el soporte de aplicaciones empotradas y de tiempo real.** Como se puede observar en el Anexo A, se han analizado y evaluado distintas plataformas *hardware* para desarrollar el dispositivo físico que se expone en este proyecto. También se han analizado las alternativas existentes a los sensores usados en este proyecto y se han seleccionado aquellos con una mejor relación calidad-precio. De igual forma, se han tomado decisiones que contemplan distintas alternativas en cuanto a las plataformas *software*. Una de estas decisiones es el uso de *Android* como sistema operativo para la aplicación móvil frente al desarrollo de una aplicación móvil para el sistema operativo `iOS`. También se han evaluado distintas plataformas *software* para el desarrollo de la plataforma *web*. Finalmente se ha usado una arquitectura cliente-servidor, con `node` en el lado del servidor y `react` en el lado del cliente.
- **Capacidad para analizar, evaluar, seleccionar y configurar plataformas hardware para el desarrollo y ejecución de aplicaciones y servicios informáticos.** Para la consecución de este proyecto se han contemplado distintas alternativas, como se puede observar en el Anexo A. Entre estas alternativas destacan la `Rasbperry Pi` y la familia de circuitos integrados `STM32`. Finalmente se decidió usar la plataforma `Arduino` (como se comenta en la sección indicada anteriormente) y se ha desarrollado *software* para esta plataforma.

Capítulo 5

# Conclusiones

En los últimos tiempos la mayoría de los deportes han sufrido una profesionalización por parte de los deportistas aficionados. En gran medida, las nuevas tecnologías han sido culpables de esta profesionalización, ya que han permitido a los deportistas entrenar mejor en el mismo tiempo. El senderismo y otros deportes relacionados con las expediciones en la montaña son un buen ejemplo de esta experticia ya que los usuarios de estos deportes cada vez buscan metas más difíciles de alcanzar y, por tanto, más peligrosas en muchos casos. En el caso del senderismo, los usuarios aumentan el nivel de peligrosidad de las expediciones que realizan conforme van tomando experiencia en el deporte. En muchas ocasiones, un guía comanda los grupos de expedición para velar por la seguridad de los participantes pero agentes como la separación entre los integrantes del grupo pueden hacer que el guía no sea consciente de la situación del grupo en un instante dado. Además, no existen dispositivos o aplicaciones que realicen un análisis de riesgos (riesgo de caída y grado de separación del grupo, entre otros) a nivel grupal sino que las alternativas existentes en el mercado se centran en el análisis individual de cada uno de los participantes. Este proyecto proporciona una visión global del grupo de expedición que no proporcionan otros dispositivos o aplicaciones actualmente en el mercado, por lo que el uso de la herramienta desarrollada a lo largo de este proyecto realmente mejora la seguridad de los usuarios de la expedición.

En este proyecto se ha realizado un dispositivo físico, una aplicación móvil y una plataforma *web* que servirá como ayuda tanto a los guías de senderismo como a los participantes de la expedición. Ambos podrán visualizar tanto en tiempo real como de forma histórica información relevante sobre la expedición.

- El proyecto ayuda a los guías a mantener un mejor control de los participantes en una expedición, maximizando la seguridad de los integrantes de la expedición.
- Los participantes de la expedición pueden visualizar tanto en tiempo real como de forma histórica información sobre la expedición realizada.

El proyecto monitoriza el estado de los integrantes de la ruta (su ritmo, nivel de batería, alertas generadas, etc.) y lo envía al guía de la expedición. Ante estos datos, el guía puede actuar en consecuencia con una mayor rapidez, mejorando la seguridad de los usuarios de la ruta.

El microcontrolador se encarga de tomar datos de los sensores que tiene conectados y enviarlos a través de una conexión *Bluetooth* a la aplicación móvil desarrollada. Los datos que recoge son los siguientes:

- Datos de aceleración lineal para realizar la detección de caídas.
- Datos de geolocalización para situar al usuario en el grupo de expedición.
- Datos de temperatura, humedad, iluminación y lluvia para realizar un análisis del riesgo de caída.

La aplicación móvil que se ha desarrollado se encarga de realizar un procesamiento de los datos que recibe del microcontrolador (también utiliza datos captados en la propia aplicación a través de APIs externas) para obtener información acerca de posibles caídas de un individuo en un momento dado, información del riesgo de caída en tiempo real e información del grado de agrupación del grupo de senderismo. Toda esta información es mostrada al usuario de una forma amigable, con el fin de que no tenga que interrumpir su actividad para manipular las tecnologías que se presentan en este proyecto.

Por último, se ha desarrollado una plataforma *web* que permite la visualización de estadísticas sobre las expediciones realizadas por un usuario de forma histórica, por lo que un usuario puede analizar todas las rutas realizadas una vez que éstas han concluido. Del mismo modo, permite visualizar en tiempo real información acerca de una expedición en curso, por lo que terceras personas (como familiares o amigos del senderista) pueden ver dónde se encuentra el senderista en tiempo real.

Tal como se muestra en el capítulo de Resultados (Capítulo 4), la experimentación diseñada para probar cada una de las partes desarrolladas ha demostrado el correcto funcionamiento de todos los módulos que conforman el sistema, por lo que todos los objetivos propuestos inicialmente (definidos en el Capítulo 2) se han cumplido. Se ha diseñado y desarrollado de forma satisfactoria el dispositivo físico (incluyendo la PCB para simplificar el diseño y eliminar los cables de prototipado). La aplicación móvil implementada tiene toda la funcionalidad que se propuso al inicio de este proyecto y además se ha cumplido también con el objetivo del análisis inteligente de la información tomada por los sensores y el dispositivo móvil.

También se ha cumplido el objetivo del desarrollo e implementación de una plataforma *web*, con un diseño *responsive* y la posibilidad tanto de visualizar en tiempo real una expedición actualmente en curso, como de visualizar información histórica acerca de expediciones realizadas con anterioridad.

Un punto clave a destacar en este proyecto es la novedad del mismo. Actualmente no existen sistemas de características similares en los que se haga una monitorización a nivel de grupo y un análisis inteligente de riesgos. Es un proyecto que tiene una utilidad real en expediciones en la montaña porque en el mercado actual existe la demanda de un producto

que ofrezca la funcionalidad propuesta. A continuación se exponen las ventajas en el uso de la herramienta desarrollada a lo largo de este proyecto:

- Es un proyecto **innovador**. Como se ha comentado anteriormente los dispositivos o aplicaciones que existen en el mercado actualmente para senderistas se centran en un análisis individual mientras que la herramienta presentada en este proyecto basa su uso en el análisis grupal.
- Es un proyecto **multidisciplinar**. A lo largo de este proyecto se han trabajado distintas áreas ya que se ha realizado un dispositivo físico basado en un microcontrolador y se ha equipado con sensores. También se ha desarrollado una aplicación móvil para el sistema operativo *Android*, se ha desarrollado una plataforma *web* siguiendo una arquitectura cliente-servidor y se ha trabajado en la comunicación inalámbrica entre el dispositivo físico y la aplicación móvil.
- En este proyecto se realiza un análisis de datos usando un modelo matemático complejo, como es la lógica difusa. Este análisis es más propio de otras intensificaciones distintas a la mía (Ingeniería de Computadores), por lo que además de cumplir con las competencias propias de mi intensificación (justificadas en la Sección 4.5) se han cumplido competencias de otras intensificaciones.
- Las distintas partes desarrolladas y que conforman este proyecto funcionan correctamente, como se ha observado en el Capítulo 4.

A lo largo de este proyecto he aplicado y sobre todo afianzado conocimientos adquiridos a lo largo del grado. Por ejemplo, he aplicado muchos conocimientos adquiridos en las asignaturas específicas de la intensificación de Ingeniería de Computadores como son los relacionados con los microcontroladores y las redes de comunicaciones. También han sido útiles los conocimientos de asignaturas relacionadas con la Ingeniería del *Software* para plantear un buen diseño de la aplicación móvil y la plataforma *web*. Por último, en este proyecto he adquirido nuevos conocimientos entre los que destacan la gestión de proyectos de larga duración, el aprendizaje de tecnologías *web* en auge como son `node` y `react` y el uso de servicios *cloud* como `Firebase` y `Firestore` para incrementar la eficiencia en el desarrollo de proyectos *software*.

## 5.1 Trabajo futuro

Las líneas de trabajo futuro más destacables para continuar con este proyecto son:

- Mejora del algoritmo de detección de caídas. Se proponen dos posibles formas de mejorar el algoritmo de detección de caídas basado en umbrales que se ha implementado en este proyecto:
  - A través de la inclusión de más datos que puedan inferir una posible caída. Por ejemplo, datos de pulso cardíaco ya que un aumento repentino en el pulso car-

díaco acompañado de unos datos de aceleración anómalos pueden servir para mejorar el detector de caídas. Otro sensor que podría resultar útil es un sensor de vibración, que podría dar picos de valores de vibración cuando existe una caída. Estos sensores son bastante más caros que los usados en este proyecto y para minimizar los costes del dispositivo desarrollado se eligió no hacer uso de los mismos.

  - Realizando el algoritmo de detección de caídas en el microcontrolador *Arduino*, en vez de en el dispositivo móvil. De esta forma, se perderían menos datos de aceleración, porque no es necesarios enviarlos por *Bluetooth*, y los resultados del detector de caídas podrían ser mejores.

- Mostrar a los guías los datos de todos los participantes en la plataforma *web*. En este proyecto se ha optado por mostrar al guía los datos de todos los participantes en la aplicación móvil, ya que el guía debe portar el dispositivo móvil durante la expedición para visualizar las alertas que en ésta se generan y poder tomar decisiones que maximicen la seguridad del grupo lo más rápido posible. Sin embargo, en la plataforma *web* solo se visualizan los datos individuales del usuario (independientemente de si el usuario es un guía o un participante).
- Minimizar el tamaño del dispositivo físico y ajustarlo al tamaño de un reloj. En el prototipo desarrollado en este proyecto, se ha usado `Arduino UNO` como microcontrolador. Este microcontrolador tiene un tamaño bastante grande, por lo que no es factible usarlo a modo de reloj de pulsera. Se podría hacer uso de un microcontrolador mucho más pequeño como el `Arduino Pico` y se podría sustituir la `PCB` de dos capas diseñada por una `PCB` flexible, para que se ajuste a la muñeca del usuario. De igual forma, habría que buscar sensores más pequeños para integrarlos todos en la `PCB`.
- Desarrollar una aplicación móvil para el sistema operativo `iOS`, ya que la aplicación móvil desarrollada en este proyecto sólo funciona en dispositivos móviles con un sistema operativo *Android*.

# ANEXOS

Anexo A

# Antecedentes

En este capítulo se presentarán todos los conceptos necesarios para comprender el proyecto que se va a llevar a cabo, tanto los elegidos para la realización del mismo como otros que se consideran relevantes para una correcta comprensión del problema.

Se explican también los proyectos existentes en el mercado relacionados de una forma u otra con la problemática que se quiere atajar en este TFG, justificando la necesidad del mismo. En el mercado se pueden encontrar distintas aplicaciones y dispositivos que pueden tener una relación con el objetivo principal del presente proyecto, la mejora de la seguridad en grupos de expedición.

## A.1 Microcontroladores

Un microcontrolador [Sha00] es un pequeño computador embebido en un chip, que por regla general incluye un microprocesador o *Central Processing Unit* (CPU), una cierta cantidad de memoria RAM, memoria ROM y elementos de conversión de señal analógica a digital y viceversa. También suele incluir elementos de control de entrada/salida además de ge1neradores de señales y *timers*.

Los microcontroladores se usan para ejecutar pequeñas tareas específicas y normalmente se encuentran embebidos en algún otro disposivo. Estas tareas pueden ser el control de algún dispositivo como una cámara de vídeo, o de alguna funcionalidad concreta dentro de un coche, por ejemplo.

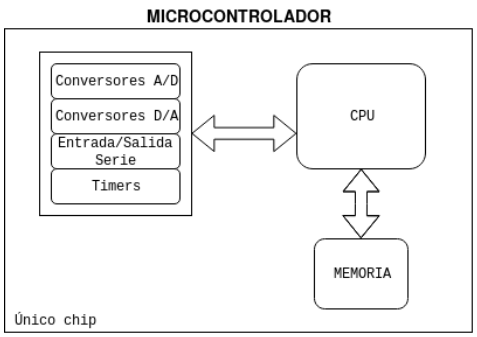



Figura A.1: Esquema de un microcontrolador.

### A.1.1 Arduino

Arduino [Ban15] es una placa de computación física de código abierto, para crear prototipos interactivos programados de una manera fácil e intuitiva. Debido a su naturaleza de código abierto, posee una gran comunidad que comparte sus diseños, por lo que aprender a usar Arduino resulta más sencillo, siendo su curva de aprendizaje relativamente leve, si la comparamos con otros microcontroladores.

La placa de Arduino [McR13] consiste en un microprocesador `ATmega328`. La placa contiene varios pines de entrada/salida, tanto analógica como digital, que servirán para conectar distintos tipos de sensores y actuadores.

Para programar Arduino, se usa Arduino IDE[1], un entorno de desarrollo gratuito que permite programar *software* en un lenguaje que Arduino comprende. Este lenguaje está basado en los lenguajes de programación C y C++ y se puede extender usando librerías programadas en C++. Usando este *Integrated Development Environment (*IDE*)*, se puede cargar un *software* determinado en la placa Arduino de una manera muy sencilla.

Que la curva de aprendizaje de Arduino sea más leve que la de otros microcontroladores [Mar11], no quiere decir que Arduino sea una peor alternativa que todos ellos, ya que la capa de *hardware* trabaja con el mismo nivel de sofisticación que otros dispositivos embebidos. Se suele elegir Arduino cuando se quiere un desarrollo rápido y obtener facilidades en la implementación.

El esquema básico de un código Arduino se puede observar en el listado A.1. Arduino siempre posee las funciones `setup` y `loop`. La función `setup` se usa para inicializar el estado del programa. Este código se ejecuta sólo una vez al inicio del programa. La función `loop`, contiene el código principal del programa y se ejecuta de forma continua. Un programa en Arduino no puede acabar, se ejecutará de forma ininterrumpida mientras la placa esté encendida. Tampoco es posible ejecutar más de un programa en paralelo en Arduino.

Listado A.1: **Esquema básico de un código Arduino.**

```
1 void setup() {
2 }
3 void loop() {
4 }
```

**Ventajas de usar el microcontrolador Arduino.** Las principales ventajas en el uso de Arduino se pueden contemplar a continuación:

1. Facilidad de configuración. Arduino es un dispositivo *plug-and-play*, por lo que el tiempo de configuración es mínimo.

[1] `https://www.arduino.cc/en/Main/Software`

2. Integración con una gran cantidad de sensores. Muchos de los sensores que se van a usar en este proyecto, poseen librerías para interactuar con Arduino, por lo que se produce un ahorro de tiempo en la creación y uso de estas bibliotecas.
3. El *hardware* es de bajo coste. Como este proyecto es una red de dispositivos, tiene sentido crear varios prototipos e interconectarlos, en vez de uno solo. El prototipo tenía que ser, por tanto, lo más barato posible.
4. La programación del *software* se realiza en un lenguaje basado en C y C++, lenguajes conocidos por el estudiante.

### A.1.2 Raspberry Pi

Raspberry Pi es un microcontrolador [Yam15], lanzado al mercado por la Fundación Raspberry Pi[2]. Existen varias versiones de este microcontrolador pero todas tienen en común el bajo precio del mismo, en torno a los 30 euros. Raspberry Pi puede ejecutar aplicaciones sobre un sistema operativo, al contrario que Arduino, que ejecuta directamente el *software* sobre el *hardware*.

Raspberry Pi tiene un microprocesador `ARM` y un juego de instrucciones RISC de 32 bits hasta la versión `Raspberry Pi3`. Desde esta versión en adelante usa un juego de instrucciónes RISC de 64 bits. También posee una *Graphics Processing Unit (*GPU*)* y memoria SDRAM.

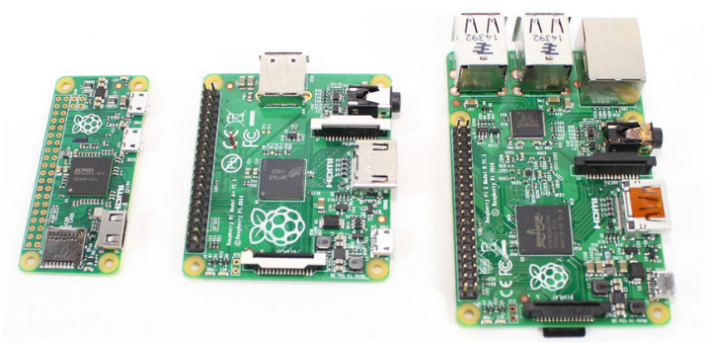


Figura A.2: Varios modelos de Raspberry Pi. `Raspberry Pi Zero`, `Raspberry Pi A+` y `Raspberry Pi2` (de izquierda a derecha). [3]

Raspberry Pi fue lanzado con el propósito de enseñar a programar [Wat16], sin embargo es una gran plataforma para entusiastas del *hardware*, que buscan un dispositivo para sus proyectos de electrónica. Raspberry Pi posee una serie de pines, denominados «*General Purpose Input/Output (*GPIO*) pins*» que proporcionan una interfaz de comunicación con distintos sensores y actuadores. Dependiendo de la versión de Raspberry Pi el número de «GPIO *pins*» puede variar.

Se pueden instalar muchos sistemas operativos en Raspberry Pi, pero sin duda el más

[2] `https://www.raspberrypi.org`
[3] Imagen original extraída de [Wat16]

usado es Raspbian, un sistema operativo basado en Debian. El sistema operativo se «*flashea*» en una tarjeta de memoria SD, que se introducirá en la Raspberry. A partir de esta tarjeta SD, se puede iniciar el sistema operativo.

El principal lenguaje para programar una Raspberry Pi es Python[4] [Mag], aunque también se pueden usar otros lenguajes como C++, C o Java. Python es uno de los lenguajes más populares hoy en día y además tiene una curva de aprendizaje mucho menos pronunciada que otros lenguajes como C o C++. Además, existen un gran número de librerías para Python, lo que supone un ahorro de tiempo a la hora de la programación. El Listado A.2 muestra cómo encender y apagar un led, usando Python como lenguaje de programación.

Listado A.2: **Encender y apagar un led en Raspberry.**

```
import RPi.GPIO as GPIO
import time

GPIO.setmode(GPIO.BCM)
GPIO.setup(18, GPIO.OUT)  # Estblecer pin 18 como salida

while True:
  GPIO.output(18, 1)      # Encender pin 18
  time.delay(1)
  GPIO.output(18,0)       # Apagar pin 18
```

### A.1.3 Otros microcontroladores

1. **mbed**.

   La plataforma mbed[5] [Mon12] es similar a Arduino en lo que a objetivos se refiere, pero está más orientado a la prototipación sobre una placa. Usa una arquitectura basada en un microprocesador ARM Cortex-M3, mucho más potente que otros microprocesadores, como el ATmega328 de Arduino.

   mbed posee un gran número de librerías, ya que existe una gran comunidad de programadores para esta plataforma. Para desarrollar para el microcontrolador mbed, no es necesario descargar *software* adicional, puesto que todas las herramientas se pueden encontrar online.

   Para subir un programa compilado a la placa tan sólo hay que conectarla por USB al computador. El código se copiará en la placa y, después de presionar el botón de *reset*, la placa estará lista para ejecutar el programa que se ha desarrollado.

   Al igual que con Arduino, no se puede acceder al código final generado por el compilador, para optimizarlo en caso de ser necesario. No hay herramientas de depuración

[4] https://www.python.org
[5] https://www.mbed.com/en/
[6] Imagen extraída de https://media.rs-online.com/t_large/R7039238-01.jpg

Figura A.3: Microcontrolador mbed. [6]

para mbed.

2. **STM32**

   STM32 es un microcontrolador desarrollado por la compañía *ST Microelectronics* [Nor18]. Existen varios *kits* de STM32 pero todos tienen en común el bajo coste de los mismos, en torno a 15 dólares. Las placas STM32 pueden ser programadas usando código ensamblador o con el lenguaje de programación C.

   La placa [Bro13] posee un microprocesador ARM Cortex-M3 (o ARM Cortex-M4 en los últimos modelos) de 32 bits. Este microprocesador nos proporciona un gran rendimiento y flexibilidad, aunque existen contras como la dificultad en el desarrollo de *software* para la placa STM32 si eres principiante. Además, la placa STM32 soporta un *Real Time Operative System* (RTOS), lo que supone que se pueden programar sistemas mucho más complejos que en otros microcontroladores, como Arduino.

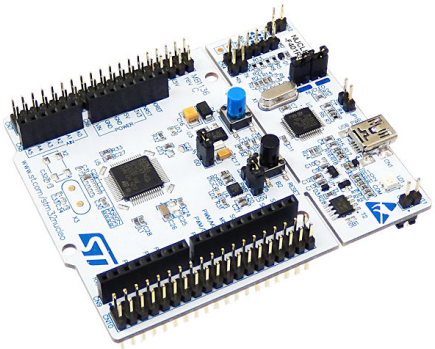


Figura A.4: Placa STM32 Nucleo. [7]

Para programar la placa STM32, se puede hacer uso del *software* STM32CubeMX[8]. Se trata de una herramienta de interfaz gráfica, que ayuda a crear configuraciones para la placa usando el lenguaje de programación C.

[7] Imagen extraída de https://grobotronics.com/stm32-nucleo-development-board-for-stm32-f1-series-with-stm32.html?sl=en

[8] https://www.st.com/en/development-tools/stm32cubemx.html

| | Procesador | Velocidad de Reloj | Memoria *Flash* | SRAM | EEPROM | Tarjeta SD | Lenguaje |
|---|---|---|---|---|---|---|---|
| Arduino UNO | ATmega328 | 16 MHz | 32 KB | 2 KB | 1 KB | No | C++ |
| Raspberry Pi 3B | ARM Cortex-A53 | 1,2 GHz | - | 1 GB SDRAM | - | Sí | Python, C, C++, etc. |
| mbed | ARM Cortex-M3 | 96 MHz | 512 KB | 32 KB | - | No | C, C++ |
| STM32 Nucleo | ARM Cortex-M4 | 84 MHz | 512 KB | 96 KB | - | No | C |

Cuadro A.1: Comparativa de las principales características de los distintos microcontroladores

## A.2 Sensores

Los sensores son [Kar14] componentes eléctricos o electrónicos que funcionan como dispositivos de entrada. No todas las entradas son explícitamente sensores, pero las entradas usan sensores. Por ejemplo, en el caso de un ratón de ordenador. El ratón no es un sensor en sí mismo, pero contiene una serie de ellos, por ejemplo el sensor que detecta el movimiento del mismo.

Podemos pensar en sensores como componentes que miden estímulos externos al sistema (miden el entorno del sistema). El sistema, puede reaccionar a estos estímulos mediante los **actuadores**. Por ejemplo, un sistema con un sensor de luminosidad, puede detectar la cantidad de luz que existe en el ambiente. Procesando esta cantidad de luz, se puede obtener como salida la activación o no de una bombilla.

Para el desarrollo de este proyecto se han usado distintos sensores, que se comentarán a continuación:

1. **Sensor *Light-Dependent Resistor* (LDR)**
   El sensor GL5528 es un LDR, que consiste en una resistencia variable con respecto a la luz. La resistencia del fotoresistor decrementa conforme incrementa la intensidad de la luz sobre el mismo.

   Para elegir el valor de la resistencia que acompaña al LDR, se puede usar la fórmula de *Axel Benz*. Si se usase el sensor en un área muy luminoso con una resistencia de *pull-down* de $10K\Omega$, se saturará rápidamente. El votaje a través de la resistencia pronto será de $5V$ y no se podrá diferenciar con facilidad entre algo brillante y algo muy brillante. En este caso, habría que reemplazar la resistencia de $10K\Omega$ por una de $1K\Omega$, pero ahora no se detectarán los niveles de oscuridad tan bien como antes. Para encontar ese valor de resistencia, usamos la fórmula de *Axel Benz*:

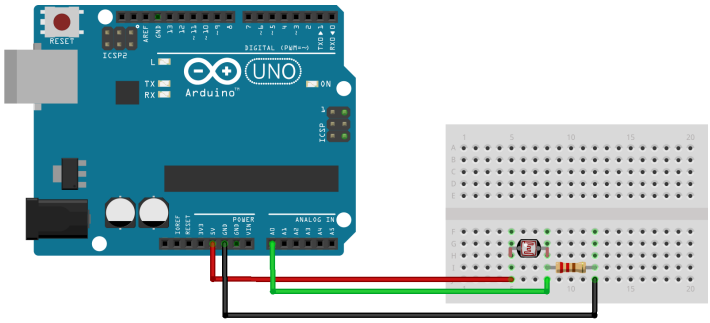


Figura A.5: Esquema de conexión del LDR y Arduino.

$$R_{ref} = \sqrt{R_{Min} \cdot R_{Max}} \tag{A.1}$$

El valor $R_{Max}$, es el valor de resistencia que ofrece el LDR, cuando se encuentra no recibe nada de luminosidad. El valor de $R_{Min}$ es el valor de resistencia que ofrece el LDR cuando recibe la luminosidad máxima.

Listado A.3: **Utilización básica del sensor GL5528.**

```
#include "GL5528.h"

GL5528 gl5528(A0);

void setup(){
 pinMode(A0, INPUT);    // Set A0 pin as an input pin
 Serial.begin(9600);
}

void loop(){
  LightThresholds light_thres = gl5528.getLightMeasure();
  if (light_thres == LightThresholds::LT_VERY_SUNNY)
  {
    Serial.println("Very Sunny");
  }
  else if (light_thres == LightThresholds::LT_NIGHT)
  {
    Serial.println("It's night");
  }
}
```

2. **Sensor de lluvia**

   El sensor de lluvia `YL-83` es una herramienta para la detección de lluvia. Se puede usar como un indicador que señala cuándo una gota de agua cae en la placa detectora y también para medir la intensidad de lluvia. Este sensor permite ajustar la sensibilidad a través de un potenciómetro.

   Con la salida analógica se obtiene un valor que puede ser usado para detectar la intensidad de lluvia. La salida digital indica la presencia o ausencia de lluvia.

   La resistencia que posee la placa detectora varía de acuerdo a la cantidad de agua que existe en su superficie. Cuanta más agua exista en la superficie, menor será el voltaje de salida. Por tanto, un voltaje bajo de salida indica que la resistencia interna de la placa detectora ha aumentado y, por tanto, existe agua en su superficie. Un voltaje alto de salida indicará que no existe agua. Los valores intermedios se pueden usar para identificar la intensidad de la lluvia.

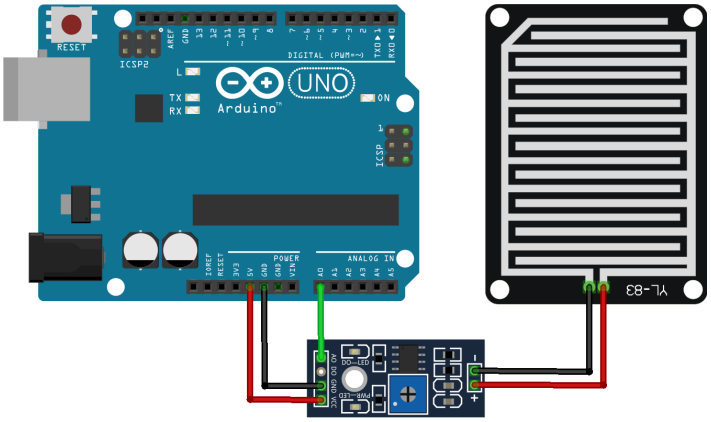



Figura A.6: Esquema de conexión del sensor de lluvia `YL-83`.

Listado A.4: **Utilización básica del sensor de lluvia YL38.**

```
#include <RainSensorYL38.h>

RainSensorYL38 RainSensor(A0);

void setup() {
  Serial.begin(9600);
}

void loop() {
  RainAmount rainamount = RainSensor.GetRainAmount();
  switch (rainamount)
  {
    case RainAmount::RA_NO_RAIN:
      Serial.println("No Rain");
```

```
        break;
      case RainAmount::RA_LOW_RAIN:
        Serial.println("Low rain");
        break;
      case RainAmount::RA_MODERATE_RAIN:
        Serial.println("Moderate rain");
        break;
      case RainAmount::RA_HEAVY_RAIN:
        Serial.println("Heavy rain");
        break;
    }
}
```

3. **Sensor de temperatura y humedad**

   En este proyecto se ha usado el sensor de temperatura y humedad `DHT22` [Liu]. Tiene un rango de medición de temperatura que va desde $-40$ hasta $125$ grados Celsius, con una precisión de $\pm 0{,}5$ grados Celsius. En lo que a la humedad se refiere, posee un rango de medición de humedad que va desde el $0\,\%$ hasta el $100\,\%$, con una precisión de entre el $2$ y el $5\,\%$. El sensor `DHT22` realiza una medición cada dos segundos y opera tanto a $3V$ como a $5V$.

   Para medir la humedad, usa un componente sensible a la humedad con dos electrodos y entre ellos un substrato en el que se mantiene la humedad. Tan pronto como la humedad cambia, la conductividad del substrato cambia y también varía la resistencia entre los electrodos. Estos cambios son medidos e interpretados por el sensor.

   En lo que a temperatura se refiere, el sensor usa un termistor, una resistencia variable que cambia su valor con respecto a la temperatura. Cuanto menor es la temperatura, más alta es la resistencia. La resistencia va disminuyendo conforme la temperatura aumenta.

   https://desarrollador-android.com/wp-content/uploads/2015/05/basic-lifecycle.png

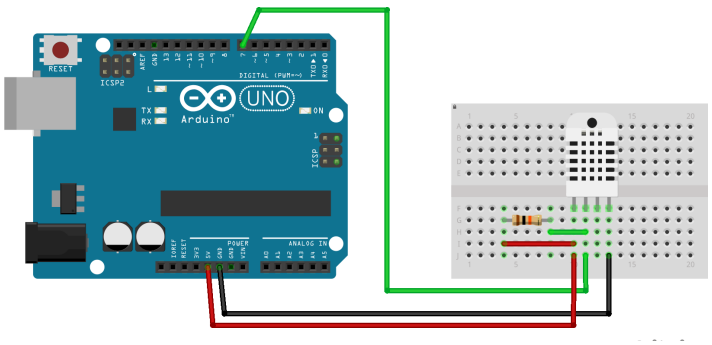



Figura A.7: Esquema de conexión del sensor de temperatura y humedad `DHT22`.

Listado A.5: **Utilización básica del sensor de temperatura y humedad DHT22.**

```
#include <DHT22.h>

#define DHT22_PIN 7

DHT22 myDHT22(DHT22_PIN);

void setup(void)
{
  Serial.begin(9600);
}

void loop()
{
  DHT22_ERROR_t errorCode;

  delay(2000);  // The sensor need 2 seconds between readings

  errorCode = myDHT22.readData();
  if (errorCode == DHT_ERROR_NONE)
  {
      Serial.print("Temperature: "); Serial.print(myDHT22.getTemperatureC());
      Serial.print("C\t|\tHumidity: "); Serial.println(myDHT22.getHumidity());
  }
  else
  {
    char buffer[50];
    myDHT22.getErrorString(errorCode, buffer);
    Serial.println(buffer);
  }
}
```

### 4. Sensor de pulso cardíaco

Para el desarrollo de este proyecto, se ha usado el sensor de pulso cardíaco `PulseSensor`[9]. Este sensor combina un sensor de pulso cardíaco óptico, con un circuito de amplificación de señal y cancelación de ruido, lo que proporciona unas mediciones de pulso cardíaco bastante confiables.

El sensor `PulseSensor`, es básicamente un fotopletismógrafo. Esto es, un dispositivo que utiliza un haz de luz para monitorizar, de una forma no invasiva el pulso cardíaco. Este sensor identifica instantes sucesivos en los que se produce un latido, para medir el tiempo entre ellos. Este tiempo se denomina *Interbeat Interval* (IBI) y se puede usar para calcular el ritmo cardíaco.

[9] `https://pulsesensor.com`

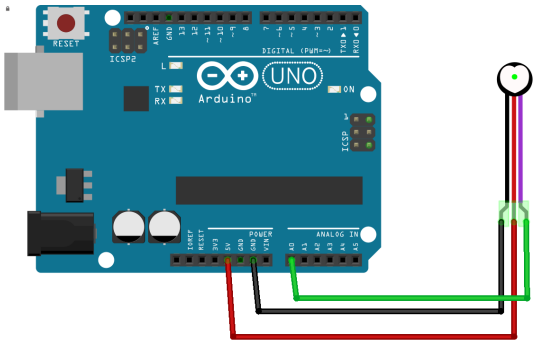


Figura A.8: Esquema de conexión del sensor de pulso cardíaco `PulseSensor`.

Listado A.6: **Utilización básica del sensor de pulso cardíaco.**

```
#define USE_ARDUINO_INTERRUPTS true
#include <PulseSensorPlayground.h>

const int OUTPUT_TYPE = SERIAL_PLOTTER;
const int PULSE_INPUT = A0;
const int THRESHOLD = 550;   // Adjust this number to avoid noise when idle
PulseSensorPlayground pulseSensor;

void setup() {
  Serial.begin(115200);
  pulseSensor.analogInput(PULSE_INPUT);
  pulseSensor.setSerial(Serial);
  pulseSensor.setOutputType(OUTPUT_TYPE);
  pulseSensor.setThreshold(THRESHOLD);
}

void loop() {
  delay(100);
  // write the latest sample to Serial.
  pulseSensor.outputSample();
  if (pulseSensor.sawStartOfBeat())
  {
    pulseSensor.outputBeat();
  }
}
```

## 5. Sensor acelerómetro y giroscopio

Un giroscopio es un sensor que se encarga de medir la velocidad angular, o lo que es lo mismo, la velocidad de rotación de un objeto alrededor de uno de sus ejes. El giroscopio puede usarse para determinar la orientación de un cierto objeto, por ejemplo.

Un acelerómetro es un sensor diseñado para medir la aceleración no gravitacional

en uno, dos o tres ejes. El acelerómetro es capaz de detectar cambios en las fuerzas que actúan sobre él, debido a un movimiento, por ejemplo. Con la variación de la velocidad, se puede calcular la aceleración del objeto. También se pueden calcular de forma indirecta otras medidas, como la velocidad y el desplazamiento de un objeto.

El sensor `MPU6050` se compone de un acelerómetro de 3 ejes y un giroscopio de 3 ejes, que permite medir aceleración y velocidad angular en el espacio tridimensional. Este sensor proporciona seis valores de salida con cada medición: aceleración en los ejes $X$, $Y$, $Z$ y velocidad angular en los ejes $X$, $Y$, $Z$.

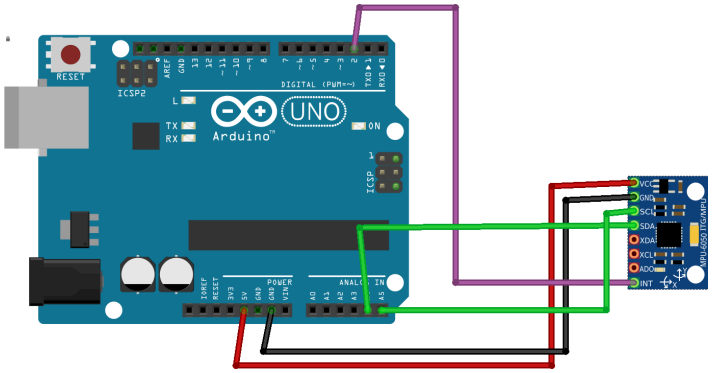



Figura A.9: Esquema de conexión del acelerómetro y giroscopio `MPU6050`.

Listado A.7: **Utilización básica del acelerómetro y giroscopio MPU6050.**

```
#include "I2Cdev.h"
#include "MPU6050.h"
#include "Wire.h"

MPU6050 sensor;

int ax, ay, az;
int gx, gy, gz;

bool initiated = false;

void setup() {
  Wire.begin();           // Initiate I2C
  sensor.initialize();    // Initiate the sensor
  if (sensor.testConnection()) initiated = true;
}

void loop() {
  if (initiated)
  {
    // Read acceleration and angular velocity
```

```
22      sensor.getAcceleration(&ax, &ay, &az);
23      sensor.getRotation(&gx, &gy, &gz);
24    }
25    delay(100);
26  }
```

## A.3 Sistemas Operativos para Dispositivos Móviles

### A.3.1 Android

Android es un sistema operativo de código abierto, desarrollado por Google [Ali16]. Está basado en el núcleo de Linux y está diseñado principalmente para dispositivos móviles con una pantalla táctil, como pueden ser *smartphones* o *tablets*. Android es el sistema operativo para móviles más usado en el mundo, con más de dos mil millones de dispositivos en la actualidad. Desarrollar una aplicación para un dispositivo Android significa que cualquier dispositivo que tenga este sistema operativo será capaz de ejecutarla, sea del fabricante que sea, abstrayendo aspectos relativos a cada modelo de dispositivo. Android no solo está presente en dispositivos móviles, en los últimos años se puede observar un crecimiento de Android en televisores y otras máquinas, como pueden ser los coches.

Android es un sistema operativo que goza de varias versiones. Cada versión se compone por un número de versión, un nombre de sistema operativo y un nivel de API. El número de versión se asemeja al sistema de versionado convencional (2.1, 4.0, 5.4.3). Si se produce un cambio en el primer dígito de la versión, el sistema operativo sufre un cambio notorio, con muchas APIs nuevas. Si cambia el segundo dígito, se produce una evolución en el sistema operativo. Si, por el contrario, es el tercer dígito de la versión el que cambia, se habrán producido cambios menores. Los niveles de la API se numeran de forma monótona y el código de nombres asignados al sistema operativo siempre se refiere a comidas dulces.

Android es un sistema operativo compatible con versiones anterirores. Esto quiere decir que una aplicación escrita para una *release* antigua se ejecutará sin necesidad de ser recompilada en una *release* más moderna, pero no al revés. Por ejemplo, una aplicación compilada en un dispositivo Android 4 podrá ejecutarse en un dispositivo con Android 7 pero no viceversa ya que en Android 7 se hacen usos de llamadas a APIs que no existían en Android 4.

Para el diseño visual de una aplicación Android, se puede usar la guía de «*Material Design*» ya que en las últimas versiones, Android sigue las pautas descritas en esta guía haciendo más fácil crear una aplicación cuyos elementos visuales estén correctamente estructurados, animados y con colores acordes.

Una aplicación Android consiste en uno o más componentes, escritos en lenguaje Java

(también se puede desarrollar una aplicación Android usando lenguaje Kotlin[10]). Estos componentes pueden ser:

1. **Actividades**. Estos componentes abarcan la parte visual (las vistas), del código que se encarga de mostrar los datos en estas vistas y de la respuesta a ciertos eventos de un usuario, como puede ser pulsar o arrastar un elemento de la vista.
2. **Servicios**. Son componentes que no tienen parte visual y pueden ejecutarse por un período de tiempo más prolongado que una actividad, normalmente en segundo plano. Los servicios pueden continuar ejecutándose aunque el usuario pase a usar otra aplicación.
3. ***Broadcast Receiver***. Es un componente que se encarga de responder a ciertos eventos generados en el sistema, como fallos de conectividad en la red o un aviso de batería baja.
4. **Proveedores de Contenido**. Se usan para compartir datos con otras aplicaciones. Se pueden usar estos proveedores de contenido para tomar datos que proporcionan ciertas aplicaciones como el calendario o los contactos.
5. **Adaptador *sync***. Se usan para sincronizar datos desde una aplicación con servicios *cloud*.

Una aplicación en Android no tiene un método `main` que ejecutará el código, ni un método `loop` que se ejecutará de forma contínua mientras la aplicación esté abierta. Las actividades en Android siguen un ciclo de vida (véase Figura A.10), que es necesario comprender. Una aplicación puede encontrarse en uno de los tres estados siguientes:

1. **Activa**. Una aplicación se encuentra activa si es visible al usuario y se está ejecutando.
2. **Pausada**. Una aplicación se encuentra en este estado si una parte de la misma ha perdido el foco y el usuario no puede interactuar con ella. Esta situación se da cuando aparece un diálogo en la aplicación, por ejemplo.
3. **Parada.** Una aplicación se encuentra parada si su vista se encuentra totalmente oculta al usuario.

Antes de comenzar a desarrollar una aplicación Android, hay que configurar la plataforma de desarrollo [Smy17]. Para realizar esto es necesario seguir una serie de pasos, como la instalación de un IDE como es `Android Studio`, que incluye el Android SDK. También es necesario configurar *Open Java Development Kit* (OPENJDK), la versión libre de la plataforma de desarrollo Java.

[10] `https://kotlinlang.org`

[11] Imagen extraída de `https://desarrollador-android.com/wp-content/uploads/2015/05/basic-lifecycle.png`

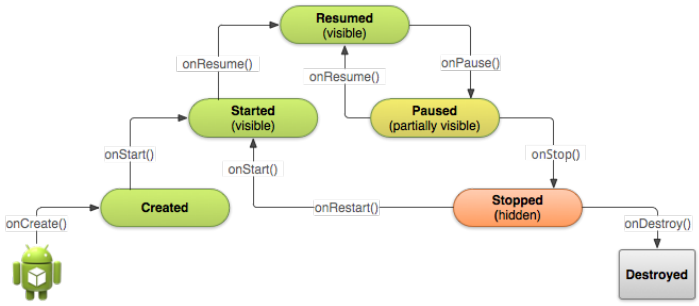



Figura A.10: Ciclo de vida de una actividad en Android. [11]

`Android Studio` es el fruto de una colaboración entre JetBrains[12] y Google [Ger15]. Se ha convertido en el IDE oficial para el desarrollo de aplicaciones Android. Este IDE está construido sobre `IntelliJ`, por lo que la funcionalidad que aporta es un superconjunto de la que aporta `IntelliJ`.

El *Android Virtual Device Manager* (AVDM) permite crear distintos *Android Virtual Device* (AVD), que emularán un dispositivo Android ejecutándose en el computador. El computador reserva un bloque de memoria para reproducir el entorno del dispositivo que se va a emular, es decir, se ejecuta una versión del núcleo de Linux y todo el sistema operativo Android que contendría el dispositivo físico. La emulación es mejor que la simulación, ya que proporciona un entorno más fiel al real. Sin embargo, para la creación y arranque de un AVD se necesitan muchas más prestaciones. Para probar la aplicación Android no es necesario usar un AVD. Si se dispone de un dispositivo Android, se puede conectar al computador para ejecutar la aplicación directamente en el dispositivo físico.

## A.4 Bluetooth

Bluetooth es una forma de comunicación inalámbrica entre dispositivos a corta distancia [Hua07]. La comunicación inalámbrica existe desde finales del siglo XIX en forma de radio, infrarojos, televisión y más recientemente el estándar 802.11. La comunicación vía Bluetooth es de corta distancia, normalmente menos de 10 metros. Tanto el *hardware* como el *software* están afectados por esta restricción.

Los dispositivos Bluetooth se dividen en tres clases según su potencia (en la Tabla A.2 se puede ver dicha división). La única diferencia entre esas clases es la distancia máxima de separación que puede soportar una conexión Bluetooth. La gran mayoría de dispositivos Bluetooth de hoy en día son de clase 2 (dispositivos móviles, auriculares, ratones y teclados, etc.). Existen dispositivos USB de clase 1, que tienen mayor potencia. Si se establece una conexión entre un dispositivo de clase 1 y uno de clase 2, la distancia máxima de separación entre ellos viene dada por el dispositivo de clase 2. Los dispositivos de clase 3 son más raros,

[12] `https://www.jetbrains.com`

ya que tienen una fuerte limitación en el rango y los hace poco útiles.

| Clase | Distancia |
|---|---|
| 1 | 100 metros |
| 2 | 10 metros |
| 3 | <1 metro |

Cuadro A.2: Clasificación de los dispositivos Bluetooth según su potencia.

No es sencillo dar un dato exacto sobre la cantidad de ancho de banda en un canal de comunicaciones Bluetooth, pero teóricamente dos dispositivos Bluetooth poseen un ancho de banda asimétrico máximo de $723{,}2$ kbps y un ancho de banda máximo simétrico de $433{,}9$ kbps. Entendemos por ancho de banda asimétrico aquel en el que solo está transmitiendo un dispositivo Bluetooth y por ancho de banda simétrico aquel en el que los dos dispositivos están transmitiendo a la vez el uno al otro. Estos anchos de banda son teóricos y los reales siempre serán menores, debido principalmente al ruido en una conexión inalámbrica.

Como en todas las comunicaciones inalámbricas, la fuerza de una señal se deteriora con el cuadrado de la distancia desde el origen. Además, las señales más débiles son más fáciles de corromper por el sonido. Por tanto, el ancho de banda en una comunicación Bluetooth será menor cuanto mayor sea la distancia de separación entre los dos dispositivos.

En la especificación Bluetooth 2.0, del año 2004, los anchos de banda aumentaron hasta $2178{,}1$ kbps en el caso del ancho de banda asimétrico máximo y hasta $1306{,}9$ kbps en el caso del ancho de banda simétrico máximo. Con la llegada de Bluetooth 3.0 en el año 2009, la velocidad máxima teórica de transferencia de datos de hasta $24$ Mbps aunque no a través del enlace Bluetooth, propiamente dicho. Si se usa la especificación Bluetooth 4.0 del año 2010, se obtienen velocidades teóricas de transferencia de datos de entre $25$ Mbps y $32$ Mbps. Con la llegada de Bluetooth 5.0 a finales del año 2016 se dobla la velocidad de transmisión de datos, además de cuadruplicar el alcance entre dos dispositivos Bluetooth.

| Año de Lanzamiento | Versión Bluetooth | Características introducidas |
|---|---|---|
| 2004 | 2.0 | Velocidad de datos mejorada |
| 2007 | 2.1 | Emparejamiento Simple Seguro (SSP) |
| 2009 | 3.0 | Alta velocidad, con 802.11 Radio Wi-Fi |
| 2010 | 4.0 | Protocolo de bajo consumo de energía |
| 2013 | 4.1 | Conexión a dispositivos IoT indirecta |
| 2014 | 4.2 | Uso de IPv6 para conexión directa a Internet |
| 2016 | 5.0 | 4 veces más rango, 2 veces más velocidad |

Cuadro A.3: Resumen de las características más importantes añadidas a las distintas versiones de Bluetooth.

No existe un estándar sobre Internet, ya que se compone por una serie de protocolos (con sus propios estándares), como pueden ser `Ethernet`, `TCP/IP` o `HTTP`. Bluetooth es parecido

a Internet, ya que está compuesto también por distintos protocolos que definen por un lado las comunicaciones físicas (como las frecuencias de radio y cómo modular y demodular señales) y por otro lado otras cuestiones como por ejemplo la transmisión de audio entre dispositivos. Sin embargo, a diferencia de Internet, Bluetooth si que posee un estándar. La mayor diferencia entre Bluetooth e Internet reside en la distancia física entre dispositivos ya que en Internet la distancia no es una limitación, a priori.

En el proceso de establecimiento de una conexión Bluetooth hay un dispositivo (al que llamaremos cliente) que es quien inicia la conexión. El dispositivo servidor estará a la escucha de conexiones. Esta nomenclatura no tiene nada que ver con el modelo cliente-servidor ya que una vez establecida la conexión cualquiera de los dos dispositivos puede actuar como cliente o servidor.

Los dispositivos que quieren establecer una conexión de salida necesitan escoger un destino y un protocolo de transporte. Los dispositivos que quieren establecer conexiones de entrada necesitan escoger un protocolo de transporte y después ponerse a la escucha de conexiones de salida. Cada dispositivo Bluetooth posee un identificador único de fábrica, de 48 bits, que actúa como la dirección física. Estos identificadores no cambian nunca y son únicos por dispositivo. Como estos identificadores no son muy amigables desde el punto de vista de un usuario, se puede asignar un nombre al dispositivo Bluetooth, que el cliente se encargará de traducir a la dirección numérica. Este nombre se denomina «nombre de exposición».

Durante el descubrimiento de dispositivos, se envía un mensaje *broadcast* de «descubrimiento», al que otros dispositivos responden con su dirección y un número que se corresponde con el tipo de dispositivo (dispositivo móvil, PC, auriculares, etc.). El nombre de exposición se puede conocer preguntando directamente a un dispositivo concreto.

Dependiendo de las necesidades concretas de la aplicación, se puede elegir un protocolo de transporte u otro. Bluetooth posee varios protocolos de transporte a escoger, como son RFCOMM, L2CAP, ACL y SCO. De entre ellos, el protocolo más usado es RFCOMM.

Para que puedan existir varias aplicaciones usando un mismo dispositivo Bluetooth, se usan los puertos. Varias aplicaciones, con un uso específico, poseen su número de puerto fijado. Así surgen los «*well-known ports*». Para conocer en qué puerto escucha una determinada aplicación a la que se quiere conectar se usa el *Service Discovery Protocol* (SDP). Todo dispositivo Bluetooth mantiene un servidor SDP que está a la escucha en un puerto reservado (el puerto 1 en este caso). Cuando una aplicación se ejecuta en un puerto, registra una descripción y un número de puerto en el SDP. Si una aplicación remota quiere acceder a ella, lo hará consultando al SDP.

Los clientes necesitan saber cuál de las aplicaciones registradas en un SDP es la que está buscando. La forma más fácil de saberlo es asignar a cada servicio un identificador único o *Universally Unique Identifier* (UUID) de 128 bits. SDP denomina a estos UUID identifi-

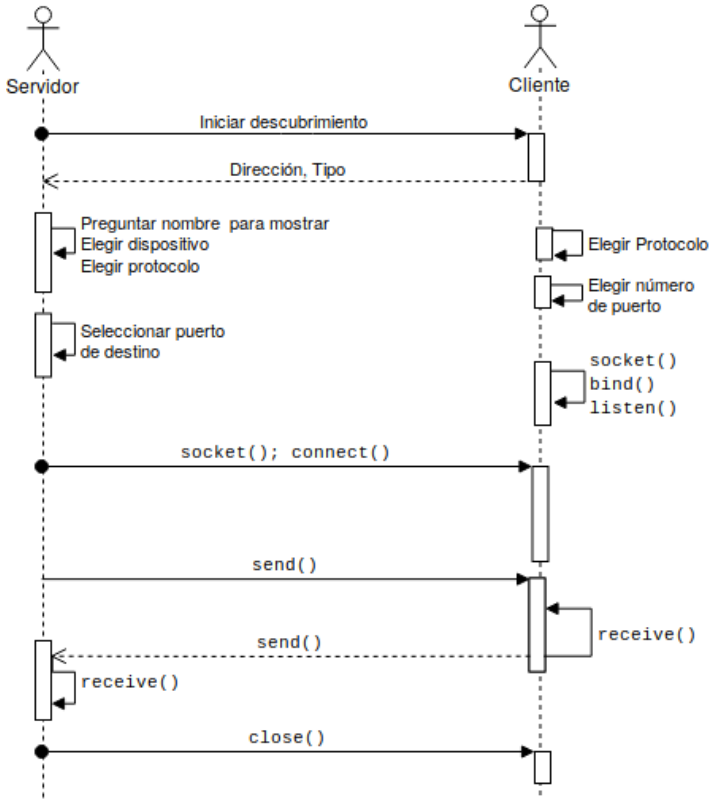



Figura A.11: Establecimiento de una conexión Bluetooth.

cadores de servicio. Cuando una aplicación se ejecuta quedando a la espera de conexiones Bluetooth, registra en el SDP su identificador de servicio. Una aplicación cliente que se quiera conectar, hará una petición al servidor SDP para saber si hay registrada alguna aplicación con un identificador de servicio igual al que está buscando.

## A.5 Desarrollo Web

La *World Wide Web* (WWW) es un sistema de interconexión de documentos de hipertexto [Cho14], que pueden ser accedidos a través de Internet, mediante un navegador web. El propósito por el que la web nació, de manos de Tim Berners-Lee, fue hacer un sistema de comunicación en el CERN más efectivo. La web ha pasado por varios cambios, desde el año 1989 hasta hoy en día. A cada uno de estos cambios se les ha llamado versiones de la web.

1. **Web 1.0**
   Fue la primera versión de la web y abarcó desde 1989 hasta 2005. En esta versión, la web se consideraba de «solo-lectura» y proporcionaba muy poca interacción con el usuario que se conectaba a la web. Por tanto, algunas características de esta primera web eran el establecimiento de una presencia online, lo que hacía que la información pudiese estar disponible para cualquier usuario en cualquier momento y la inclusión de páginas web estáticas usando para su creación un lenguaje básico de marcado.

2. **Web 2.0**
   Se trata de la segunda generación de la web y se definió en 2004 como una web en la que además de leer se podía «escribir». La Web 2.0 permitió la participación, colaboración y distribuición de información entre usuarios de la web. Esta web también implicaba cambios en la flexibilidad del diseño web, actualizaciones y contenido colaborativo, entre otros. Se pasó de una web con unos 250000 sitios y unos 46 millones de usuarios a una web con unos 80 millones de sitios y más de mil millones de usuarios globales. En los primeros años de la web 2.0 comenzaron a nacer las redes sociales, las wikis y los blogs.

3. **Web 3.0**
   Se considera su aparición a partir del año 2016 y se conoce como la «web ejecutable». La idea detrás de la web 3.0 es definir una estructura de datos y enlazarlos entre ellos para hacer más efectivo el descubrimiento, la automatización y la integración y reuso de los mismos en varias aplicaciones. A este marco de compartición y reuso de datos se le llama «web semántica». Esta web semántica, permite a las máquinas «entender» y responder a las peticiones humanas basándose en su significado.

### A.5.1 HTML5 + CSS3

HTML (*HyperText Markup Language*) es un lenguaje de marcado que se usa para crear paginas web y aplicaciones [Nie13]. El propósito principal de HTML es proporcionar una descripción semántica del contenido y establecer una estructura de documento, una jerarquía de elementos.

Los documentos HTML5 se pueden escribir usando sintaxis XHTML. HTML5 ofrece nuevas características como pueden ser nuevos elementos, atributos, manejadores de eventos y APIs, para hacer más fácil y sofisticado el desarrollo de aplicaciones web. En vez de usar *Document Type Definition* (DTD), que define la estructura, los elementos y los atributos de un documento XML usa un modelo *Document Object Model* (DOM), que se trata de un modelo de objetos estándar y una interfaz de programación para HTML. Los elementos HTML se tratan de objetos con propiedades, métodos y eventos.

Con la creciente demanda de contenido interactivo en las páginas web, HTML5 introduce APIs para la creación de estas webs. Las APIs estandarizan tareas que tradicionalmente requerían de contenido propietario o programación personalizada para llevar a cabo dichas tareas.

Las APIs más importantes de la especificación HTML5 tienen que ver con la manipulación de vídeo y audio, así como la sincronización multimedia y los subtítulos. Otras APIs importantes se usan para manipular el historial del navegador (avanzar, retroceder y manipular el contenido de dicho historial), para permitir que ciertos recursos de una web estén disponibles de forma *offline* y para añadir ciertos eventos como los *drag and drop*.

Para que el navegador sepa que el documento que va a mostrar es HTML5, basta con añadir la declaración de tipo de documento o *doctype* (véase el Listado A.8). Se puede observar que el número 5 referente a la versión de HTML no se encuentra en la declaración *doctype*. Esto se debe a que la versión se encuentra implícita ya que HTML5 es una evolución natural de los estándares HTML previos.

Listado A.8: **Esqueleto de código HTML5.**

```
<!DOCTYPE html>
<html>
  <head>
    <title> Document Title </title>
  </head>
  <body>
    Content of document . . .
  </body>
</html>
```

Las hojas de estilo o CSS fueron introducidas con el propósito de separar el diseño del contenido [Fra12]. CSS no existió hata que pasaron unos años desde el nacimiento de HTML y desde entonces se ha convertido en la forma estándar de definir la capa de presentación de una página web. CSS3 se trata de una evolución natural de sus versiones anteriores, CSS2 y CSS1 e introduce efectos visuales, como sombras que dan la apariencia de un realce del elemento, esquinas redondeadas, gradientes y otros.

CSS3 es una especificación que se encuentra en contínuo cambio [Gas15]. Algunas partes de la especificación se consideran estables y están implementadas en los navegadores modernos, mientras que otras se consideran experimentales y puede que los navegadores no las soporten aún. Algunos navegadores han creado sus propias propiedades CSS que no están en la especificación oficial.

Listado A.9: **Sintaxis de una regla en CSS.**

```
Selector { property: value; }
```

Listado A.10: **Uso de propiedades experimentales en distintos navegadores.**

```
Selector {
    -moz-property: value;    /* Firefox */
    -ms-property: value;     /* Internet Explorer */
    -webkit-property: value; /* Chrome/Safari */
}
```

### A.5.2 JavaScript

JavaScript es el lenguaje usado para la programación web [Fla11]. Se trata de un lenguaje de alto nivel, interpretado, débilmente tipado (la declaración de una variable no exige la asociación de un tipo de datos) y orientado a objetos y complementa a HTML (que especifica el contenido de una web) y CSS (que establece la presentación de la web), definiendo el comportamiento de la web. El nombre de JavaScript nada tiene que ver con Java, siendo dos lenguajes completamente distintos.

JavaScript posee una librería estándar de funciones para el tratamiento de textos, *arrays*, fechas y expresiones regulares. Sin embargo, no posee funcionalidad para manejar la entrada y la salida siendo las mismas responsabilidad del «entorno del *host*», en el que JavaScript está embebido. Normalmente, este entorno es el navegador web, pero no está limitado al mismo.

Listado A.11: ***Hello world* en JavaScript.**

```
<!DOCTYPE HTML>
<html>
<body>
  <script>
    alert("Hello, world!");
  </script>
</body>
</html>
```

Listado A.12: **Creación de un objeto en JavaScript.**

```
var object = {
        property1: "value",         // String value
        property2: false,           // Boolean value
        property3: [[0,1], [2,3]],  // Array of arrays
        property4: undefined        // Null value
};
```

#### *Frontend*: React

React es una popular librería desarrollada por Facebook y escrita en código JavaScript [Ban17], que se usa para crear interfaces de usuario. Se desarrolló con el objetivo de atajar los desafíos que se presentan en las webs basadas en datos. El código React tiene una apariencia de código HTML dentro de un código JavaScript.

React introduce el concepto de arquitectura basada en componentes como método para encapsular ciertas partes de una interfaz de usuario mucho mayor. Estas partes de la interfaz, se denominan componentes y podemos pensar en ellos como pequeñas características que

forman parte de la interfaz de usuario. Los componentes son independientes entre sí (véase Figura A.12), por lo que un componente puede hacer una llamada al servidor para modificar su valor sin que el resto de componentes se modifiquen y sin, por tanto, tener la necesidad de recargar la web completamente.

JavaScript no es un lenguaje funcional, pero se pueden usar técnicas funcionales en el código JavaScript. React enfatiza la programación funcional por encima de la programación orientada a objetos. Este cambio en el pensamiento puede producir beneficios en la capacidad de *testing* de un código, así como en su rendimiento. Sin embargo, el cambio de mentalidad puede ser complicado en un principio.

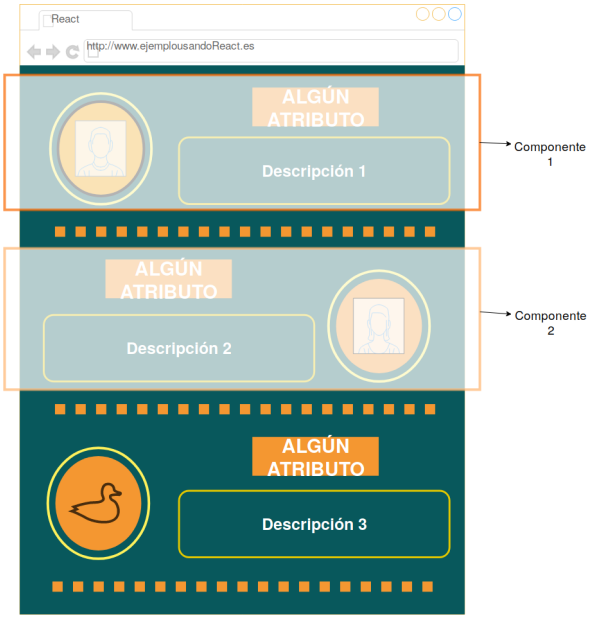



Figura A.12: Componentes en React.

Listado A.13: ***Hello world*** **usando React.**

```
var HelloWorld = React.createClass({
  render: function() {
    return (<h1>Hello, World!</h1>);
  }
});

ReactDOM.render(<HelloWorld/>, document.getElementById('root'));
```

### *Backend*: Node.js

NodeJs es un entorno de ejecución de JavaScript, es decir, un entorno que incluye todo lo necesario para ejecutar un programa que se escribe en lenguaje JavaScript. JavaScript era un lenguaje que sólo se podía ejecutar sobre un navegador, por lo que con el nacimiento de

NodeJs se hace posible ejecutar una aplicación JavaScript sin necesidad de un navegador web.

NodeJs opera usando el motor de ejecución $V8$ de *Chrome*. Este motor se encarga de transformar el código JavaScript en código máquina, más rápido de ejecutar. El entorno de NodeJs está conducido por eventos (véase Figura A.13), con un modelo de entrada/salida no bloqueante (al manejar ficheros o hacer peticiones HTTP a una API, por ejemplo), lo que lo hace ligero y eficiente.

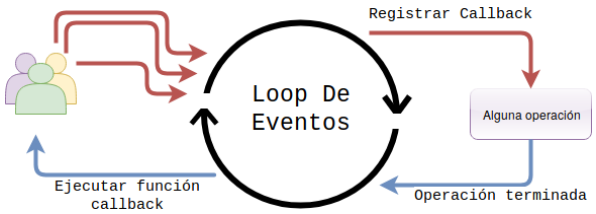



Figura A.13: Modelo de ejecución conducido por eventos en NodeJs.

El gestor de paquetes de Node NPM (*node package manager*) proporciona una forma fácil de gestionar las dependencias en una aplicación Node. NPM permite añadir paquetes (especificando una versión determinada del mismo) a una aplicación de una forma muy sencilla.

Listado A.14: ***Hello world*** **en NodeJs.**

```
1 console.log("Hello, world!");
```

`Express` es un *framework* [Bro14] para construir aplicaciones web sobre Node.js, que proporciona un conjunto de características especialmente atractivas para el desarrollo de webs *single-page* (llamadas así porque se descarga por completo la página en el cliente sin estar contínuamente haciendo peticiones al servidor, por lo que la web es más rápida), webs multipágina y webs híbridas, que mezclan ambos conceptos.

## A.6 *Application Programming Interface* (API)s

Hoy en día muchos de los negocios son digitales. Las APIs [Nij14] se encargan de conectar los procesos de negocio, los servicios, el contenido y los datos con los socios, equipos internos y desarrolladores individuales, de una forma sencilla y segura. Las APIs se han convertido en el estándar de facto para el intercambio de datos por parte de estas empresas.

Por ejemplo, imaginemos una compañía que se dedique a la venta de productos. Un día dado, hacen una oferta de un producto en su web y esa oferta también se actualiza en su aplicación móvil. La aplicación no tiene que tratar con datos internos sobre los precios, tan sólo hace una petición a la API (véase Figura A.14) que proporciona las ofertas de la compañía y ésta responde con la información necesaria.

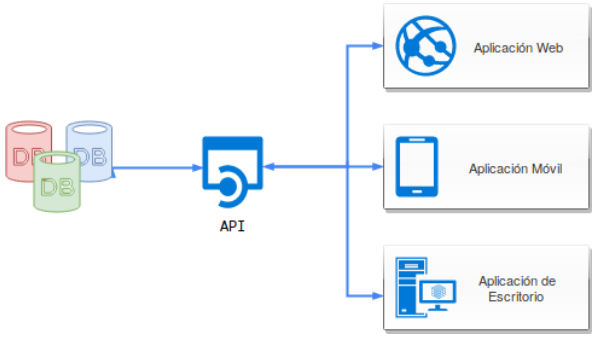



Figura A.14: Esquema básico de funcionamiento de una API.

La principal ventaja del uso de las APIs es el fomento de la ubicuidad. Si una empresa desarrolla una aplicación web en la que se accede a los datos a través de una base de datos tradicional y más tarde quiere desarrollar una aplicación móvil (y otras aplicaciones), el esfuerzo de desarrollo en el acceso de esos datos es mayor que mediante una API que todas las aplicaciones compartirían, ya que tan sólo tendrían que hacer peticiones a la misma para que devolviese la información deseada.

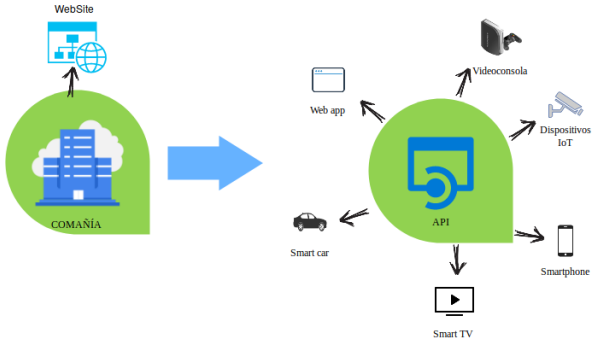



Figura A.15: Ubicuidad en el uso de las APIs.

Con el crecimiento de las aplicaciones móviles, ha aumentado el uso de las APIs. Hoy en día, no resulta raro que una aplicación móvil permita registrarnos e iniciar sesión para acceder a sus servicios mediante redes sociales como *Facebook*, *Twitter* o *Google*. Estas aplicaciones hacen uso de las APIs que dichas empresas ponen a la disposición de los desarrolladores. Normalmente, para hacer uso de una API externa, se necesita una API *Key*. Se trata de una clave que identifica y autentica a un usuario como legítimo para usar los servicios de la API. También se suele utilizar para tasar el uso de la API y cobrar al usuario por el empleo de la misma.

REST (*Representational State Transfer*) se trata de una interfaz entre sistemas que usa el protocolo HTTP para obtener o manipular datos. En una comunicación cliente-servidor podemos usar una comunicación HTTP para realizar la petición de datos. Este tipo de arquitectura de comunicación se denomina arquitectura REST y es muy sencilla. Existen otro tipo de ar-

quitecturas como puede ser *Simple Object Access Protocol* (SOAP), pero su complejidad para manejar los datos es mucho mayor.

Hay que tener en cuenta que una arquitectura REST debe cumplir ciertas propiedades y restricciones. No se pueden publicar servicios RPC (por lo que no se puede, por ejemplo, hacer una llamada a una función expuesta por el servidor). En vez de publicar estos servicios, se publican recursos entendidos como entidades que pueden accederse de forma pública.

### A.6.1 API de AEMET

AEMET (Agencia Estatal de Meteorología) [AEM17] posee una API REST gracias a la cual se pueden descargar de forma gratuita datos meteorológicos. Gracias a esta API un desarrollador puede realizar consultas de una manera programada y períodica a los datos meteorológicos, con el objetivo de usarlos en la aplicación que está desarrollando. Los datos a los que un desarrollador tiene acceso son:

- Datos de observación, radiación y contaminación de fondo.
- Imagenes de radar, mapas de rayos y productos derivados de satélite.
- Climatología, valores normales y productos climatológicos.
- Predicciones normalizadas, específicas y marítimas.
- Mapas significativos, de análisis y previstos.
- Avisos de fenómenos meteorológicos adversos e índices de incendios.

Para tener acceso a la API de AEMET es necesario obtener primero una «API *Key*». Esta clave servirá para acceder al servicio que proporciona la API y dura 90 días desde que la generación de la misma. La petición de la clave se puede realizar en la web `https://opendata.aemet.es/centrodedescargas/altaUsuario` y se enviará al correo que se introduzca en dicha web (véase Figura A.16).

Listado A.15: **Inventario con todas las características de todas las estaciones climatológicas.**

```
1  from requests import request
2
3  url = "https://opendata.aemet.es/opendata/api/valores/climatologicos/inventarioestaciones/
       todasestaciones/"
4
5  querystring = {"api_key":"........"}
6  headers = { 'cache-control': "no-cache" }
7
8  response = request("GET", url, headers=headers, params=querystring)
9  print(response.text)
```


Figura A.16: API *key* para el uso del servicio AEMET.

### A.6.2 API de *Google Maps*

*Google Maps* nació en febrero del año 2005 a manos de la compañía *Google* [Sve10] revolucionando la forma en la que se usaban mapas en las páginas web, permitiendo a los usuarios navegar por el mapa. Las soluciones para tratar mapas antes del nacimiento de *Google Maps* requerían servidores de mapas especiales, que eran muy caros. La primera versión de la API de *Google Maps* fue concebida en junio de ese mismo año.

Las localizaciones en el mundo se expresan en forma de coordenadas. Existen distintos sistemas de coordenadas, sin embargo *World Geodetic System 84* (WGS 84) es el usado por *Google Maps* y también por GPS. Las coordenadas se expresan en términos de latitud y longitud. La latitud se mide desde el sur hacia el norte y la longitud se mide desde el oeste hacia el este. En el ecuador, la latitud es 0, por lo que todo valor de latitud por debajo del ecuador (relativo al hemisferio sur) será negativo. El valor 0 de longitud se encuentra en el meridiano de *Greenwich* y toda longitud al oeste de este meridiano será negativa. El formato de las coordenadas es el de dos números decimales separados por una coma. El primer número se refiere a la latitud y el segundo a la longitud.

Para usar la API de *Google Maps* es necesario obtener previamente una «API Key» que se usará para hacer un seguimiento de las peticiones a la API relacionadas con un proyecto para la facturación de los servicios. Normalmente se usará una «API Key» distinta para cada uno de los proyectos que necesiten hacer uso de la API de *Google Maps*. En el Listado A.16 se puede observar el lugar en el que irá la clave de la API (`key=YOUR_API_KEY`). Sin esta clave, no se podrá disfrutar de los servicios de la API.

Listado A.16: **Ejemplo básico del uso de la API de *Google Maps*, para visualizar en un mapa Ciudad Real.**

```
<!DOCTYPE html>
<html>
  ...
  <body>
    <div id="map"></div>
    <script>
      var map;
      function initMap() {
        map = new google.maps.Map(document.getElementById('map'), {
```

```
            center: {lat: 38.955 , lng: -3.98},
            zoom: 8
          });
        }
      </script>
      <script src="https://maps.googleapis.com/maps/api/js?key=YOUR_API_KEY&callback=initMap"
      async defer></script>
    </body>
  </html>
```

Anexo B

# Diagrama de clases

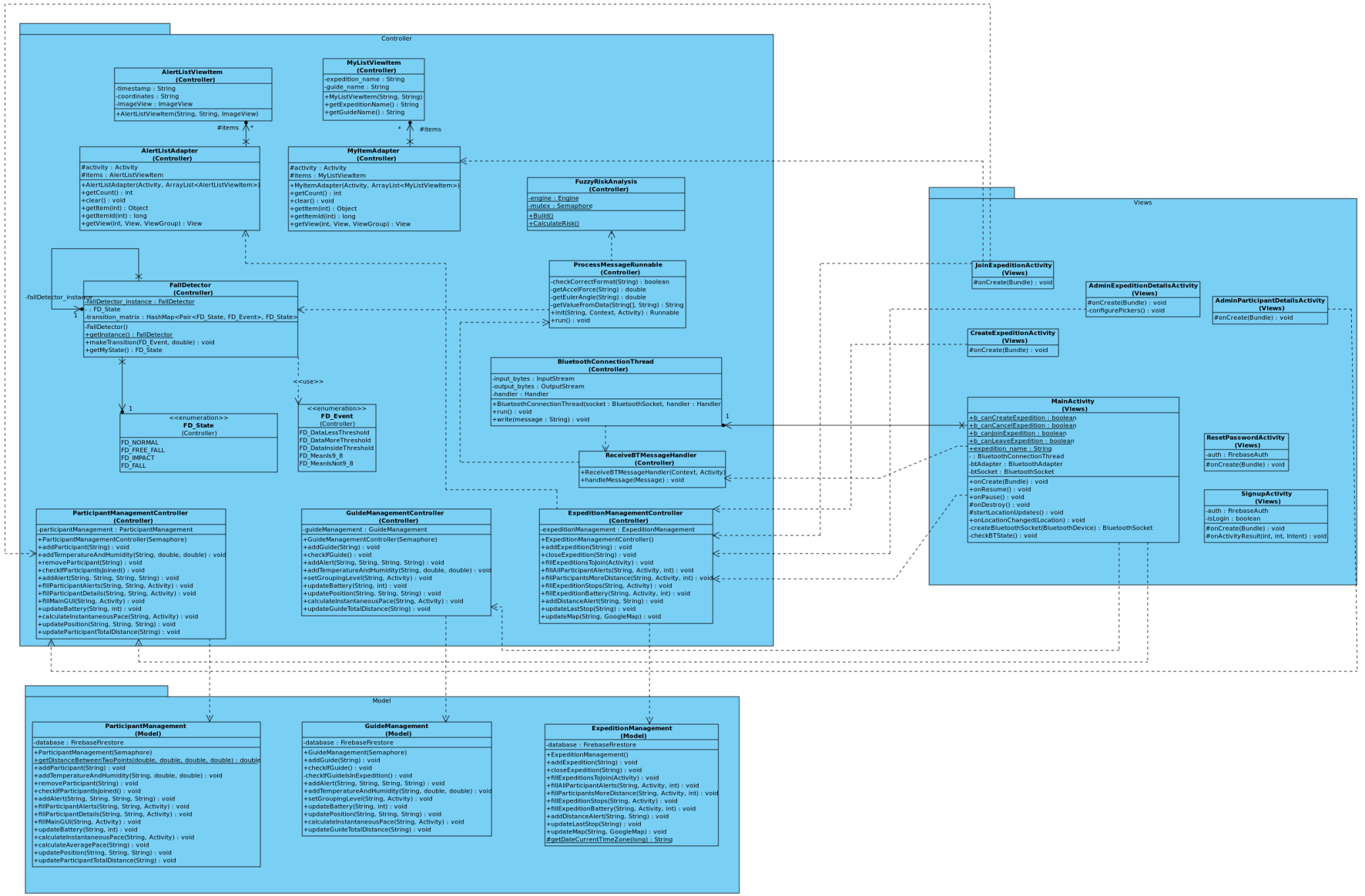



Figura B.1: Diagrama de clases completo.

# Bibliografía


[AB08] G.M. Lyons A.K. Bourke. A threshold-based fall-detection algorithm using a bi-axial gyroscope sensor. *Medical Engineering & Physics*, páginas 84–90, 2008.

[AEM17] AEMET. Preguntas Frecuentes del Servicio AEMET OPENDATA. Technical report, Ministerio de Agricultura y Pesca, Alimentación y Medio Ambiente, 2017.

[Ali16] C. Aliferi. *Android Programming Cookbook*. Java Code Geeks, 2016.

[Azi18] Rashid S. Aziz, M. Risk taking behavior and interpersonal relationship of adrenaline junkies: A qualitative study. *Indian Journal of Health and Well-being*, 3:384–391, 2018.

[Ban15] Shiloh M. Banzi, M. *Getting Started with Arduino*. Maker Media, 2015.

[Ban17] Porcello E. Banks, A. *Learning React. Functional Web Development with React and Redux*. O'Reilly Media, 2017.

[Bel13] C. Bell. *Beginning Sensor Networks with Arduino and Raspberry Pi*. Apress, 2013.

[Bou07] O'Brien J. Lyons-G. Bourke, A. Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm. *Gait & posture*, 26 2:194–9, 2007.

[Bro13] G. Brown. *Discovering the STM32 Microcontroller*. OpenLibra, 2013.

[Bro14] E. Brown. *Web Development with Node & Express*. O'Reilly Media, 2014.

[Bur15] Ed Burnette. *Hello, Android: Introducing Google's Mobile Development Platform*. 2015.

[Cab16a] Polo M. Rodríguez-M. Caballero, I. *Apuntes de Ingeniería del Software II*, capítulo Metodologías de desarrollo ágiles: Scrum, páginas 6–12. 2016.

[Cab16b] Polo M. Serrano-M. Caballero, I. *Apuntes de Ingeniería del Software II*, capítulo Procesos de Ingeniería del Software, páginas 21–22. 2016.

[Cho14] N. Choudhury. World Wide Web and Its Journey from Web 1.0 to Web 4.0. *International Journal of Computer Science and Information Technologies*, 5:8096–8100, 2014.

[CSD] Histórico de Licencias. `http://www.csd.gob.es/csd/estaticos/asoc-fed/historico-de-licencias.pdf`. CSD.

[dD12] Ministerio de Defensa. Establecimiento de una red de control y seguimiento por medio satélite (SPOT). Technical report, 2012.

[Esf14] et al. Esfahani, M. The Influence of Spirituality and Physical Activity Level on Responsible Behaviour and Mountaineering Satisfaction on Mount Kinabalu, Borneo. *Current Issues in Tourism*, páginas 1162–1185, Dec 2014.

[Ewe85] A. Ewert. Why People Climb: The Relationship of Participant Motives and Experience Level to Mountaineering. *Journal of Leisure Research*, 17:241–250, Jan 1985.

[Fal94] Brugger H. Adler-Kastner L. Falk, M. Avalanche survival chances. *Nature Publishing Group*, 368, 1994.

[FD17] et al. Frajberg D. Heterogeneous information integration for touristic augmented reality mobile apps. *International Conference on Data Science and Advanced Analytics*, 2017.

[FED] Tarjeta Federativa FEDME. `http://www.fedme.es/index.php?mmod=staticContent&IDf=96`. FEDME.

[FED17a] FEDME. Accidentalidad en Deportes de Montaña de Federados FEDME. páginas 1–28, 2017.

[FED17b] FEDME. Estudio del perfil de deportistas federados y aficionados a los deportes de montaña. páginas 1–29, Mayo 2017.

[Fla11] D. Flanagan. *JavaScript: The Definitive Guide*. O'Reilly Media, 2011.

[Fra12] B. Frain. *Responsive Web Design with HTML5 and CSS3*. Packt Publishing, 2012.

[Gas15] P. Gasston. *The Book of CSS3. A Developer's Guide to the Future of Web Design*. No Starch Press, 2015.

[Ger15] Clifton C. Gerber, A. *Learn Android Studio. Build Android Apps Quickly and Effectively*. Apress, 2015.

[Gon11] C. González. *Lógica Difusa. Una introducción práctica. Técnicas de Softcomputing*. 2011.

[Hua07] Rudolph L. Huang, A. *Bluetooth Essentials for Programmers*. Cambridge University Press, 2007.

[Kar14] Karvinen T. Karvinen, K. *Getting Started with Sensors*. Maker Media, 2014.

[Lim14] Park C. Kim-N. Kim S. Yu Y. Lim, D. Fall-Detection Algorithm Using 3-Axis Acceleration: Combination with Simple Threshold and Hidden Markov Model. *Journal of Applied Mathematics*, 2014, 2014.

[Liu] T. Liu. DHT22 Datasheet. Digital-output relative humidity & temperature sensor/module. Technical report, Aosong Electronics Co.,Ltd.

[Mag] Raspberry Pi Magazine. The Official Raspberry Pi Projects Book.

[Mao17] Ma X. He-Y. Luo J. Mao, A. Highly Portable, Sensor-Based System for Human Fall Monitoring. *Sensors*, 17, 2017.

[Mar11] M. Margolis. *Arduino Cookbook*. O'Reilly Media, 2011.

[McR13] M. McRoberts. *Beginning Arduino*. Technology in Action, 2013.

[Mon12] A. Mondragon. So Many Educational Microcontroller Platforms, so Little Time! Technical report, Rochester Institute of Technology, Jun 2012.

[MS04] D. Moscoso Sánchez. El proceso de institucionalización del montañismo en España. *Acciones e Investigaciones Sociales*, 19:5–29, Marzo 2004.

[Nie13] J. Niederst. *HTML5 Pocket Reference*. O'Reilly Media, 2013.

[Nij14] Pagano B. Nijim, S. *APIs For Dummies*. Apigee, 2014.

[Nor18] D. Norris. *Programming with STM32. Getting Started with the Nucleo Board and C/C++*. McGraw-Hill Education, 2018.

[Sha00] M. Shaaban. Microcontroller Basics; An Example: The Motorola 68HC12. Technical report, Rochester Institute of Technology, Feb 2000.

[Smy17] N. Smyth. *Android Studio 3.0 Development Essentials*. Payload Media, 2017.

[Sve10] G. Svennerberg. *Beginning Google Maps API 3*. Apress, 2010.

[Wat16] S. Watkiss. *Learn Electronics with Raspberry Pi Physical Computing with Circuits, Sensors, Outputs, and Projects*. Apress, 2016.

[Yam15] S. Yamanoor. *Raspberry Pi Mechatronic Projects*. Packt Publishing, 2015.