



Universidad de
Oviedo



ESCUELA POLITÉCNICA DE INGENIERÍA DE GIJÓN

GRADO EN TECNOLOGÍAS Y SERVICIOS DE LA TELECOMUNICACIÓN

ÁREA DE TELEMÁTICA

TRABAJO FIN DE GRADO N°1503_23

APLICACIÓN MÓVIL PARA LA MONITORIZACIÓN DE CAÍDAS

MARTÍNEZ ÁVILA, Marco
TUTOR: JOSÉ ÁNGEL VALLEJO PINTO

FECHA: Julio del 2015

Índice

1. Introducción	4
1.1. ¿QUÉ ES FALLAPP?	4
1.2. USUARIO FINAL	4
1.3. ¿POR QUÉ PERSONAS DE TERCERA EDAD?	4
1.4. SISTEMA OPERATIVO MÓVIL	4
1.5. OBJETIVO	5
1.6. REQUISITOS	5
1.7. FUNCIONAMIENTO	6
2. Antecedentes	7
3. Estudio previo	10
3.1. USO DE SENsoRES	10
3.2. EL ACELERÓMETRO	11
3.3. APLICACIONES WEB, HÍBRIDAS Y NATIVAS	15
3.4. APACHE CORDOVA	17
3.5. PROGRAMACIÓN Y SERVICIOS EN ANDROID	19
3.5.1. ACTIVIDADES	19
3.5.2. INTENTS	20
3.5.3. SERVICIOS	20
3.6. CORDOVA CLI E IDEs	22
3.6.1. ECLIPSE	22
3.6.2. ANDROID STUDIO	23
3.6.3. WEBSTORM	23
3.6.4. INTELLIJ IDEA	23
3.7. GIT	24
3.8. ANÁLISIS DE DISPOSITIVOS Y APLICACIONES EXISTENTES	24
3.8.1. DETECTOR DE CAÍDAS SENSE4CARE	24
3.8.2. VIGI'FALL	25
3.8.3. SISTEMA DE VIDEOVIGILANCIA DE LA SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING	26
3.8.4. GEA	27
3.8.5. ISCREAM	28
3.8.6. FADE: FALL DETECTOR	29
4. Desarrollo de la aplicación	31
4.1. REQUISITOS DE UNA APLICACIÓN ANDROID	31
4.2. INSTALACIÓN DE CORDOVA	31
4.2.1. CONFIG.XML	34
4.3. BUENAS PRÁCTICAS EN EL DESARROLLO CON CORDOVA	34
4.4. ESTRUCTURA DE FALLAPP	36
4.5. PLUGINS, AÑADIENDO FUNCIONALIDAD	37
4.6. ASISTENTE DE CONFIGURACIÓN	38
4.6.1. JAVASCRIPT, JQUERY Y LOS EVENTOS	40

4.7. PÁGINA PRINCIPAL	45
4.8. VISTA DE ALERTAS	54
4.9. ESCALABILIDAD EN LA INTERFAZ WEB	55
4.10. PRUEBAS DE RENDIMIENTO	57
4.10.1. USO DE SELECTORES	58
4.10.2. ESTILOS	59
4.11. ORGANIZACIÓN DEL CÓDIGO JAVASCRIPT	60
4.12. DESARROLLO DEL ALGORITMO DE DETECCIÓN DE CAÍDAS	61
4.12.1. TOMA DE DATOS CON EL ACCELERÓMETRO	61
4.12.2. ANÁLISIS DE CAÍDAS	63
4.12.3. FUNDAMENTO DEL ALGORITMO	70
4.12.4. IMPLEMENTACIÓN DEL ALGORITMO	73
4.12.5. ALERTAS	77
4.12.6. SERVICIO EN SEGUNDO PLANO	79
4.13. COMUNICACIÓN ENTRE LA INTERFAZ WEB Y EL NÚCLEO ANDROID	80
4.13.1. ACTIVIDADES Y SU CORRESPONDIENTE INTERFAZ	80
4.13.2. PLUGIN APIS	81
4.14. DIAGRAMAS	83
4.15. POSIBLES MEJORAS	93
5. Pruebas de funcionamiento	95
5.1. CAIDA DEL SMARTPHONE DESDE UNA MESA	95
5.2. CAÍDA DEL SMARTPHONE AL SACARLO DEL BOLSILLO	95
5.3. CAÍDA DEL SMARTPHONE GIRANDO SOBRE SÍ MISMO	96
5.4. CAMINAR	96
5.5. TROTAR	97
5.6. SENTARSE EN UNA SILLA	97
5.7. LEVANTARSE DE UNA SILLA	97
5.8. LLEVAR EL MÓVIL A LA OREJA	98
5.9. CAMINAR Y CAER	98
6. Planificación	100
7. Presupuesto	101
8. Conclusiones	102

1. Introducción

En este primer capítulo se expone la **información básica** sobre FallApp, la aplicación desarrollada en este Trabajo Fin de Grado. Se analizarán distintos aspectos como el por qué de la idea, la justificación de algunos aspectos de funcionamiento o el objetivo a conseguir con este proyecto.

1.1. ¿QUÉ ES FALLAPP?

FallApp es una aplicación móvil que **detecta caídas y envía alertas de forma automática**. Para ello, hace uso del acelerómetro, sensor del que disponen la inmensa mayoría de smartphones actuales, monitorizando en segundo plano los valores devueltos por el mismo.

1.2. USUARIO FINAL

FallApp es una **aplicación orientada a personas de la tercera edad**, tanto en su fundamento como en su interfaz. Teniendo en cuenta la relativa facilidad con la que este grupo de la población sufre incidentes físicos, como el concerniente a este TFG, detectarlos de forma inmediata resulta crucial.

El usuario podrá configurar varios números de teléfono a los que avisar en caso de caída, a los que se les enviará una serie de mensajes de texto. De cara a facilitar su localización, existe la opción de añadir la posición obtenida por GPS o por red, si se tiene habilitado el servicio pertinente en los ajustes del smartphone.

1.3. ¿POR QUÉ PERSONAS DE TERCERA EDAD?

La **idea** que derivó en el desarrollo de este TFG tiene su origen en una persona de 90 años. Sufrió una **caída en una zona** en principio **desierta**, teniendo la inmensa suerte de que alguien la vio y pudo llamar a la ambulancia. ¿Qué habría ocurrido si hubiera pasado inadvertida?

Estas personas no tienen la movilidad, la resistencia y las habilidades con los móviles de los jóvenes, factores que contribuyen a que una caída en un lugar solitario pueda resultar fatal. El envío de alertas de forma automática e inmediata puede contribuir a que el riesgo se reduzca de forma significativa. Este aspecto será tratado con más detenimiento en el [capítulo 2](#).

1.4. SISTEMA OPERATIVO MÓVIL

Android es la plataforma escogida para este desarrollo. Cada vez es más común que las personas mayores hagan uso un smartphone de gama media-baja, por lo que el iPhone de Apple fue considerado desde el principio un sistema operativo poco conveniente. Por su parte, como se puede ver en la Figura 1.1, Windows Phone tiene una

cuota de mercado muy reducida.

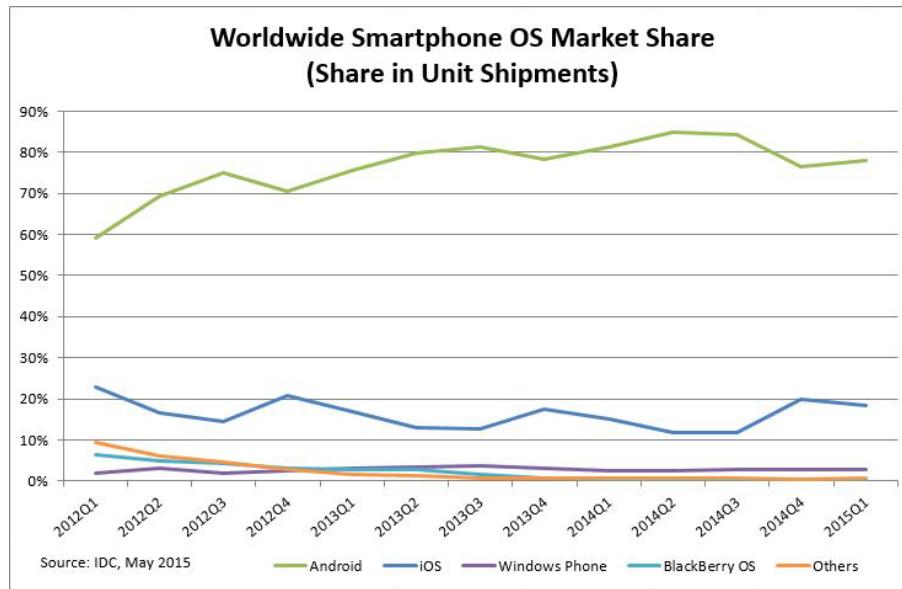


Figura 1.1.- Porcentaje de unidades de smartphones enviadas (vendidas) por las compañías en los últimos años.

Estos hechos, y el disponer el desarrollador de un móvil con Android, desembocan en la decisión de que hacer uso del sistema operativo de Google es la mejor opción.

1.5. OBJETIVO

Teniendo en cuenta lo expuesto anteriormente, se puede determinar que el **objetivo** de FallApp es servir como **sistema de monitorización** de personas de la tercera edad, haciendo uso de la versatilidad y la funcionalidad de los smartphones actuales para enviar mensajes de alerta en caso de caída.

1.6. REQUISITOS

Los teléfonos móviles inteligentes cuentan, desde hace varios años, con distintos sensores que permiten aumentar su funcionalidad, más allá de realizar llamadas, mandar SMS o conectarse a Internet. Alguno de ellos son:

- **Sensor de luz:** permite regular el brillo de la pantalla en función de la luminosidad del ambiente, optimizando así el consumo de batería.
- **Sensor de proximidad:** como su propio nombre indica, detecta objetos próximos a él. Su funcionalidad más extendida es apagar la pantalla al acercar el móvil a la oreja durante una llamada telefónica, previniendo así toques indeseados.
- **Termómetro:** permite controlar la temperatura interior del móvil y su batería, con el fin de evitar potenciales daños por sobrecalentamiento.

- **Giroscopio:** elemento que permite medir la rotación de un smartphone. Suele utilizarse para identificar gestos que ejecuten determinadas funciones.
- **Acelerómetro:** sensor que mide la aceleración del smartphone. Su uso más común consiste en determinar la orientación del dispositivo para interactuar con aplicaciones como juegos, o alternar entre el modo vertical y el modo apaisado. Es el **sensor utilizado para determinar cuándo se produce una caída**.

Todos los smartphones actuales cuentan con diversos sensores, normalmente con más o menos en función de su precio. Sus usos, como se ha comentado más arriba, son de lo más diverso; sin embargo, aunque dispositivos de gama baja puedan carecer de alguno, a día de hoy se puede dar por hecho que cualquier teléfono inteligente cuenta con una serie de funcionalidades que están siempre presentes.

De esta forma, y teniendo en cuenta la extensión de esta clase de terminales entre la población (incluyendo la envejecida), es posible **aprovechar las oportunidades** que nos brindan para desarrollar la aplicación en la que se basa este TFG.

Concretando, los requisitos para que esta funcione son:

- **Disponer de acelerómetro:** condición cumplida por todos los smartphones.
- **Conexión a la red móvil para enviar SMS:** se comprobará a la hora de enviar los mensajes de texto.
- **Sistema operativo Android:** por los motivos expuestos anteriormente.
- **[Opcional] Disponer de GPS:** condición cumplida por todos los smartphones. Se usará en caso de que se obtenga la posición en un plazo de tiempo asumible.

1.7. FUNCIONAMIENTO

El funcionamiento de la aplicación se puede subdividir en **tres etapas** bien diferenciadas: la monitorización y análisis de valores devueltos por el acelerómetro, la detección de la caídas y el envío de alertas.

- **Monitorización y análisis de valores devueltos por el acelerómetro:** el sistema operativo Android proporciona un mecanismo para ejecutar **tareas de larga duración en segundo plano: los servicios**. Por medio de un servicio es posible mantener un módulo de la aplicación corriendo indefinidamente, de forma que se obtengan y procesen los valores del sensor.
- **Detección de la caída:** gracias a un algoritmo desarrollado a partir de pruebas empíricas, es posible determinar con cierto grado de precisión cuándo se cae una persona. Esto excluye, principalmente, situaciones en las que sólo el smartphone sufre una caída.
- **Envío de alertas:** en función de la detección anterior, la aplicación puede enviar SMSs (incluyendo la posición opcionalmente) a los números de teléfono configurados. Esta acción se realiza inmediatamente después de producirse la caída, minimizando los riesgos derivados de la misma.

2. Antecedentes

Aunque el caso comentado en la sección [¿Por qué personas de tercera edad?](#) es concreto y personal, la ocurrencia de una caída en ancianos es un hecho al que muchas personas se enfrentan en el día a día. De esta forma, **esa experiencia fue el desencadenante de la aplicación FallApp**, derivando en un estudio más profundo de las causas y las consecuencias de las caídas en ancianos.

Durante décadas se ha experimentado un **incremento constante de la esperanza de vida**; en los últimos años y según el [Instituto Nacional de Estadística](#), en España es de 82.8 años. **La gente vive más**, lo que conlleva un aumento de los **riesgos producidos por edad**, desde enfermedades hasta las caídas tratadas en este TFG, e incluso las repercusiones que tienen posteriormente. En la [web Espacio Mayores](#), página especializada en la información y documentación sobre este grupo poblacional, se especifica un número de 8.572.779 personas de 65 años o más. Este valor equivale a un 18.40 % de la población, cifra que no deja de aumentar.

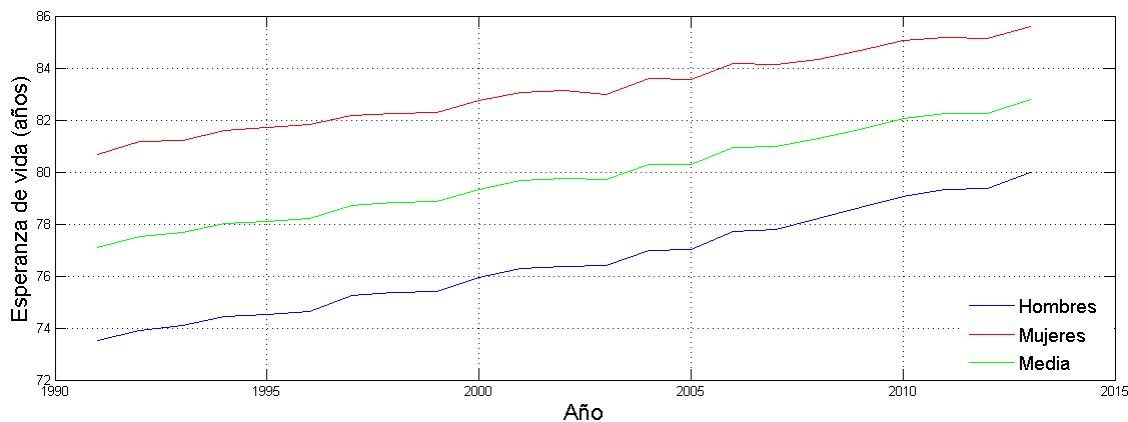


Figura 2.2. Evolución de la esperanza de vida al nacer.

En diversos medios y estudios, entre los que se incluye [este artículo](#) en la página web de Mapfre, se estima que aproximadamente un **30 % de las personas mayores de 65 años se caen una vez al año**. Este porcentaje se incrementa hasta un 50 % cuando el individuo iguala o supera los 80 años. Además, es más probable -2 ó 3 veces más- que se repita esta incidencia a lo largo del año siguiente con los ancianos que se han caído con anterioridad.

Continuando con los datos, la doctora Carmen Pablos Hernández, geriatra del Hospital de Salamanca, comenta según [este artículo](#) que entre un 1 % y un 5 % de los mayores de 65 años presenta lesiones graves al sufrir uno de estos percances. Este porcentaje se incrementa significativamente a medida que aumenta la edad, siendo de hasta un 35 % en mayores de 75 años y hasta un 50 % en mayores de 80.

Por los motivos expuestos con anterioridad, resulta apreciable que **los ancianos son propensos a caerse con relativa facilidad**, derivando este hecho en los problemas que se comentarán más adelante. Las causas, por su parte, pueden catalogarse según la siguiente clasificación:

- **Causas intrínsecas:** aquellas que dependen del anciano, de su estado de salud.

- **Causas extrínsecas:** las que dependen del entorno, de factores sociales y ambientales.

Dentro del primer grupo se engloba la mayoría de patologías que pueda sufrir el individuo. Algunos ejemplos se detallan a continuación:

- **Deterioro físico:** a lo largo de la infancia, **el cuerpo humano aprende a maneterse en pie** haciendo uso de los músculos, de la información sensorial y de diversos estímulos, convirtiéndose a lo largo de esta etapa en una acción completamente automática e inconsciente; sin embargo, **la edad puede alterar de forma significativa estas percepciones**, tanto por la pérdida de reflejos y el fallo de los sentidos como por el deterioro muscular. De esta forma, un anciano puede encontrar dificultades a la hora de mantener el centro de gravedad, pudiendo esto derivar en una caída.
- **Aumento del tiempo de reacción:** este punto está ligado al anterior, al tardar estas personas más tiempo en ejecutar una acción para disminuir el riesgo.
- **Enfermedades:** cualquier tipo de enfermedad puede ser el **desencadenante de una caída**, ya sea neurológica, como epilepsia o demencia; reumatólogica, como la artrosis o la artritis reumatoide; o cardiovascular, como las arritmias o la insuficiencia cardíaca.
- **Medicación:** como destaca el doctor Alfonso González, geriatra del Hospital de Salamanca, “los fármacos juegan un papel muy importante como factor de riesgo de caídas. Especialmente psicótropos, antihipertensivos, antiarrítmicos y los diuréticos, ampliamente utilizados en población anciana”. Pueden alterar la conciencia, el equilibrio y la atención, así como repercutir negativamente en el aparato locomotor.
- **Lesiones previas:** la existencia de daños anteriores es un motivo que puede desembocar en una caída. Más aún, existen casos en los que un anciano no reconoce sufrir una lesión y se lo oculta a familiares y personas queridas. De esta forma, es mucho más complicado prevenir cualquier daño.

Por lo que respecta a las causas extrínsecas, la siguiente lista comenta algunos casos que forman parte de este grupo:

- **Mala adaptación del hogar:** aspecto en el que se incluye una escasa iluminación, falta de material adecuado como pasamanos o agarraderas en las duchas, o la existencia de escaleras en la casa.
- **Compañía en la vivienda:** en función de con quién viva el individuo, es más o menos probable que tenga lugar un incidente de esta clase.
Según [un estudio](#) de las Fundaciones Pfizer e IAVANTE, el 34,5 % de las personas que se caen viven con amigos; el 29,8 % solas; el 29,6 % con sus nietos; el 22,3 % con su cónyuge; el 18,9 % con otro familiar; y el 14,7 % con sus hermanos.
- **Infraestructura y condiciones externas:** desperfectos en la obra pública, calles mal iluminadas, excesiva elevación de escalones y/o aceras, prisa por llegar a algún lugar, condiciones meteorológicas adversas...

Incluso más importante que las causas de una caída son las consecuencias. Estas pueden ir desde un simple susto hasta la muerte, por lo que es siempre importante tenerlas en cuenta. Se pueden dividir, a grandes rasgos, en físicas, psicológicas y sociales:

- **Consecuencias físicas:** lesiones que engloban las conocidas fracturas, contusiones o heridas. Las primeras son los daños más frecuentes, ocurriendo entre un 2 % y un 10 % de las ocasiones, teniendo la fractura de cadera, antebrazo y pelvis su origen, la mayoría de las ocasiones, en una caída.
- **Consecuencias psicológicas:** están relacionadas con las sensaciones y los sentimientos posteriores al incidente. Es muy habitual que **el individuo tenga miedo** a que se vuelva a repetir, lo que puede desembocar en cambios de hábitos y comportamientos que, a corto o medio plazo, repercutan negativamente.

También es posible una **sobreprotección** por parte de la familia y personas cercanas, derivando este hecho en una pérdida de autonomía no beneficiosa. Además, no es extraño que se den casos de ansiedad o depresión, en ocasiones debidos a un **sentimiento de inutilidad**.

Estos sucesos se ven agravados, como se ha comentado con anterioridad, si la persona opta por ocultar el accidente y sus posibles consecuencias.

- **Consecuencias sociales:** como la necesidad de ayuda para las tareas cotidianas, el ingreso en el hospital o en una residencia.

Algunas etapas de la caída también tienen mucha importancia, más allá del impacto. Por ejemplo, como comenta la doctora Carmen Pablos en [el artículo](#) citado más arriba: “Muchas veces nos olvidamos, por un lado, de las secuelas que se derivan de la **estancia prolongada en el suelo tras la caída** (hipotermia, deshidratación, etcétera), y de la inmovilidad secundaria (contracturas, rigidez, úlceras por presión, trombosis, estreñimiento...)”.

3. Estudio previo

Este capítulo recoge todos los estudios, investigaciones realizadas e información obtenida previos al comienzo del desarrollo de la aplicación móvil.

- **Sensores utilizados en la detección de caídas:** se estudiarán las opciones de las que disponen los smartphones de cara a detectar caídas.
- **Aplicaciones web, híbridas y nativas:** distintas posturas o metodologías existentes en el mercado en lo que se refiere a programación de aplicaciones móviles.
- **Entornos de desarrollo:** breve vistazo a los entornos de desarrollo disponibles para la programación de aplicaciones Android.
- **Análisis de alternativas:** repaso a diversos dispositivos de monitorización existentes en el mercado.

3.1. USO DE SENSORES

Un sensor es, según define la RAE, un “dispositivo que detecta una determinada acción externa, temperatura, presión, etc., y la transmite adecuadamente”. Entre todos los disponibles en los smartphones actuales, existen dos que tienen mucho potencial en la detección de caídas: **el giroscopio y el acelerómetro**.

El primero es un elemento que permite medir la orientación y la rotación del dispositivo. Se comenzó a implementar en teléfonos inteligentes a partir del 2010, a raíz de su inclusión en el iPhone 4 de Apple.

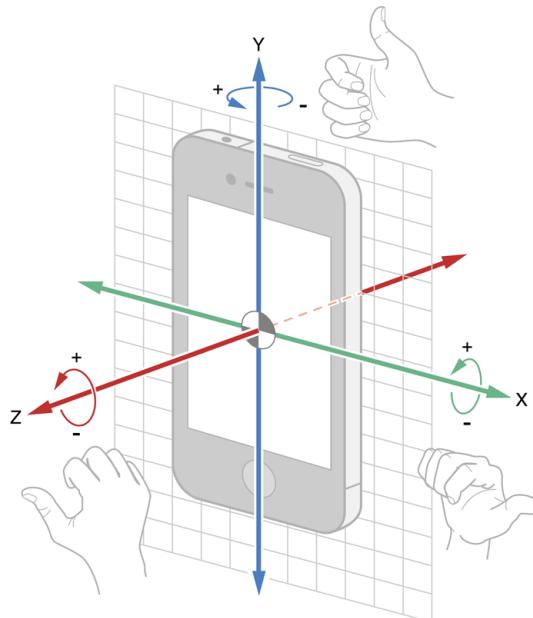


Figura 3.3. Esquema explicativo del giroscopio de un smartphone.

El esquema explicativo del mismo se puede ver en la Figura 3.3. Su funcionamiento se basa en el **momento angular**, pudiendo detectar **cuándo el móvil es rotado a**

lo largo de cualquiera de los 3 ejes del sistema de coordenadas. Algunos fabricantes, como Motorola, están haciendo uso de este sensor para lanzar aplicaciones: un doble giro de muñeca (rotaciones en un eje), permite abrir la cámara teniendo la pantalla apagada.

La trayectoria y los movimientos que describe un móvil cuando cae al suelo, bien sea al haber sido lanzado por los aires o al estar en el bolsillo de una persona, presentan una aleatoriedad significativa; existe la opción de que, al caer de una mesa, se desplace a lo largo del eje vertical hasta el momento del impacto, en cuyo caso el giroscopio no detectaría rotación alguna. A su vez, es posible que **caiga girando sobre sí mismo**, siendo este movimiento detectado por el sensor. De esta forma, si bien este último no podría garantizar exactitud en la detección de caídas, **podría ser un buen complemento** que contribuyera a aumentar la fiabilidad de la aplicación.

Sin embargo, existe un inconveniente importante en el uso del giroscopio: **no está presente en todos los smartphones actuales**. Aunque la mejoría al respecto en referencia a hace años es significativa, puesto que cada vez se encuentra en más terminales del mercado, muchos dispositivos de gama baja todavía no cuentan con él. Por lo tanto, y dado que su uso no es imprescindible, se consideró mejor opción utilizar en exclusiva el sensor explicado a continuación.

3.2. EL ACELERÓMETRO

El acelerómetro es un sensor que mide la aceleración (o cambios de velocidad) en el smartphone con respecto a la superficie terrestre. De esta forma, es posible conocer en todo momento **la orientación** (no la rotación) del dispositivo, tomando como referencia la aceleración de la gravedad, aproximadamente 9.8 m/s^2 .

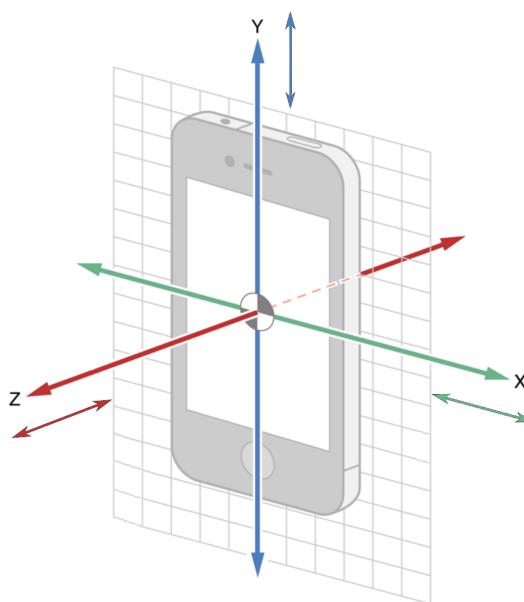


Figura 3.4. Esquema explicativo del acelerómetro de un smartphone.

Como se puede apreciar en la Figura 3.4, y en contraposición a la Figura 3.3, el primero detecta **variaciones lineales**, mientras que el segundo hace lo propio con las

variaciones angulares. El sensor tratado en esta sección, al igual que el smartphone al que pertenece y todos los cuerpos de la Tierra, **está afectado por la fuerza de la gravedad**. De esta forma, en relación a las variaciones con respecto a la misma, se pueden detectar eventos como la caída del móvil o el impacto contra el suelo.

La aceleración, según su definición física, es la **variación de la velocidad con respecto al tiempo**. De este modo, es posible calcular la misma a partir de estos dos últimos parámetros, respondiendo la aceleración media a la siguiente ecuación:

$$\vec{a} = \frac{\Delta \vec{v}}{\Delta t} \quad (1)$$

Sin embargo, Newton enunció en el siglo XVII su segunda ley, o Ley de la Fuerza. En ella, **relaciona la masa de un objeto con la fuerza y la aceleración que actúan sobre él**, según la siguiente ecuación:

$$\vec{F} = m\vec{a} \longrightarrow \vec{a} = \frac{\vec{F}}{m} \quad (2)$$

De esta forma, es posible **determinar la aceleración** de un cuerpo prescindiendo de las variables velocidad y tiempo, y es tal y como lo hace el acelerómetro de un smartphone: **midiendo fuerzas**.

Existen distintos tipos de acelerómetros, que aunque comparten los mismos principios, funcionan de forma distinta. Entre ellos se encuentran algunos como los piezoelectrómicos o los capacitivos, pero **explicar el comportamiento** de los mismos resulta más esclarecedor si se toma como base el mecánico.

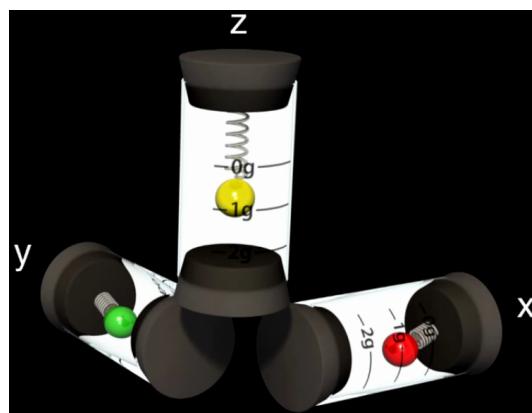


Figura 3.5. Explicación simplificada del funcionamiento de un acelerómetro.

Partiendo de la Figura 3.5, se tienen compartimentos con una masa unida al extremo de un muelle. En función de cuánto se estire o encoja este último, el sensor es capaz de determinar la aceleración a la que está sometido en ese momento, tal y como explica Bill Hammack en [este vídeo de YouTube](#). Cabe destacar que los 3 muelles son unidireccionales, en contraposición a lo que pueda parecer atendiendo a la imagen.

De esta forma, a medida que el smartphone se mueve, los 3 muelles se estirarían o comprimirían en función de la fuerza de la gravedad; sin embargo, como se ha comentado anteriormente, un móvil no tiene dentro 3 compartimentos con muelles y 3 masas, sino que realiza su misma labor por medio de -generalmente- un **acelerómetro capacitivo**.

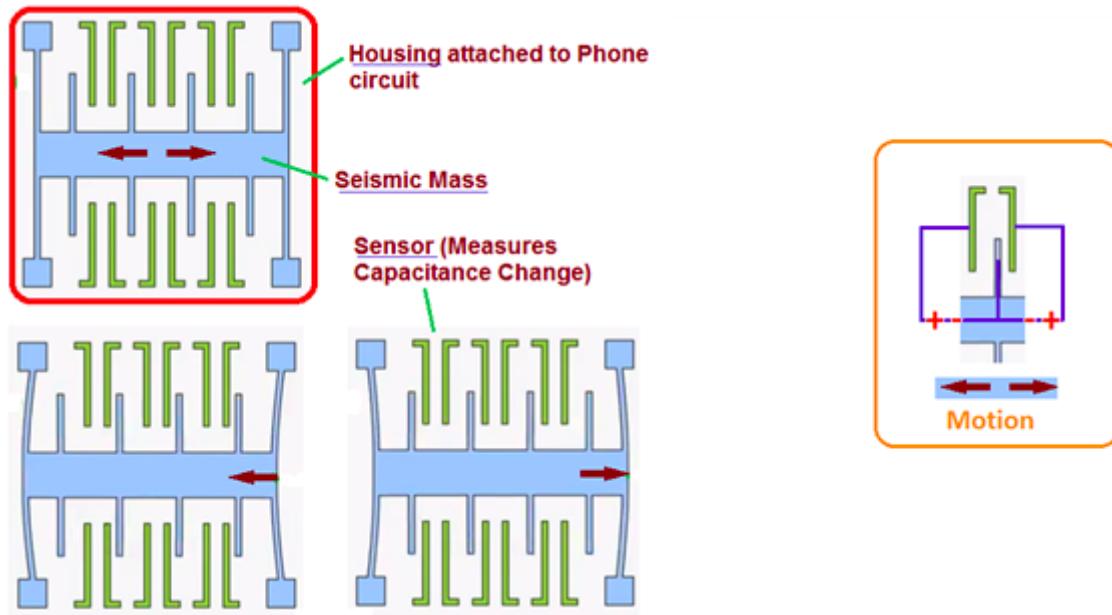


Figura 3.6. Esquema del funcionamiento de un acelerómetro capacitivo.

Su funcionamiento se basa en los **cambios en la capacitancia** de los elementos del mismo. La pieza azul de la Figura 3.6 desempeña la misma labor que la masa en el extremo del muelle, y su elasticidad permite que se pueda balancear con el movimiento del teléfono móvil.

Así, la **distancia existente entre la patilla de la masa y las dos placas del medidor de capacitancia** hace que cambie este parámetro, interpretándolo el dispositivo para poder determinar la aceleración del teléfono.

	a (m/s ²)	a (g)
Eje X	0.153	0.0156
Eje Y	0.0	0.0
Eje Z	9.807	1.00004
Módulo	9,80819	1.0001

Tabla 1. Valores devueltos por el acelerómetro en reposo sobre una mesa.

La Tabla 1 muestra los valores devueltos por el acelerómetro estando el smartphone en horizontal sobre una mesa. En esa situación, el esquema en lo que respecta a las fuerzas que influyen sobre este sensor se puede ver en la Figura 3.7.

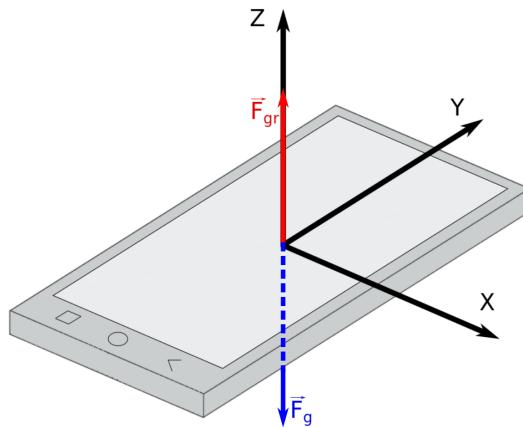


Figura 3.7. Fuerzas que actúan sobre el móvil en reposo sobre una superficie plana.

Según la **Tercera Ley de Newton**, o Principio de Acción-Reacción, si un objeto A ejerce una fuerza sobre un objeto B, B ejerce una fuerza sobre A con igual módulo y dirección, pero de sentido contrario. Por lo tanto, y teniendo en cuenta que el móvil está sobre una mesa y siendo sometido a la fuerza de la gravedad, existe una fuerza de igual módulo en sentido contrario con la dirección del eje Z: 9.8 m/s^2 , o **1g**, la aceleración de la gravedad. Esta fuerza normal **coincide en sentido con el semieje positivo de las Z**, motivo por el que el valor devuelto por el acelerómetro es mayor que 0; situar el móvil boca abajo implicaría la cifra opuesta.

En este caso, toda la fuerza a la que está sometida el dispositivo coincide con un sólo eje; sin embargo, inclinar el móvil en distintos ángulos tiene repercusión en los valores devueltos por el sensor (prueba de ello es la gran cantidad de juegos que aprovechan su funcionalidad). Por ejemplo, el smartphone podría estar inclinado con un determinado ángulo con respecto a la superficie de la mesa.

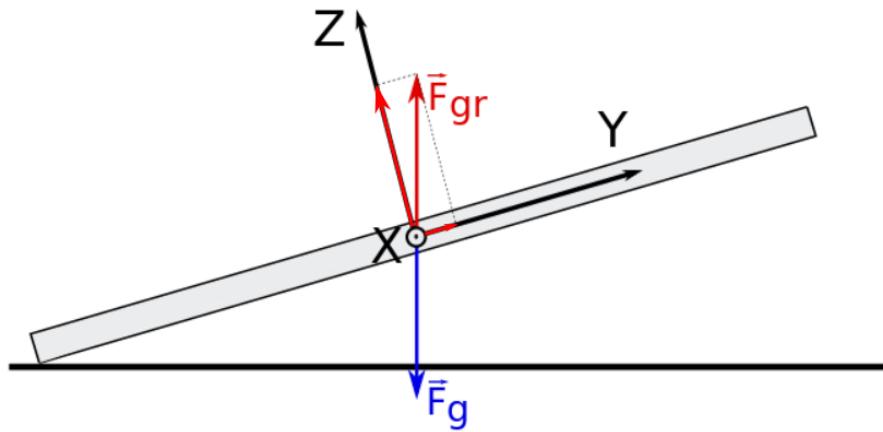


Figura 3.8. Fuerzas que actúan sobre el móvil inclinado en una superficie plana.

En este caso, la **fuerza reactiva se descompone** en una componente en el eje Y y otra en el eje Z, devolviendo datos como los que se pueden observar en la Tabla 2.

De esta forma, queda patente el efecto de la fuerza de la gravedad sobre el smartphone. Esta es una **fuerza estática**, que actúa siempre con el mismo módulo, dirección y sentido; sin embargo, el móvil puede verse afectado por otras fuerzas, como algún

	a (m/s^2)	a (g)
Eje X	0.153	0.0156
Eje Y	2.2298	0.22738
Eje Z	9.5	0.96873
Módulo	9.75938	0.99518

Tabla 2. Valores devueltos por el acelerómetro inclinado sobre una mesa.

objeto golpeándolo o una caída. Como se ha explicado anteriormente, el sensor es capaz de determinar cómo una fuerza actúa sobre él, para devolver el valor de la aceleración teniendo en cuenta la gravedad.

Como referencia, en la Tabla 3 se puede ver a cuántas g estamos expuestos al realizar algunas **acciones cotidianas**. Cabe destacar que los valores mostrados se mantienen durante fracciones de segundo, por lo que esa aceleración no repercute de forma negativa en nuestro organismo.

	a (g)
Gravedad de la Tierra	1
Bache en la carretera	2
Estornudo	2.9
Tos	3.5

Tabla 3. Aceleración sufrida ante determinados eventos cotidianos.

3.3. APLICACIONES WEB, HÍBRIDAS Y NATIVAS

Actualmente, existe una **división** en lo que respecta al **desarrollo de aplicaciones móviles**. Por un lado, se encuentran las **nativas** (para Android, en este TFG), escritas con código propio de la plataforma -Java- y que requieren la instalación en el dispositivo; por otro, se encuentran las **aplicaciones web**, que se ejecutan sobre el navegador del smartphone y no requieren instalación; y por último las **híbridas**, que se encuentran a medio camino entre ambas alternativas.

A día de hoy, se requiere una **conexión a Internet** para desempeñar muchas acciones dentro de lo que permite una aplicación, desde descargarla del Google Play Store hasta enviar y recibir datos para refrescar cierto contenido. Además, el peso de las aplicaciones fácilmente supera los 20 MB, lo que puede suponer un serio inconveniente en smartphones con una memoria interna limitada. Estos argumentos son los más comentados a la hora de justificar la existencia y la utilidad de las aplicaciones web, o *web apps*.

Este tipo de software **no se instalan en el dispositivo**, sino que el **propio navegador es el encargado de procesar la información recibida de un servidor web**. De esta forma, teniendo en cuenta que muchas aplicaciones requieren una conexión de datos de por sí, se lleva este hecho al extremo de **tener la propia aplicación en Internet**. La idea es poder tener **una página web que parezca una aplicación**, con la que el usuario pueda interactuar y consumir su contenido de forma satisfactoria.

ria simplemente accediendo a una URL. Esta idea la avala, por ejemplo, Dominique Hazaël-Massieux, ingeniero del W3C.

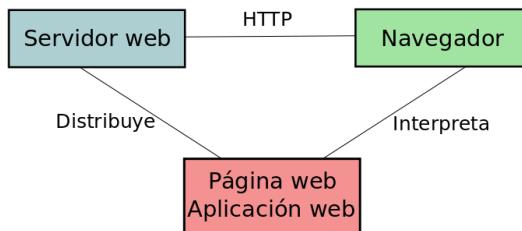


Figura 3.9.- Arquitectura web, teniendo en cuenta tanto páginas como aplicaciones web.

La fragmentación se ve, de este modo, reducida de forma drástica o incluso desaparece completamente: este sistema responde a un modelo centralizado, donde **la aplicación está en un sólo lugar** (la web) en vez de en todos los dispositivos de las personas que la utilicen. Se consigue así que las actualizaciones sean inmediatas, siendo necesario modificar únicamente el contenido al que posteriormente se accederá desde el navegador.

Aunque existen alternativas en lo que respecta al lenguaje de programación a utilizar -como PHP o Python-, en la mayoría de los casos se hace uso de **HTML5, CSS y JavaScript**. Esto, además, posibilita la **compatibilidad con todos los sistemas operativos**, ya que no es necesario tener conocimientos de los lenguajes propios de cada uno de ellos.

Sin embargo, con las aplicaciones web siguen existiendo **limitaciones** que pueden ser consideradas decisivas en comparación con las **nativas**. Estas últimas **aprovechan al máximo el hardware del dispositivo**, siendo posible acceder a todas las funcionalidades del mismo -al contrario que las *web apps*- y consiguiendo, por norma general, un mejor redimimiento. La distribución, en lugar de ser centralizada, se realiza por medio -en este caso- del Google Play Store, descargando e instalando las aplicaciones de la forma convencional.

Entre estos dos modelos se encuentra una **alternativa intermedia**, que intenta reunir las ventajas de cada uno de ellos: las **aplicaciones híbridas, como la desarrollada en este TFG**.

Estas se escriben en **HTML5**, al igual que las web, y **se pueden combinar con código nativo**. Se encuentran en la tienda de Google y también **se ejecutan sobre el navegador, con la particularidad de que este se encuentra incrustado en la propia aplicación**. Pueden tener acceso a prácticamente todo el hardware del dispositivo, como el GPS, la cámara o el acelerómetro.

En la siguiente gráfica se muestra la relación entre el acceso a las funciones del smartphone y el modelo utilizado.

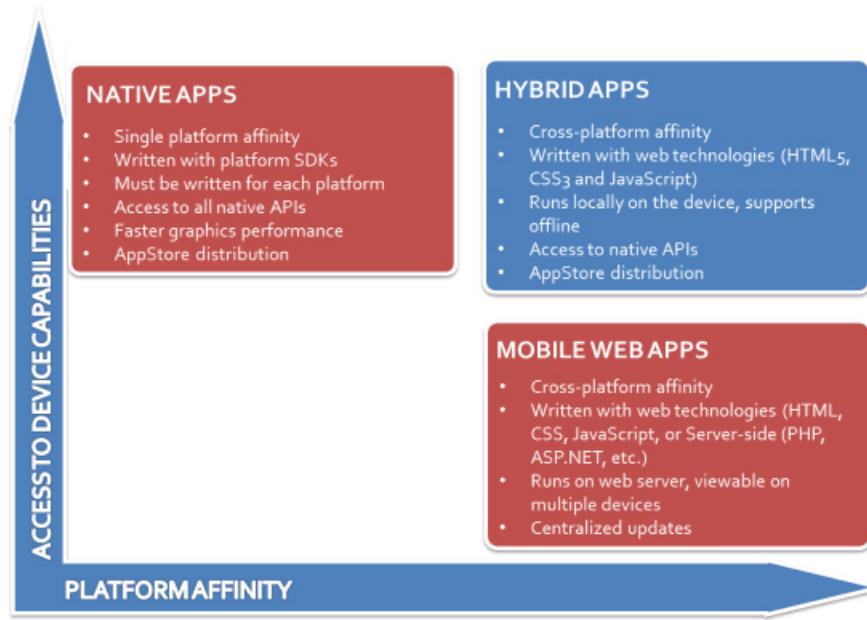


Figura 3.10.- Relación entre el modelo utilizado y el acceso al hardware del dispositivo.

Por su parte, la Tabla 4 resume de las características de cada forma de desarrollo.

	Aplicaciones nativas	Aplicaciones híbridas	Aplicaciones web
Lenguaje	Java	HTML5, CSS, JavaScript, Java	HTML5, CSS, JavaScript
Acceso al hardware	Total	Muy completo	Limitado
Inversión inicial	Elevada	Intermedia	Baja
Compatibilidad	Baja	Intermedia	Alta
Distribución	Google Play Store	Google Play Store	Centralizada
Peso de la aplicación	Elevado	Intermedio	No ocupan espacio en el dispositivo
Actualizaciones	Por medio del Play Store	Por medio del Play Store	En el servidor que aloja la aplicación web
Funcionamiento offline	Adecuado	Adecuado	Limitado
Rendimiento	Óptimo	Peor que nativo	Peor que nativo

Tabla 4. Resumen de las características de los distintos modelos

Lo explicado en esta sección sirve como base para entender el fundamento de la herramienta utilizada para el desarrollo de gran parte de este TFG: **Apache Cordova**.

3.4. APACHE CORDOVA

Aprovechando el auge de las tecnologías web de los últimos años, en 2009 la empresa Nitobi creó un framework para el desarrollo de aplicaciones móviles utilizando tecnologías web. Tras donar el proyecto a Apache en 2011, fue comprada por Adobe, provocando esto la coexistencia actual de dos proyectos con prácticamente las mismas

funciones: PhoneGap y Apache Cordova, siendo esta última la plataforma utilizada en este Trabajo Fin de Grado.



Figura 3.11. Evolución temporal del framework PhoneGap.

Cordova reúne un conjunto de APIs que permiten **hacer uso de las funciones nativas del dispositivo**, como el acelerómetro o el GPS. Mientras que este acceso al hardware se consigue haciendo uso de JavaScript, el resto de la aplicación se escribe combinando este lenguaje con HTML5 y CSS3, tal y como se haría en una página web de escritorio. De esta forma, se consigue que esta sea **multiplataforma** sin necesidad de programar código en los lenguajes nativos; Cordova actúa como un contenedor, **empaquetando el desarrollo web y generando un archivo .apk** (extensión de las aplicaciones Android) que se puede instalar en el smartphone.

La idea de este framework es aprovechar los conocimientos web de los programadores, **minimizando el coste y el esfuerzo** que implica desarrollar aplicaciones para todos los sistemas operativos. En la Tabla 5, disponible en [su página web](#), se puede ver a qué funciones nativas se puede acceder para diversos sistemas operativos móviles del mercado. En ella se puede comprobar la **alta compatibilidad** con las distintas alternativas existentes a día de hoy, en contraposición a los desarrollos orientados a una sola plataforma.

	Amazon FireOS	An-droid	Black-berry 10	Fire-fox OS	iOS	Ubuntu Phone	Windows Phone 8
Aceleróme-tro	✓	✓	✓	✓	✓	✓	✓
Estado de la batería	✓	✓	✓	✓	✓	✗	✓
Cámara	✓	✓	✓	✓	✓	✓	✓
Capturas de pantalla	✓	✓	✓	✗	✓	✓	✓
Brújula	✓	✓	✓	✗	✓	✓	✓
Conexión de datos	✓	✓	✓	✗	✓	✓	✓
Contactos	✓	✓	✓	✓	✓	✓	✓
Geolocaliza-ción	✓	✓	✓	✓	✓	✓	✓
Notificacio-nes	✓	✓	✓	✗	✓	✓	✓
Splashscreen	✓	✓	✓	✗	✓	✓	✓
Vibración	✓	✓	✓	✓	✓	✗	✓

Tabla 5. APIs de Cordova para los distintos sistemas operativos móviles.

Sin embargo, **en este TFG Cordova cuenta con una limitación insalvable con código web**. La idea es crear un servicio de monitorización en segundo plano, complementado con una interfaz con la que el usuario pueda interactuar. En lo que respecta a esta última opción, este framework cumple su cometido sin problemas; no obstante, **la plataforma de Apache permite programar aplicaciones que se vayan a ejecutar en primer plano**. No está pensada para el desarrollo de lo que en Android se conoce como **servicios**.

3.5. PROGRAMACIÓN Y SERVICIOS EN ANDROID

Antes de explicar qué son los servicios, es conveniente hacer un **breve repaso por algunos conceptos fundamentales** en el desarrollo de aplicaciones Android.

Una aplicación Android está formada por distintos componentes, que en su conjunto forman toda la experiencia que resulta de su uso. Estos componentes tienen que estar estrechamente relacionados para que el funcionamiento sea correcto, y si bien la idea de esta sección no es hacer un tutorial de programación Android, se hará un repaso a algunos conceptos básicos que se utilizarán a lo largo del TFG.

3.5.1. ACTIVIDADES

Según la propia definición de [la página oficial](#), una actividad es un componente de una aplicación que proporciona una **pantalla con la que el usuario puede interactuar**. Coloquialmente, se puede entender como lo que este está viendo en un determinado momento.

A no ser que la aplicación sea muy sencilla, normalmente consta de una serie de actividades enlazadas las unas con las otras y que pueden intercambiar información. A lo largo de todo el proceso, **las actividades atraviesan una serie de estados** en lo que se conoce como su **ciclo de vida**.

- **Resumed:** también conocido como “corriendo” o “running”, es el estado en el que se encuentra la actividad cuando **es visible para el usuario**.
- **Paused:** cuando una actividad está en este estado, significa que otra es visible sobre la primera, pero no la cubre completamente. Puede ser cerrada por el sistema en caso de extrema necesidad de memoria.
- **Stopped:** actividad completamente cubierta por otra, por lo que pasa a estar **en segundo plano**. Puede ser cerrada por el sistema si es necesario.

Una actividad, en el ámbito de la programación, no es más que una clase que hereda de la clase *Activity* de Android. De esta forma, cuenta con sus propios métodos que posibilitan la transición entre los estados comentados anteriormente. La Figura 3.12 ilustra este comportamiento de manera precisa, ya que es algo a tener en cuenta al desarrollar aplicaciones con las que interactúen los usuarios.

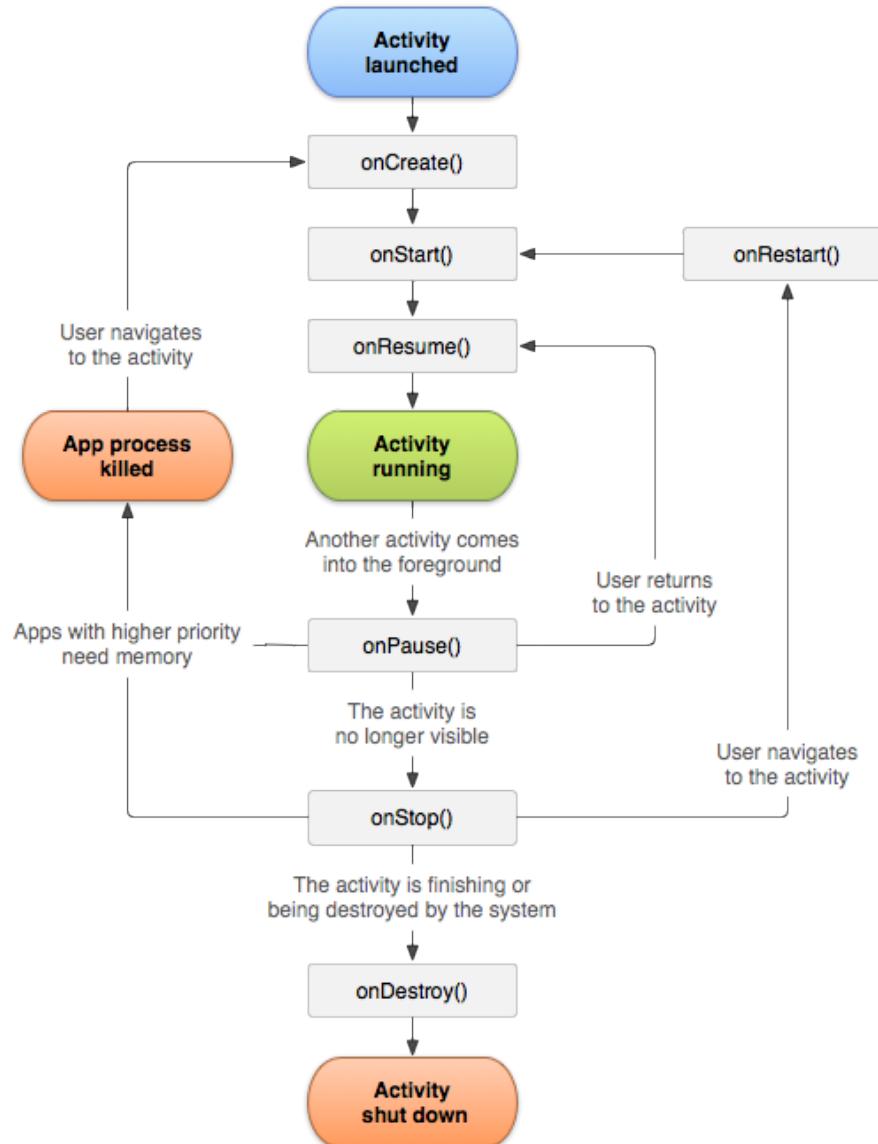


Figura 3.12. Ciclo de vida de una actividad.

3.5.2. INTENTS

Un intent es el **método de comunicación entre componentes de una aplicación**. Según se describe en [la página oficial](#), es “una descripción abstracta de una operación a ser ejecutada”. De esta forma, y aglutinando la información de ambas definiciones, un intent contiene información a transmitir entre componentes, pudiendo esta ser el **punto de partida para lanzar actividades o servicios**.

3.5.3. SERVICIOS

Android proporciona varios **mecanismos para la ejecución de tareas en segundo plano**. Estas tareas se ejecutan sin que el usuario perciba que lo hacen, ya que de por sí no tienen asociada una interfaz que se muestre en la pantalla del smartphone.

El abanico de opciones posibles es muy amplio, desde reproducir música hasta realizar conexiones a internet.

Al igual que las actividades, estos componentes son una clase que hereda de otra clase de Android, “Service”. También tienen un **ciclo de vida** que los define, consistente, de igual manera, en una serie de métodos que desencadenan un cambio de estado. La Figura 3.13 muestra este comportamiento.

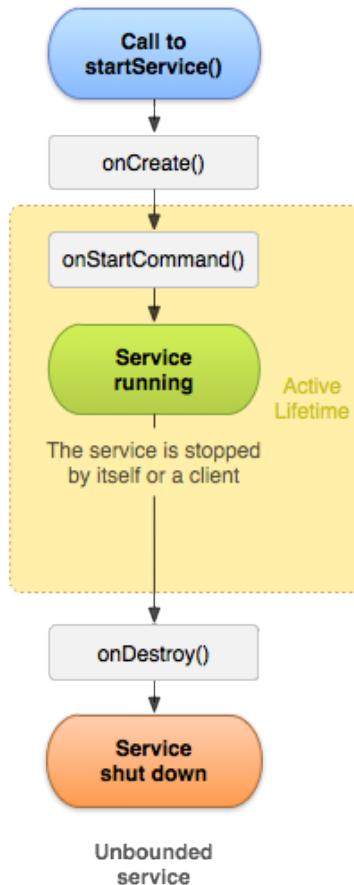


Figura 3.13. Ciclo de vida de un servicio.

Enlazando con la sección de Apache Cordova, **un servicio es lo que se utiliza para implementar el servicio de monitorización**: es posible ejecutar un algoritmo de detección de caídas en segundo plano, de forma que tome medidas automáticas en caso de que determine un incidente de ese tipo.

Por lo tanto, dos etapas bien diferenciadas en el desarrollo de este TFG son las siguientes:

- **Servicio de monitorización de caídas**: programado utilizando código nativo, en base a las características [explicadas con anterioridad](#).
- **Interfaz**: programada en HTML5, CSS3 y JavaScript, utilizando la plataforma [Apache Cordova](#).

3.6. CORDOVA CLI E IDES

Apache Cordova proporciona **dos métodos de trabajo** de cara al desarrollo de aplicaciones. La principal diferencia consiste en la finalidad del mismo:

- **Command-Line Interface (CLI)**: consiste en una interfaz que permite **ejecutar acciones por medio de comandos**, como crear un nuevo proyecto o añadir nuevos sistemas operativos objetivo al mismo. Está **orientado a desarrollos multiplataforma**, proporcionando una abstracción que permite trabajar a un nivel lo suficientemente alto como para no tener que preocuparse de las particularidades de cada sistema. Posibilita, por ejemplo, compilar proyectos para varias plataformas a la vez, así como añadir plugins del mismo modo.
- **Centrado en una plataforma**: es el método de trabajo recomendado cuando se quieren **combinar componentes web con código y funciones nativas**. La idea en este caso es profundizar a más bajo nivel, utilizando el SDK (Software Development Kit) y un IDE.

La **segunda opción, por tanto, es la más recomendable para este TFG**, teniendo en cuenta que es necesario la implementación de un servicio que trabaje en segundo plano.

La elección del IDE, por su parte, es importante de cara a la comodidad en el desarrollo. Por tanto, se han tenido en cuenta distintas alternativas de las muchas disponibles.

3.6.1. ECLIPSE

Eclipse era el entorno de desarrollo recomendado por Google hasta que presentó Android Studio en el I/O del 2013. Aunque es posible desarrollar con él en multitud de lenguajes, es muy conocido por todo lo relacionado con Java, por lo que fue un candidato ideal teniendo en cuenta el sistema operativo móvil objetivo.



Figura 3.14. Logo de Eclipse.

No obstante, el soporte para **la programación web también es importante**, dado que la interfaz de FallApp se conseguirá usando HTML5, CSS3 y JavaScript. Eclipse dispone de un plugin, Aptana, que añade las funcionalidades necesarias para este nuevo requisito.

3.6.2. ANDROID STUDIO

Android Studio es actualmente el IDE recomendado por Google para el desarrollo de aplicaciones Android. Está basado en IntelliJ IDEA, añadiendo características que facilitan la programación en esta plataforma.



Figura 3.15. Logo de Android Studio.

Esta es, a día de hoy, la opción ideal si el sistema operativo objetivo es Android; sin embargo, dado que también se hará uso de lenguajes web, y que **no se encontró el plugin Aptana** para complementar esta alternativa, **no se consideró la mejor opción**.

3.6.3. WEBSTORM

Si bien los IDEs anteriores tenían a Java como foco principal, Webstorm está **enfocado a los lenguajes web**. De esta forma, resulta muy adecuado para el desarrollo de la interfaz.



Figura 3.16. Logo de Webstorm.

Sin embargo, al igual que los anteriores tiene una carencia, siendo en este caso la **no interpretación de Java**. Llegados a este punto, se intentó buscar una alternativa que tuviera soporte para todos los lenguajes utilizados, encontrando la opción descrita a continuación.

3.6.4. INTELLIJ IDEA

IntelliJ IDEA es un IDE desarrollado por JetBrains. Mientras que su lenguaje objetivo principal es Java, proporciona multitud de herramientas para programar, por ejemplo, en HTML5, CSS3 y JavaScript.



Figura 3.17. Logo de IntelliJ IDEA.

Teniendo en cuenta que **reúne todas las características necesarias en este TFG**, este IDE fue el escogido para el desarrollo de FallApp. Por su parecido a Android Studio, además, fue considerado mejor alternativa que Eclipse.

3.7. GIT

Git es un software de control de versiones ampliamente utilizado en el mundo de la informática. Fue creado por Linus Torvalds en 2005, y desde entonces ha experimentado un rápido crecimiento, albergando multitud de desarrollos y proyectos.

Permite tener distintas versiones de un programa, siendo posible deshacer o rehacer cambios. Además, resulta de mucha utilidad en cosas donde existe más de un desarrollador, ya que cada contribución queda asociada a cada uno de ellos, posibilitando una mayor organización en desarrollos muy extensos.

Git es un sistema descentralizado, ya que no necesita un servidor central para funcionar; sin embargo, hay numerosos casos en los que conviene tener el código en un lugar accesible de forma permanente, punto en el que cobra importancia GitHub. GitHub es un servicio de alojamiento de repositorios de código que utiliza Git, añadiendo a este último el concepto de servidor central.

Aunque aprovechar todo el potencial que ofrecen estas herramientas sólo es posible en proyectos de gran tamaño y con varios contribuyentes, no dejan de ser muy buenas alternativas para tener en todo momento una copia de seguridad del trabajo realizado, así como para poder controlar los cambios hechos de una forma muy precisa. Por tanto, dadas sus ventajas, se han utilizado en este TFG.

3.8. ANÁLISIS DE DISPOSITIVOS Y APLICACIONES EXISTENTES

La detección de caídas es un campo que se lleva estudiando años, tanto en proyectos comerciales como Fin de Carrera. Tiene mucha aplicación en geriátricos y residencias de personas de tercera edad, ya que estos son individuos, como se ha comentado anteriormente, proclives a las caídas debido a las [causas explicadas](#).

Por lo tanto, existen diversos dispositivos que son capaces de monitorizar y detectar esta clase de incidente, algunos de los cuales se comentan a continuación.

3.8.1. DETECTOR DE CAÍDAS SENSE4CARE

Es un dispositivo de bordes redondeados y forma de piedra, con un peso de 52 gramos. Se coloca en un cinturón de neopreno, llevándose este alrededor de la cintura.



Figura 3.18. Detector de caídas SENSE4CARE.

La detección de caídas se realiza por medio de un **acelerómetro triaxial y de un algoritmo no especificado**. Esto se complementa con Bluetooth y **ZigBee**, dos tipos de comunicación inalámbrica que permiten enviar alarmas cuando se detecta el incidente.

Cuenta con algunas características orientadas a las personas mayores, como un **sistema de carga imantado** (más simple de utilizar que enchufar un cable) y **un botón de pánico**, cuya idea es ser pulsado en situaciones de necesidad de ayuda inmediata.

Otras características destacables citadas en su [ficha técnica](#) son:

- Bajo número de falsos positivos.
- Interfaz de usuario sencilla con luz y sonido.
- Se suministra con cinturón de neopreno.
- Configurable a través de la conexión Bluetooth.

3.8.2. VIGI'FALL

Vigi'Fall es un dispositivo de detección de caídas creado por un consorcio de empresas tecnológicas llamado FallWatch. Consiste en **un parche** que se coloca en un lateral del tórax del anciano, **un conjunto de sensores dispuestos en las paredes** de la casa y **una caja de control central**.



Figura 3.19. Detector de caídas Vigi'Fall.

El parche monitoriza en todo momento al paciente, que detecta una caída por medio de sensores (no se indica cuáles). Este sistema se complementa con **infrarrojos** situados en las paredes, de forma que puedan detectar ausencia de movimiento y, por tanto, enriquecer la información proporcionada por el biosensor con el fin de evitar falsos positivos. El dispositivo portátil cuenta con una **batería recargable** y puede usarse con normalidad en cualquier situación cotidiana, como ducharse.

En caso de ser necesario, se le envía un **aviso a la caja de control central**, que a su vez lo redirige a una enfermera o centralita.

Su precio varía en función de la modalidad contratada, siendo la más económica:

- **Paquete Vigi'Fall:** 1 biosensor, 1 caja de control central, 2 sensores de movimiento, 3 años de pilas y parches, soporte telefónico gratuito y garantía de 2 años.
- **Alertas:** al centro de llamadas Europ Assistance 24/3 durante 3 años.

Su **precio** es de 390 euros iniciales, con una cuota de 49 euros al mes.

3.8.3. SISTEMA DE VIDEOVIGILANCIA DE LA SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

Este sistema se sale de lo convencional y **no utiliza sensores como el acelerómetro** para la detección de caídas. En su lugar, hace uso de un **sistema de cámaras de vídeo** que cubren un determinado área de la habitación, una visión artifical capaz de determinar cuándo se produce una caída.

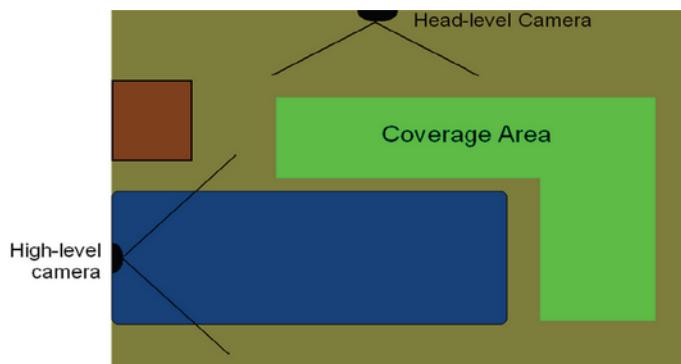


Figura 3.20. Superficie monitorizada por el sistema de cámaras.

Las cámaras se pueden situar en distintos lugares en función de la **comodidad del paciente**, ya que **puede sentir violada su privacidad**.

El **precio de este conjunto** es de 440 euros por habitación y por persona.

3.8.4. GEA

Es un dispositivo de alarma portable que comparte similitudes con la apuesta de SENSE4CARE. Más toscos y menos ergonómicos, también se **sitúa en la cintura**. La empresa detrás de este artílugo es Tecnalia.



Figura 3.21. Dispositivo GEA.

Algunas características técnicas destacadas por la propia compañía son:

- **Comunicaciones:** incorpora un módulo GSM/GPRS, por lo que es posible **contestar llamadas y usarlo de manos libres**. También puede enviar mensajes de alarma y permite la actualización remota de firmware.
- **Posicionamiento:** dispone de un **módulo GPS** para posicionamiento.
- **Alarmas:** detección automática de caídas y de si el dispositivo se lleva o no puesto, nivel bajo de batería, etcétera.
- **Batería:** autonomía de más de 24 horas y carga en menos de 2.

En la Tabla 6 se puede ver una comparativa de las especificaciones de los distintos dispositivos brevemente analizados.

	SENSE4CARE	Vigi'Fall	Sistema videovigilancia	GEA
Sistema centralizado	Sí (routers ZigBee)	Sí (caja de control central)	Sí (videovigilancia)	No
Dispositivo portable	Sí (cinturón de neopreno incluido)	Sí	No	Sí
Alerta por SMS	No	No	No	Sí
GPS	No	No	No	Sí
Botón del pánico	Sí	No	No	Sí
Conexiones inalámbricas	Bluetooth y ZigBee	Infrrojos (en las paredes)	No	GSM y GPRS
Batería	Sí	Sí	No	Sí
Precio	-	390€+ 49€/mes	440€/habitación	-

Tabla 6. Comparativa de características básicas de los dispositivos analizados.

Además de estos dispositivos, existen otros muchos con funciones bastante similares, variando la forma, el precio, el público objetivo y el algoritmo (nunca especificado). También se encuentran en la red multitud de tesis y Proyectos Fin de Carrera que tratan este tema.

En ellos, se utilizan distintos modelos de acelerómetros y placas microcontroladoras que, sujetas al cuerpo del paciente, pueden detectar caídas como lo hacen los dispositivos analizados. Asimismo, suelen incluir sistemas de alertas, normalmente conectados a un sistema de alarmas central situado en geriátricos o residencias de ancianos.

En este TFG no se explica ninguno de ellos por los siguientes motivos:

- El funcionamiento es muy similar al explicado en los puntos anteriores.
- No son proyectos comerciales, sino que sólo figuran en los pdf encontrados en Internet.
- Teniendo en cuenta que se basan en placas electrónicas, en muchas ocasiones gran parte del proyecto consiste en analizar y detallar las especificaciones y requerimientos que esta debe cumplir.

Las alternativas analizadas cuentan con **muchas características** y con un **porcentaje de falsos positivos muy reducido**, estando normalmente comprendido entre un 0% y un 10%; sin embargo, casi todos ellos (menos el sistema de videovigilancia, bastante distinto al resto) cuentan con la **desventaja de tener que usar un dispositivo externo**, normalmente situado en la cintura. De esta forma, ya **no son tan accesibles al gran público**, ya que muchos son caros y están orientados a geriátricos y residencias. FallApp intenta aunar la **capacidad de detectar caídas y su consecuente utilidad para las personas de tercera edad, y la extensión y la versatilidad de los smartphones**. No todo el mundo puede, quiere o conoce estas alternativas, mientras que es mucho más fácil que una persona pueda descargar del Play Store una simple aplicación. Esto da paso al siguiente capítulo de este TFG, aunque primero se repasarán algunas alternativas de software móvil disponibles para Android.

3.8.5. ISCREAM

iScream es una aplicación consistente en una **interfaz muy simple y un servicio que corre en segundo plano**. Su única función es hacer sonar el móvil en caso de que se caiga.



iScream
when I fall



Figura 3.22. Pantalla principal de la aplicación iScream.

El propio desarrollador indica en el Play Store que iScream está **hecha por diversión** y lleva sin actualizarse más de un año, por lo que no tiene fines comerciales; sin embargo, puede resultar interesante comprobar la fiabilidad a la hora de detectar una caída del móvil, ya que es **uno de los eventos que FallApp debería diferenciar** de cara a evitar falsos positivos.

Las pruebas realizadas no han sido muy satisfactorias, si bien es cierto que se ha dejado caer el móvil siempre sobre una superficie blanda. Aún así, no está de más repasar las alternativas que la tienda de Google ofrece en lo referido a las caídas.

3.8.6. FADE: FALL DETECTOR

Fall detector constituyó, inicialmente, una desilusión importante en el tránsito de este TFG. Fue una aplicación que se encontró cuando se buscaba información con la que respaldar y comparar FallApp, una vez iniciado ya su desarrollo.

Fade realiza la misma tarea: **detectar caídas y enviar alertas de forma automática**. Permite regular distintos parámetros:

- **Métodos de contacto:** permite configurar **números de teléfono** (para llamadas y SMS) y **correos electrónicos**. Con respecto a estos últimos, en las pruebas realizadas no ha llegado ninguno.
- **Sensibilidad:** permite escoger entre alta, media o baja.
- **Tono de alerta:** melodía que sonará para avisar de la detección de una caída.
- **Tiempo de aviso:** tiempo que transcurre antes de enviar una alerta.

Cuenta con una interfaz futurista con bastantes opciones. Por su aspecto y rendimiento, resulta evidente que fue elaborada utilizando **código nativo en lugar de lenguajes web**.



Figura 3.23. Interfaz de la aplicación Fade: Fall Detector.

Aunque en un principio FallApp parecía inútil comparada con Fade, existen **dos factores clave que hacen que el desarrollo sea factible**:

- **Aplicación no actualizada:** Fade lleva sin ser actualizada aproximadamente **2 años**, lo que da a entender que el **proyecto está descontinuado**.
- **Detección de caídas imprecisa:** en las pruebas realizadas, **con un ligero impacto el móvil detectaba una caída**. Lo que ocurriría a continuación (constancia de datos, móvil en horizontal...) parecía no tener importancia, lo que derivaba en la posibilidad de **falsos positivos**. Es posible que lo que haga Fade sea detectar un impacto por medio del acelerómetro, **sin entrar a valorar qué ha ocurrido antes o que ocurre después de ese instante**.

Por lo tanto, dadas estas circunstancias, se consideró una buena opción **buscar un algoritmo que permitiera detectar caídas con una mayor precisión**.

4. Desarrollo de la aplicación

En este capítulo se detalla el **desarrollo de la aplicación para Android**. Se comenzará por las pautas seguidas en lo que respecta a la interfaz web elaborada con Apache Cordova, continuando con el núcleo en Java. Para acabar, se estudiará cómo enlazar ambas partes, haciendo que una aplicación híbrida para Android combine las funcionalidades proporcionadas por las dos plataformas.

4.1. REQUISITOS DE UNA APLICACIÓN ANDROID

Google proporciona la posibilidad de programar una aplicación Android en cualquier sistema operativo de escritorio, ya sea Windows, una distribución de Linux o MacOS. En este TFG se han utilizado los dos primeros, alternando el desarrollo en Windows, Linux Mint y Elementary OS.

La preparación del entorno de desarrollo consiste en la instalación de los siguientes elementos:

- **Máquina virtual Java**: permite ejecutar código Java en el equipo. Dado que las aplicaciones Android se escriben con este lenguaje, **este paso es fundamental** para poder comenzar el desarrollo.
- **Android SDK**: el SDK (Software Development Kit) es el **conjunto de funcionalidades y herramientas necesarias** que, proporcionadas por Google, permiten programar aplicaciones para su sistema operativo.
- **IDE**: el recomendado por la multinacional es Android Studio, pero por los motivos expuestos en la sección **CORDOVA CLI E IDES**, se usará IntelliJ IDEA.

4.2. INSTALACIÓN DE CORDOVA

En la propia [página web](#) de Apache Cordova existe una guía con los pasos a seguir para instalar esta plataforma en el ordenador, y poder empezar a desarrollar aplicaciones.

Cordova se distribuye como un paquete NPM, instalable por medio de NodeJS. Este es un framework que permite implementar operaciones de entrada y salida, muy utilizado desde que surge en el año 2009. Está construido encima del motor de JavaScript V8, con el que funciona JavaScript en Google Chrome.

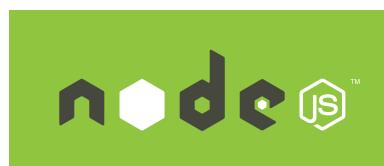


Figura 4.24. Logotipo de NodeJS.

Si bien JavaScript nativo no soporta módulos, en NodeJS es el método de almacenamiento de código. *NPM*, por su parte, es el nombre del gestor de paquetes del que se hace uso. Por lo tanto, utilizándolo es posible instalar Apache Cordova por medio de línea de comandos.

A continuación, y una vez finalizado el proceso de instalación, se puede **crear un nuevo proyecto por medio del CLI**, así como ejecutar otras tareas como la inclusión de distintos sistemas operativos o de plugins.

Esta primera acción preparará todo lo necesario para poder a utilizar la programación web: creará los archivos index.html, index.css e index.js, además de todos los archivos necesarios para poder instalar la aplicación en el emulador de Android o en el smartphone, como se puede ver en la Figura 4.25.

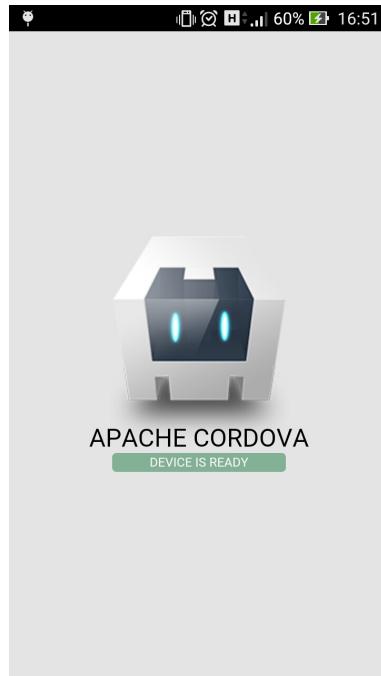


Figura 4.25. Aplicación generada automáticamente por Cordova.

Además, se podrá importar el proyecto en IntelliJ IDEA y trabajar con este IDE a partir de ese momento. La estructura, que se mantendrá a lo largo de todo el desarrollo, se puede ver en la Figura 4.26.

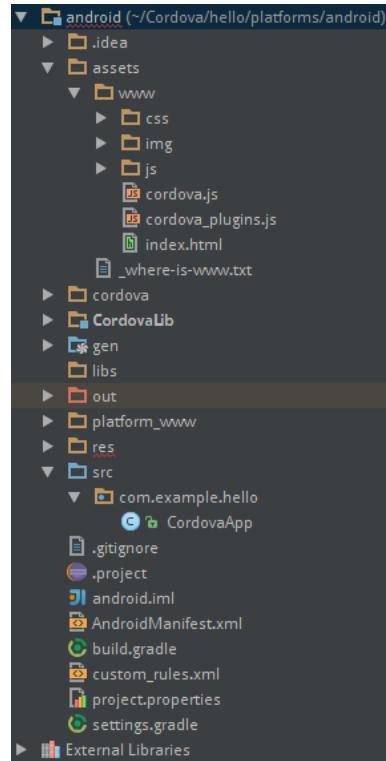


Figura 4.26. Estructura de una aplicación Android generada por Cordova.

En ella se pueden destacar, de entre todos los archivos y directorios, **tres carpetas y un fichero** tienen una influencia directa en el desarrollo de la aplicación ya que van a ser los que sufran la mayor cantidad de cambios en el mismo.

- **assets:** es el **directorio que almacena los archivos web relativos a la interfaz**. Estos ficheros responden a los principios de la programación web, siendo escritos en HTML5, CSS3 y JavaScript. Cualquier nueva pantalla (o actividad) de la aplicación debe contar con su correspondiente .html.
- **res:** contiene los **recursos usados por la aplicación**, como el icono de la misma o el archivo “config.xml” -del que se habla [más adelante](#)-.
- **src:** contiene las **clases Java que constituyen el núcleo de la aplicación Android**. En este TFG, este directorio albergará el servicio de detección de caídas y otras clases necesarias para el correcto funcionamiento de FallApp.
- **AndroidManifest.xml:** es un archivo que se genera automáticamente al crear un proyecto. Contiene **características básicas de la aplicación**, como las actividades de las que está formada, los servicios que utiliza o los permisos que necesita.

El archivo “config.xml”, bajo el directorio *app/res/xml*, contiene información de importancia para la aplicación, por lo que se detalla más detenidamente en el siguiente apartado.

4.2.1. CONFIG.XML

Es un archivo de configuración global que se autogenera con el nuevo proyecto. Por defecto se encuentra en el directorio raíz de la aplicación: app/config.xml. Además, al añadir una nueva plataforma de desarrollo (Android, en este caso), en su carpeta se crea una copia del mismo que se puede modificar directamente desde el IDE. Cuenta con distintos elementos, cuya descripción se expone a continuación:

- **widget**: los campos *id* y *version* de este elemento indican, respectivamente, el identificador y la versión de la aplicación.
- **name**: indica el **nombre de la aplicación**. Es lo que verán los usuarios tanto en el Google Play, como en el menú de aplicaciones una vez instalada.
- **description y author**: contiene metadatos e información de contacto.
- **content**: define la **página de inicio por defecto**.
- **access**: define los **dominios con los que la aplicación se puede comunicar**. El valor por defecto indica que se puede comunicar con cualquier servidor.

4.3. BUENAS PRÁCTICAS EN EL DESARROLLO CON CORDOVA

Antes de comenzar la explicación y la estructura de FallApp, es importante descartar lo mencionado en la sección [APLICACIONES WEB, HÍBRIDAS Y NATIVAS](#) referido al rendimiento.

Uno de los mayores inconvenientes en las aplicaciones web e híbridas, y un importante motivo por el que muchos desarrolladores se decantan por las nativas, es el **rendimiento**. Aunque no tiene por qué ser malo -más aún si se siguen una serie de buenas prácticas-, dista de ser óptimo como en estas últimas. Por lo tanto, a la hora de programar es recomendable **dejar de pensar en los ordenadores portátiles y de sobremesa** como los únicos dispositivos que van a visualizar el contenido. Estos cuentan con una potencia que, por norma general, no es comparable a la de un smartphone, por lo que son capaces de representar de forma correcta prácticamente cualquier página web; sin embargo, si se quiere conseguir un rendimiento óptimo en dispositivos móviles, hay **una serie de pautas que es recomendable seguir**.

- **Usar el diseño SPA**: el diseño SPA (Single Page Application) implica que los recursos asociados al HTML, CSS y JavaScript se cargan inicialmente, en lugar de cada vez que se haga clic en un enlace o se vaya a una nueva pantalla. De esta forma, la manipulación del DOM (Document Object Model, la estructura del documento .html) se puede realizar por medio de JavaScript. Más concretamente, **esta buena práctica consiste en tener un sólo archivo html**, haciendo sobre el mismo las modificaciones pertinentes en lugar de cargar nuevos ficheros.

Asimismo, una aplicación Cordova debe esperar a que se ejecute un evento “`deviceready`” -explicado en la sección [JavaScript, jQuery y los eventos](#)- nada más iniciar el conjunto de archivos web, por lo que usar este método también evita ejecuciones innecesarias.

- **Usar aceleración por hardware:** mientras que las aplicaciones nativas pueden exprimir al máximo la tarjeta gráfica del móvil, **las web se ejecutan en el contexto del navegador**, por lo que **la representación del contenido se realiza por software**. Esta diferencia se hace patente en las transiciones, donde en las primeras versiones de FallApp se pudo comprobar que eran lentas y a tirones, como si no hubiera potencia suficiente para mantener la tasa de FPS (a pesar de usar un smartphone de gama alta). La solución fue utilizar transiciones aceleradas por hardware, que maneja directamente la GPU para conseguir resultados muchísimo más fluidos.
- **Imitar aplicaciones nativas:** cada plataforma, ya sea Android, iOS, Windows Phone, etc, tiene su propio estilo en lo que respecta a la estética de las aplicaciones. Así, la misma aplicación en Android e iOS luce de forma distinta, adoptando las líneas propias de cada plataforma.

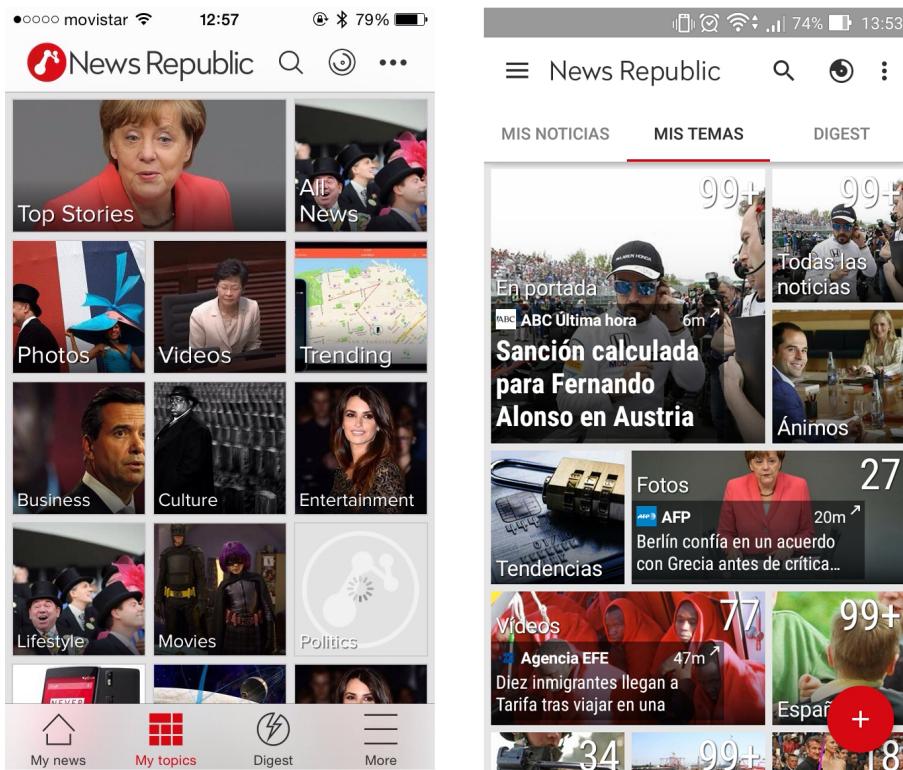


Figura 4.27.- Aplicación News Republic visualizada en iOS (izquierda) y Android (derecha).

Conseguir una buena experiencia de usuario tiene como requisito adoptar una estética acorde al sistema operativo, haciendo que la aplicación se integre perfectamente en el entorno.

- **No sobrecargar la aplicación:** aunque pueda resultar bonito, incluir muchas tipografías, sombras, bordes degradados o sombreados puede repercutir negativamente en el rendimiento. Son efectos que se usan mucho en la web, pero al ser interpretados por un dispositivo móvil pueden resultar contraproducentes.

- **Manejar de forma elegante las cargas y conexiones a Internet:** el hacer una petición para mostrar o referescar cierto contenido puede suponer que la aplicación parezca lenta o congelada, en especial si no se indica qué se está intentando. Por lo tanto, es aconsejable utilizar algún símbolo que aclare que el contenido está cargando en ese momento.
- **Usar plugins:** existen multitud de plugins para Cordova, tanto los que permiten acceder al hardware del dispositivo, como los que añaden funcionalidades. Usarlos es, en muchos casos, una buena forma de complementar un desarrollo.

Una vez explicado este aspecto, ya es posible **entender las pautas y las decisiones tomadas en el diseño de FallApp**.

4.4. ESTRUCTURA DE FALLAPP

En la Figura 4.28 se muestra la **estructura básica**, y sin entrar en detalles, de la interfaz de la aplicación FallApp.

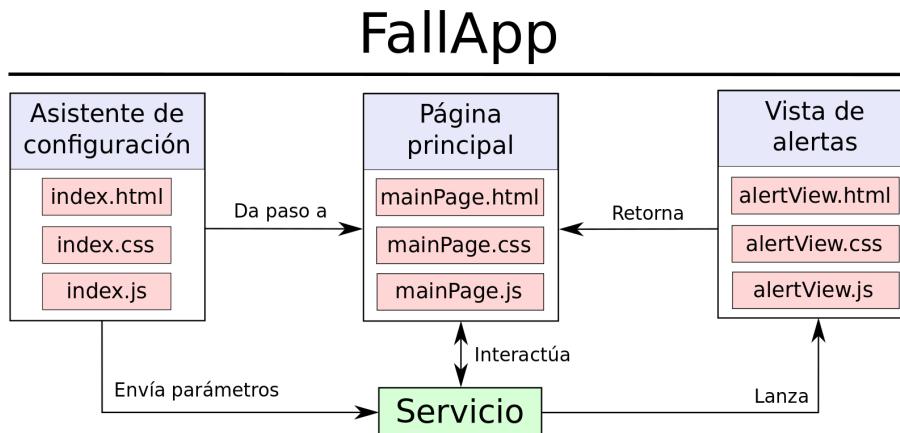


Figura 4.28. Estructura de FallApp.

Se ha optado por utilizar tres archivos .html diferenciados: uno para el asistente de configuración; otro para la página principal, la pantalla que se encontrará el usuario cuando abra la aplicación en sucesivas ocasiones; y el último para la vista de alerta. Se planteó la opción de mantener toda la funcionalidad en un solo fichero de este tipo, pero dada la total diferencia existente entre las tres partes, se consideró mejor opción lo comentado inicialmente.

- **Asistente de configuración:** una serie de pantallas donde se le explicará al usuario el funcionamiento básico. También contendrá los elementos necesarios

para que este se identifique a sí mismo, introduzca los números de contacto y especifique la sensibilidad de la aplicación.

- **Pantalla principal:** consiste en una pantalla donde se mostrarán las opciones básicas de la aplicación, así como un menú lateral muy propio de Android.
- **Vista de alertas:** página donde se le dará al usuario la opción de cancelar el envío de una alerta justo antes de realizarlo.

Teniendo en cuenta la **orientación de FallApp a personas de tercera edad**, todo lo relativo al diseño se ha intentado mantener **muy simple y sencillo**, sin elementos visuales confusos ni menús sobresaturados.

4.5. PLUGINS, AÑADIENDO FUNCIONALIDAD

Aunque Cordova proporciona el entorno y las herramientas básicas para desarrollar aplicaciones, los **plugins son scripts o módulos que permiten ampliar la funcionalidad inicial**. A grandes rasgos, se pueden dividir en tres grupos:

- **Acceso al hardware del dispositivo:** algunos plugins **proporcionan acceso al hardware del dispositivo**, como al acelerómetro o al GPS, de forma similar a como se haría utilizando código nativo. Están soportados oficialmente y se pueden instalar desde la línea de comandos (CLI).
- **Acceso al código nativo:** sirven de **puente entre los archivos web y el código nativo de las clases Java**. Normalmente son escritos por el desarrollador en función de los requisitos del proyecto.
- **Nuevas características:** permiten **aumentar la funcionalidad de la aplicación**, consistiendo normalmente en un archivo .js al que se hace referencia desde el .html. Están escritos por terceros y se pueden encontrar en la web, normalmente en repositorios de GitHub. Cualquiera puede escribir un plugin para satisfacer una determinada necesidad, por lo que el número de ellos en la red es muy elevado.

Si bien con los primeros se hicieron pruebas para comprobar cómo tomaba valores el acelerómetro, fueron los segundos y los terceros los que más importancia cobraron a lo largo del desarrollo. Dada la extensión de estos últimos, se encontraron dos con un potencial interesante:

- **FastClick:** una aplicación web elaborada con Cordova conlleva un **retardo de 300ms** entre el instante de pulsar un botón o un enlace, y la acción que desencadena. Este desfase lleva años existiendo en la red, y permite diferenciar un clic de un doble clic.

Sin embargo, teniendo en cuenta la orientación de una *web app*, esa fracción de segundo desemboca en lo que **parece un rendimiento deficiente**. Aunque no es un hecho especialmente notable o molesto, puede hacer que la aplicación parezca lenta o poco fluida, por lo que es una buena práctica **evitar que eso ocurra**.

Una vez incorporado el archivo .js, la forma de ejecutar funciones JavaScript, en lugar de con el *onclick*, es la siguiente:

```
1 <a href="javascript:menu('opcion1');">Opcion 1</a>
```

- **IScroll**: este plugin añade una funcionalidad muy interesante, permitiendo **aprovechar al máximo el espacio en una pantalla táctil**: hacer scroll dentro de capas. De esta forma, dentro de un div que ocupe, por ejemplo, media pantalla, se puede **desplazar el contenido** para visualizar texto o imágenes que en principio no cabrían en ese espacio.

La forma de conseguirlo es, una vez añadido el fichero .js, instanciar el constructor *IScroll* pasándole como parámetro el elemento en el que queremos hacer scroll.

```
1 new IScroll(".block");
```

4.6. ASISTENTE DE CONFIGURACIÓN

El asistente de configuración consiste en una serie de pantallas que el usuario irá desplazando -cumpliendo los requisitos de las mismas-, hasta lanzar la pantalla principal. En ese momento, además, se pondrá en marcha el servicio de detección de caídas.



Figura 4.29. Pantallas del asistente de configuración.

La estructura se ha elaborado teniendo en cuenta lo explicado en la sección [Buenas prácticas en el desarrollo con Cordova](#). De esta forma, y usando el diseño SPA, **se descartó la idea inicial de utilizar un archivo .html para prácticamente todo**; en cambio, existe un sólo fichero de este tipo, donde las transiciones entre las distintas pantallas consisten en el traslado del carrusel central.

```

1 <body>
2   <div id="header">
3     ...
4   </div>
5
6   <div id="slider">
7     ...
8   </div>
9 </body>

```

A la vista de este código, la cabecera se mostraría siempre en pantalla, siendo el “slider” el que se desplace a lo largo de la pantalla (de derecha a izquierda o viceversa, como exemplifica la Figura 4.30), estando el **contenido distribuido a lo largo del mismo**. Así, una vez cargado el .html, todo el contenido de la página estará ya disponible, aunque en cada momento el usuario sólo tendrá una porción del mismo disponible. Se dispone, además, de unos elementos circulares en la parte inferior, que indican en todo momento cuál es la pantalla actual.



Figura 4.30. Disposición de las pantallas en el asistente de configuración.

El contenido que se encontrará el usuario consiste en los siguientes puntos:

- **1^a página:** bienvenida a FallApp.
- **2^a página:** información, recomendaciones y modo de uso de la aplicación.
- **3^a página:** un campo para introducir su nombre.
- **4^a página:** varios campos para introducir entre 1 y 5 números de contacto.
- **5^a página:** una barra lateral para escoger la sensibilidad del servicio de detección de caídas.
- **6^a página:** finalización del asistente de configuración.

Por su parte, el desplazamiento comentado anteriormente se consigue de dos formas distintas: usando los botones “Atrás” y “Siguiente”, o moviendo el dedo por la pantalla en la dirección y el sentido correcto. Pero antes de explicar cómo se consigue, es importante destacar la importancia de JavaScript en las interacciones del usuario.

4.6.1. JAVASCRIPT, JQUERY Y LOS EVENTOS

Como se ha comentado con anterioridad, por medio de JavaScript se posibilita que el usuario **interactúe con la aplicación**, usando funciones, eventos o accediendo al hardware del dispositivo.

Aunque a lo largo de toda la memoria siempre se ha mencionado este lenguaje como método para conseguir estos fines, también es posible utilizar un framework como jQuery.

jQuery es una librería escrita en JavaScript que **simplifica su uso** en gran medida. Se puede entender como una capa de abstracción, ya que oculta la complejidad del mismo, siendo posible escribir con pocas líneas lo que originalmente conllevaría muchas más. Aunque, siguiendo las pautas disponibles en la página web oficial, inicialmente la interfaz FallApp comenzó a escribirse usando JavaScript, esta intención pronto derivó en el uso del framework. Esto es debido, principalmente, a los **métodos que proporciona para la modificación de clases de los elementos del DOM**, simplificando sobremanera este proceso; y a la **modificación de propiedades CSS**.

Por ejemplo, en [este tutorial](#) de la página de PhoneGap Spain, usan el siguiente código para añadir una clase CSS a un elemento:

```

1 function addClass( classname, element ) {
2     var cn = element.className;
3     if( cn.indexOf( classname ) != -1 ) {
4         return;
5     }
6     if( cn != '' ) {
7         classname = ' '+classname;
8     }
9     element.className = cn+classname;
10}

```

Un proceso similar se repite para eliminar una clase. Sin embargo, tanto para añadir, como para eliminar o modificar clases, el homólogo en jQuery sería:

```

1 element.addClass("newClass");
2 element.removeClass("oldClass");
3 element.removeClass("oldClass").addClass("newClass");

```

jQuery, al igual que JavaScript, proporciona una serie de eventos que desencadenan la ejecución de funciones. Uno de ellos es el **evento “deviceready”**, de especial importancia en aplicaciones elaboradas con Cordova.

Este evento es lanzado cuando las APIs de Cordova se han cargado y están listas para su acceso. Por tanto, en ese momento es seguro hacer llamadas a métodos nativos de esta plataforma. Aunque este aspecto no ha dado problemas en ninguna de las pruebas realizadas, en su [propia página](#) lo resaltan como muy significativo, ya que es potencialmente posible que la aplicación web pueda intentar ejecutar una función JavaScript de Cordova antes de que esta haya cargado.

La forma de esperar a que haya cargado toda la aplicación es la siguiente:

```

1 document.addEventListener("deviceready", onDeviceReady, false);
2
3 function onDeviceReady() {
4     // Now safe to use device APIs
5 }
```

De esta forma, a partir de la ejecución de la función “onDeviceReady()” se puede utilizar la aplicación en su totalidad sin ningún tipo de problema.

Otros eventos utilizados son los relacionados con los toques en la pantalla o con el fin de las transiciones CSS, que se explicarán más adelante.

Volviendo a la explicación de las **transiciones entre pantallas**, los dos métodos para conseguirlas se detallan a continuación.

- **Botones “Atrás” y “Siguiente”:** la transición se consigue utilizando el siguiente método CSS de jQuery.

```

1 globalElements.slider
2     .css("transform", "translate3d(" +
3         sliderFunctions.var.pixelOffset + "px, 0, 0)")
4     .addClass("animate");
```

Mediante esta instrucción, y teniendo en cuenta que *translate3d* utiliza aceleración por hardware, se consigue desplazar el elemento *slider* -comentado anteriormente- tantos píxeles como indice la variable *pixelOffset*. Se le añade, además, la clase *animate*, que posibilita que el desplazamiento se consiga de forma animada. En otras palabras, con *.css* se indica qué hay que hacer, y con la clase *animate* se indica cómo. De esta forma, se consigue una transición suave y fluida.

Además del desplazamiento, hay otras acciones que se deben ejecutar cuando el usuario pulsa los botones “Atrás” o “Siguiente”:

1. En la primera pantalla, el botón “Atrás” muestra una **opacidad menor**, indicando así que está desactivado. Lo mismo ocurre con “Siguiente” en la última. Por lo tanto, cuando el usuario pulsa alguno de ellos hay que comprobar si es necesario realizar alguna operación al respecto para **evitar desplazamientos en los extremos**, y modificar la opacidad cuando sea necesario.
2. Cambiar el **indicador activo** de la parte inferior de la pantalla, que indica en cuál de ellas se encuentra el usuario.
3. Calcular el número de píxeles a desplazar.
4. Realizar la transición explicada más arriba.

- **Eventos de toque:** JavaScript proporciona, pensando en las pantallas táctiles, unos eventos de toque con los que se pueden ejecutar determinadas acciones cuando **se inicia una pulsación, se finaliza, o se desplaza el dedo por la pantalla**. Estos son *touchstart*, *touchend* y *touchmove*.

Por medio de un *listener* sobre el elemento en el que se quieran registrar las pulsaciones, se puede configurar la aplicación para que ejecute esas acciones.

```

1 globalElements.slider
2     .on("touchstart", sliderFunctions.touchStart)
3     .on("touchmove", sliderFunctions.touchMove)
4     .on("touchend", sliderFunctions.touchEnd);

```

El método *on* de jQuery permite asociar funciones a un determinado evento, estando *touchStart*, *touchMove* y *touchEnd* definidas para responder a los touch events.

El algoritmo para conseguir el desplazamiento a medida que el usuario desplaza el dedo por la pantalla se basa en estados:

- **Estado “waitingTouch”**: todavía no se ha pulsado la pantalla, por lo que tampoco se ha lanzado el evento “touchstart”.
- **Estado “startTouch”**: se ha pulsado la pantalla, pero todavía no ha comenzado el desplazamiento por la misma.
- **Estado “moveTouch”**: se está desplazando el dedo por la pantalla.

En este estado, **se calcula dinámicamente el número de píxeles que se debe desplazar el elemento *slider***, utilizando de nuevo el método *.css* sobre el mismo para que se produzca el desplazamiento.

Cabe destacar la siguiente instrucción presente en la función *touchMove*, debido a su importancia en el comportamiento del evento “touchmove”.

```

1 event.preventDefault();

```

Esta instrucción, según la propia [documentación de jQuery](#), evita que se lance la acción por defecto del evento.

En el caso de “touchmove”, en el momento en el que se desliza el dedo por la pantalla se ejecuta la función *touchMove* una única vez; es decir, aunque se deslice de forma continuada, la función no se volverá a ejecutar. Teniendo en cuenta que en este caso, la transición del elemento *slider* debe producirse de forma dinámica y continua, **el comportamiento por defecto no sería adecuado** para resolver este problema.

La instrucción mencionada evita que esto ocurra, **ejecutándose la función de forma reiterada**. Así, se puede ir actualizando la variable que alberga el número de píxeles a desplazar, produciendo un **correcto funcionamiento de las transiciones con el dedo**.

Una vez se deja de tocar, se actualiza el número de pantalla actual, se cambia el indicador activo en la parte inferior de la misma, y se lleva a cabo la transición restante.,

Como se ha expuesto en el fragmento de código mostrado más arriba, el *listener* de los eventos de toque se establece sobre el elemento *slider*. De esta forma, al tocar sobre cualquier elemento descendiente del mismo, se ejecutan las funciones correspondientes, *touchStart*, *touchMove* y *touchEnd*.

En la mayoría de acciones, como pulsar los botones “Atrás” y “Siguiente”, este hecho no presenta ningún inconveniente; sin embargo, existen **dos casos en los que el funcionamiento se ve perjudicado**.

- **La barra lateral que permite regular la sensibilidad:** esta barra tiene establecidas, por defecto, sus propias acciones con las que responder a los toques.

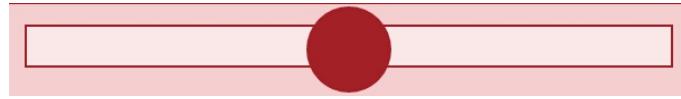


Figura 4.31. Barra lateral que permite regular la sensibilidad de la aplicación.

Así, un usuario puede pulsar, arrastrar y soltar la bola central, desplazándola a lo largo de su vía; sin embargo, y teniendo en cuenta que este elemento HTML es descendiente de *slider*, establecer el *listener* implica **sobreescribir las acciones por defecto por las definidas para los eventos de toque**, siendo imposible desplazar la esfera.

- **El plugin IScroll:** este plugin también presenta incompatibilidades similares a las anteriores, resultando imposible su uso bajo la misma condición.

La solución encontrada consiste en el uso de las instrucciones mostradas a continuación, basadas en la estructura mostrada en la Figura 4.32.

```

1 globalElements.block
2     .on("touchstart", sliderFunctions.stopTouchEventPropagation)
3     .on("touchmove", sliderFunctions.stopTouchEventPropagation)
4     .on("touchend", sliderFunctions.stopTouchEventPropagation);
5
6 stopTouchEventPropagation : function(event) {
7     event.stopPropagation();
8 }
```

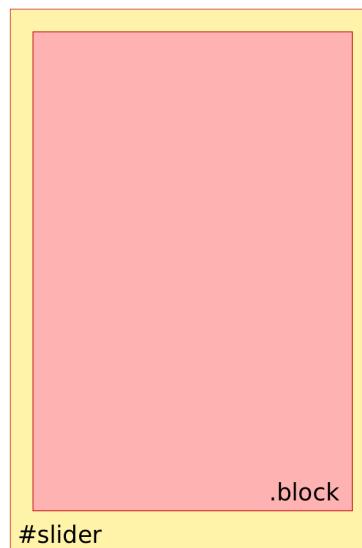


Figura 4.32. Esquema de la disposición de los elementos *block* y *slider*.

Definiendo un *listener* para los eventos de toque sobre el elemento *block*, y usando la instrucción presente en la función *stopTouchEventPropagation*, se consigue evitar el efecto descrito. De este modo, todos los elementos presentes en *block* responden a los efectos de la forma definida por defecto, **funcionando correctamente el conjunto**: los bordes color beige de la Figura 4.32 servirían para desplazarse entre las distintas pantallas, mientras que la sección color granate se comportaría de la forma habitual.

A lo largo de todo el asistente, hay implementadas funciones de validación. De esta forma, el usuario no podrá avanzar si, por ejemplo, introduce un nombre de una sola letra o números de teléfono de 8 dígitos. **Estas comprobaciones sólo se realizan en aquellas en las que sea necesario introducir algún tipo de valor**, ya que es en las únicas en las que tiene sentido hacerlo.



Figura 4.33. Errores en la validación del nombre.

Para ello, se ha utilizado una tabla hash. De esta forma, se consigue automatizar en cierta medida el proceso: sólo es necesario llamar a una función que, teniendo en cuenta el número de pantalla en ese momento, ejecuta la validación del nombre o de los números. La idea fue conseguir un programa que siguiera el **OCP (The Open/Closed Principle)**, de modo que **añadir nueva funcionalidad** implique no reestructurar el código existente, sino el **mínimo número de cambios posibles**. Así, en caso de necesitar validar más campos, sólo habría que incluir la función correspondiente.

En las funciones de validación se han utilizado **expresiones regulares**. Estas son modelos que describen las combinaciones de caracteres de texto aceptadas, de forma que si este no cumple el patrón establecido por las primeras, la validación devolverá un error.

1 `var numberRegEx = /^[0-9]{9}$/;`

Esta expresión, por ejemplo, define un patrón de 9 números, cuyo valor está comprendido entre 0 y 9. Mediante la función *test* de jQuery, se puede evaluar el texto introducido por el usuario y determinar si es correcto.

4.7. PÁGINA PRINCIPAL

La página principal consiste en la pantalla que se va a encontrar el usuario cuando abra la aplicación, una vez completado el asistente de configuración.

Con el fin de mantener el diseño de FallApp sencillo y directo, se optó por situar a simple vista botones que permitan modificar parámetros básicos. La funcionalidad de detección de caídas se ejecuta en segundo plano, evitando la interacción con el usuario y limitando las opciones de configuración posibles.

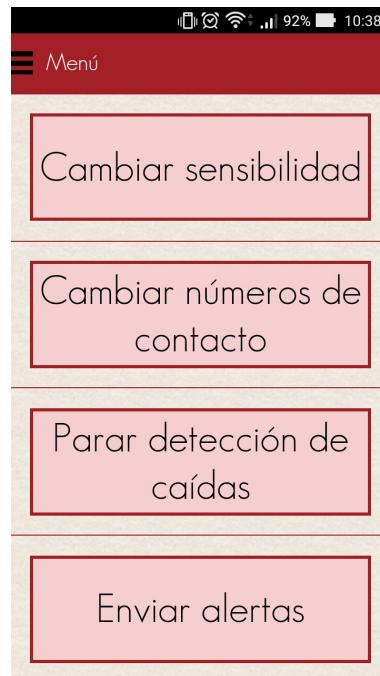


Figura 4.34. Pantalla principal de FallApp.

- **Cambiar los números de contacto:** como su propio nombre indica, permite que el usuario **modifique los números de contacto** que había introducido en el asistente de configuración.
- **Cambiar la sensibilidad:** permite que el usuario **regule la sensibilidad** definida en el asistente de configuración.
- **Parar/arrancar el servicio:** permite, una vez finalizado el asistente y en cualquier momento, **parar o arrancar de nuevo el servicio de detección de caídas**.
- **Enviar alertas:** teniendo en cuenta que se puede dar una situación en la que el usuario necesite pedir ayuda, se añade la opción de **enviar las alertas pulsando un botón**, en lugar de hacerlo de forma automática.

Siguiendo las pautas de diseño de Android, se implementó un menú lateral, en principio oculto a la vista. Cuenta con las siguientes opciones y elementos:

- **Cabecera:** parte superior del menú, donde se muestra el ícono de la aplicación, el nombre del usuario y sus números de contacto.

- **Página principal:** opción seleccionada por defecto, permite **volver a la página principal** en caso de que el usuario se encuentre en alguna de las siguientes secciones.
- **Contacto:** sección con **información de contacto**.
- **Sobre la aplicación:** sección con **información sobre FallApp**.

Los métodos que permiten acceder al mismo son similares a los detallados en la **sección** relativa al asistente de configuración. En este caso, en lugar de contar con una serie de pantallas en disposición horizontal, se cuenta con dos pantallas superpuestas, tal y como se puede ver en la Figura 4.35.



Figura 4.35. Disposición de las pantallas en la página principal.

- **Uso del botón de menú:** la pulsación del botón de menú **desplaza la pantalla principal hacia la derecha**, haciéndolo de esta forma visible.

Para conseguir abrir y cerrar el menú se usa una variable que actúa de estado: en función cuál sea su valor, desplaza la pantalla comentada a la derecha o a la izquierda.

- **Uso de los eventos de toque:** en un principio se definió un *listener*, al igual que en el asistente, en la ventana principal. De esta forma, al hacer un swype en cualquier lugar la misma se desplazaba en un sentido u otro; sin embargo, esta implementación también presentaba incompatibilidades con el plugin iScroll, además de requerir más procesamiento de datos del necesario. En cualquier lugar de la pantalla en el que se tocara, se lanzaban siempre las funciones asociadas a los toques, lo que implicaba ejecutar código de forma inútil.

Por lo tanto, se optó por situar un elemento de una anchura reducida, al que se asociaría el *listener*, en la parte izquierda de la pantalla. El esquema se puede ver en la Figura 4.36.

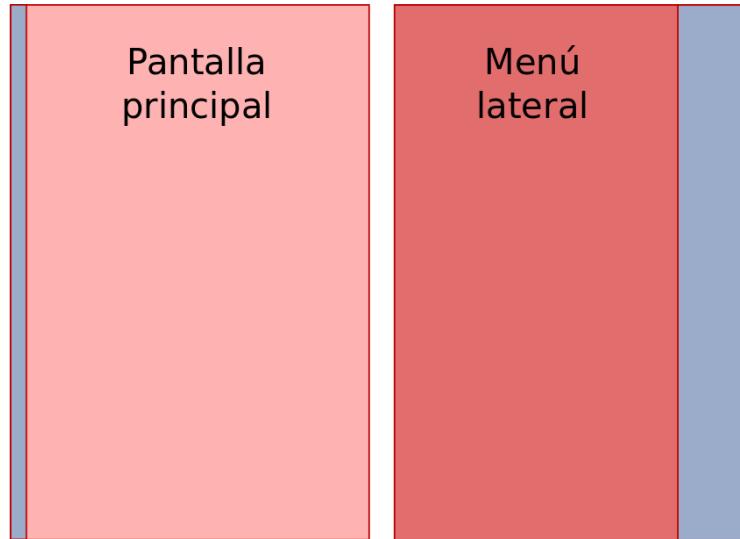


Figura 4.36. Esquema de la pantalla principal de FallApp.

La imagen de la izquierda representa la pantalla principal con el menú oculto, estando el elemento lateral comentado pintado de azul. Es este último el que tendrá asociados los eventos de toque, en lugar de toda la página.

En el momento en que se muestra el menú, la barra lateral adopta todo el espacio restante. De esta forma, se consigue que el usuario pueda cerrarlo de 3 formas posibles:

- Apretando el botón “Menú”.
- Apretando la opción “A pantalla principal” del menú.
- Por medio de un evento de toque pulsando la zona azul.

Ofrecer estas alternativas implica una mejor experiencia de usuario, siendo este un requisito fundamental en una aplicación -aunque la parte más importante corra en segundo plano-. El objetivo es **conseguir una experiencia similar a la obtenida utilizando aplicaciones de Google**, como por ejemplo Gmail.

Por lo tanto, se optó por incluir una opción estética que mejora mucho la impresión de uso de FallApp. El **fondo oscurecido al abrir el menú lateral**, que se puede ver en la Figura 4.37.

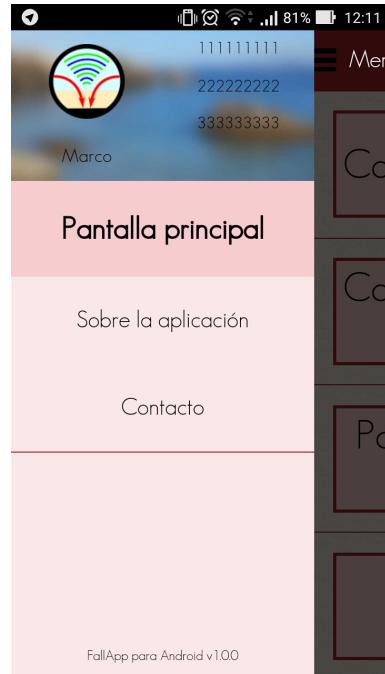


Figura 4.37. Menú lateral de FallApp, con la pantalla principal oscurecida.

Este efecto consigue diferenciar, de forma muy visual, **qué opciones están activas en ese momento**, dejando claro a su vez qué componentes de la interfaz pertenecen a la pantalla principal y no al menú. Es un recurso muy utilizado en aplicaciones con un aspecto visual cuidado, como se puede ver en la Figura 4.38.

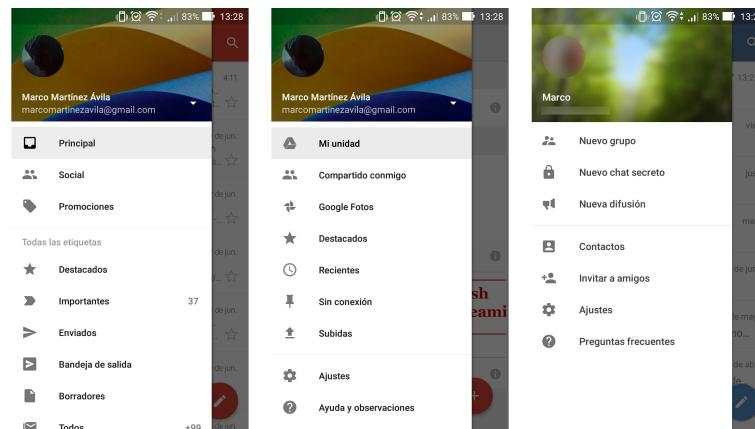


Figura 4.38.- Zona oscura en Gmail (izquierda), Google Drive (centro) y Telegram (derecha).

Este efecto, trasladado a las tecnologías web, equivale a lo que en muchos posts y artículos se conoce como “modo cine”. Es muy utilizado en páginas de reproducción de contenido, donde se oscurecen las partes de la pantalla que no son el vídeo. De esta forma, se mejora sobremanera la visualización, al no brillar tanto los fondos blancos o el resto de elementos.

Aunque existen diversas formas de conseguirlo, y después de barajar y probar varias de ellas, la opción escogida fue la de utilizar **un elemento con fondo negro y una**

opacidad menor que 1.

```

1 <div id="mainWindow">
2   <div id="swypeRight"></div>
3
4   <div id="darken" class="notShowDarken"></div>
5
6   <div id="windowHeader">
7     ...
8   </div>
9
10  <div id="contentContainer">
11    ...
12  </div>
13 </div>

```

El elemento *darken*, oculto por defecto, sólo consiste en un **fondo negro con opacidad variable**. Todas las operaciones que hay que realizar con él se basan en las dos siguientes propiedades CSS:

- **opacity**: un valor “0” en este campo implica que el elemento es totalmente transparente, mientras que “1” implica que es opaco. Una **opacidad de “0.7” con un fondo negro** simula a la perfección el comportamiento visto en la Figura ??.
- **z-index**: cuanto más alto sea el valor numérico asignado, “más arriba” se situará el elemento.

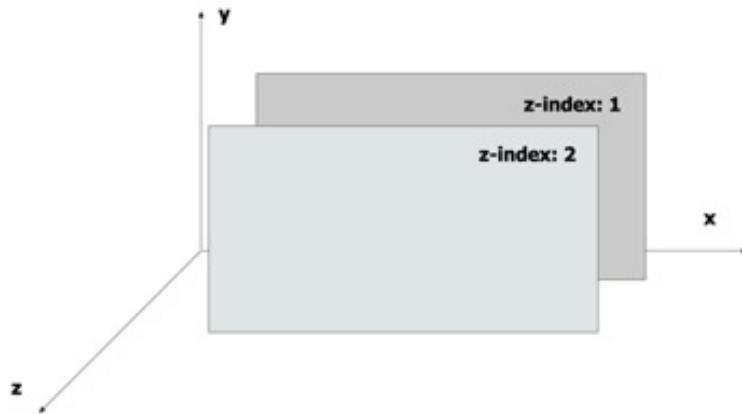


Figura 4.39. Esquema gráfico de la propiedad *z-index* de CSS.

Así, el eje Z sería perpendicular al plano de la pantalla, y cuanto mayor sea el valor del *z-index*, mayor valor tendrá el elemento en este eje. Por lo tanto, estará superpuesto con respecto al resto con menor *z-index*.

Aplicándolo al contexto de FallApp, un *z-index* superior permite mostrar el **fondo negro**, mientras que **reducirlo hasta cierto valor** implica **ocultarlo**.

La combinación de estas dos propiedades permitirá mostrar y ocultar el elemento con fondo negro, a la par que se modifica su opacidad para adaptarla a las condiciones que se den en la aplicación.

La alternativa más simple implica definir una opacidad fija para el elemento *darken* y variar el *z-index* entre dos valores; sin embargo, aunque el resultado inicial (menú cerrado) y final (menú abierto) es el deseado, **el transcurso intermedio está muy poco pulido**: al abrir el menú, instantáneamente se oscurecería la pantalla principal y viceversa, sin ningún tipo de transición. Aunque esta opción sería válida, a lo largo de todo el desarrollo se ha considerado la experiencia de usuario como un aspecto clave, por lo que se optó por **buscar una solución mejor**.

De una forma similar al desplazamiento del menú, **realizar estas operaciones varía entre apretar directamente el botón del mismo, o hacer uso de los gestos para desplazarlo con el dedo**. En todo caso, entran en juego las animaciones CSS usando *keyframes*, por lo que se hará una breve descripción al respecto antes de explicar cómo se aplican en este trabajo.

Los *keyframes* permiten definir cómo se comporta una animación entre los estados inicial y final. En este caso concreto, se pretende que al abrir el menú la opacidad cambie de valor, de “0” a “0.7”.

```

1  @-webkit-keyframes fadein {
2      from { opacity: 0; }
3      to   { opacity: 0.7; }
4 }
```

La animación, de esta forma, tendrá lugar de forma uniforme desde el valor en *from* hasta el valor en *to*. También se pueden definir distintos intervalos con el fin de personalizar cómo se produce la transición, pero para este caso en concreto con el código mostrado es suficiente.

Con respecto al prefijo *-webkit*, como se ha comentado en la sección [Aplicaciones web, híbridas y nativas](#), la aplicación se ejecuta sobre un navegador incrustado en la misma. Webkit es el motor de navegación usado por ese navegador, que **interpreta el código recibido** por este último para mostrarlo por pantalla de la forma en que lo interpretan las personas.

Existen distintos motores de navegación con sus respectivos prefijos usados por los diversos navegadores del mercado, como *-moz* para Firefox, o *-ms* para Internet Explorer. Mediante estas cadenas, se indica que una determinada propiedad CSS sólo debe ser procesada por el motor correspondiente.

Una vez aclarados estos conceptos, se puede exponer cómo conseguir el efecto deseado.

■ Uso del botón de menú:

- **Al abrir el menú lateral:** en el mismo instante en el que se pulsa el menú, se modifica el *z-index* para que el elemento *darken*, con fondo negro, se superponga y aparezca en primera línea. Además, se le asigna una clase CSS que ejecuta la animación *keyframes* mostrada anteriormente a lo largo de 0.2 segundos.

- **Al cerrar el menú lateral:** esta acción conlleva un inconveniente visual importante. Si bien antes se modificaba el *z-index* inmediatamente después de pulsar el botón, hacer lo mismo en este caso implica que *darken* desaparece al instante de la pantalla, siendo totalmente invisible la animación que elimina la opacidad.

Por lo tanto, fue necesario encontrar una solución que permitiera: **primero, realizar la animación de la opacidad; y segundo, cambiar el *z-index* para enviar el elemento *darken* a segundo plano en la pantalla.** Esa solución, después de estudiar las funciones de *callback* de jQuery con resultados infructuosos, fueron los eventos que se lanzan cuando finaliza una animación.

```

1  this.elements.darken
2      .one("webkitAnimationEnd", function() {
3          menuFunctions.elements.darken
4              .removeClass("showDarken removeOpacity")
5              .addClass("notShowDarken");
6      });

```

Asignando un *listener* a *darken* sobre el evento “*webkitAnimationEnd*”, se consigue **ejecutar una función en el momento en que finalice la animación;** en este caso, cuando la opacidad llegue a valor 0. De esta forma, se puede modificar el *z-index* y hacer desaparecer el elemento, **sin repercutir en la estética y la experiencia de usuario de la aplicación.**

■ Uso de eventos de toque:

- **Al abrir el menú lateral:** con el fin, de nuevo, de conseguir la mejor experiencia de usuario posible, **la opacidad varía a medida que el usuario mueve el dedo por la pantalla.** Para ello, se calcula qué valor de la misma varía con cada píxel que se desplaza el elemento *mainWindow*, modificándolo de forma dinámica al hacer el swype lateral.

Este efecto, si bien es visualmente muy atractivo, presenta un inconveniente con respecto al explicado en **Uso del botón de menú.** En ese último, el valor de la opacidad siempre variaba entre 0 y 0.7; sin embargo, en este caso no ocurre lo mismo, ya que tendrá que **variar desde la cifra correspondiente a los píxeles que haya desplazado el usuario, y 0.7.** Esta cifra no se puede prever y especificar en el archivo CSS *a priori*, por lo que es necesario calcularla dinámicamente con jQuery.

```

1  .setOpacityFromValue {
2      -webkit-animation: fadeinFromValue 0.2s forwards;
3  }

```

La forma encontrada más eficiente para conseguir un efecto perfecto fue definir *a priori* una clase CSS que ejecute una animación, pero sin estar esta definida inicialmente.

Así, en el momento en que el usuario quita el dedo de la pantalla, se agrega la misma con el valor inicial que corresponda al punto en el que se dejó de

tocar.

```
1 var style = document.createElement("style");
2 style.type = "text/css";
3 style.innerHTML = "@-webkit-keyframes fadeinFromValue {
4     from { opacity: " + this.var.actualOpacity + "; }
5     to { opacity: 0.7;}}";
6 document.getElementsByTagName("head")[0]
7 .appendChild(style);
```

Asignado la clase mencionada -que ejecuta la animación recientemente creada- al elemento *darken*, este cambia su opacidad de forma fluida desde un valor aleatorio hasta 0.7.

La interfaz de la página principal, dado que está pensada para ser usada por personas mayores, se ha diseñado con el objetivo de ser **simple y directa**. De este modo, se han utilizado, como se puede ver en la Figura 4.34, botones grandes y visuales, tanto en la pantalla principal como en el menú lateral. Para ello, se ha hecho uso del modelo de cajas flexibles de CSS3.

La propiedad *flex* permite distribuir el espacio que ocupan una serie de elementos, adaptándolos al espacio disponible en función de las necesidades existentes. Supone una evolución con respecto al modelo *bloque* tradicional y las conocidas instrucciones *float* o *clear*, ya que se puede **controlar de una forma mucho más precisa el comportamiento de los elementos del DOM**, así como conseguir de una forma sencilla lo que antes requería de complejas líneas de código o trucos.

Este modelo de diseño distingue entre dos clases de elementos: el contenedor, y los hijos contenidos en el mismo. Cada uno permite una determinada serie de parámetros, pudiendo personalizar aspectos como la dirección a lo largo de la que se disponen estos últimos, cómo se distribuyen a lo largo de ese eje o cómo ocupan el espacio disponible.

Los botones para cambiar los números de contacto y la sensibilidad, además de los elementos del menú, consisten en un flujo vertical -de arriba a abajo-, ocupando los hijos todo el espacio horizontal disponible. El esquema se puede ver en la Figura 4.40.

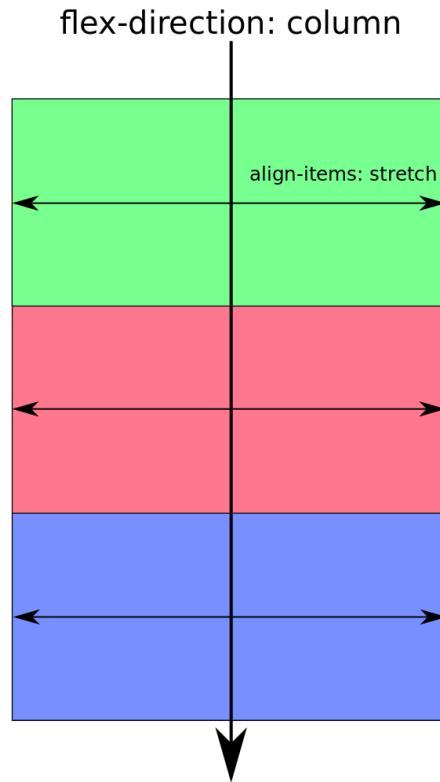


Figura 4.40. Esquema gráfico del modelo de cajas *flex*.

Para acabar con la pantalla principal, cabe mencionar **las ventanas emergentes y las acciones que se desempeñan al pulsar los botones disponibles**.

- **Ventanas emergentes:** al pulsar algunos de los botones, es necesario que el usuario siga interactuando para completar el objetivo que pretende conseguir. Para ello, en ese momento aparecerán, en función de dónde se pulse, unas ventanas emergentes que complementan la acción -como modificar los números de teléfono o cambiar la sensibilidad, como se puede ver en la Figura 4.41-.



Figura 4.41.- Ventanas emergentes al cambiar la sensibilidad (izquierda) y modificar los números de contacto (derecha).

- **Acciones de los botones:** las funciones que desempeñan requieren una comunicación con el servicio de detección de caídas. Por lo tanto, es necesario establecer un puente entre la interfaz web y las clases Java, que se tratará más adelante en la sección [Comunicación entre la interfaz web y el núcleo Android](#).

4.8. VISTA DE ALERTAS

La vista de alertas consiste en la pantalla que se abrirá **cuando el algoritmo de detección de caídas no pueda asegurar un incidente**. De esta forma, se usará esta página para que el usuario pueda especificar si ha sufrido un accidente.

Constará, únicamente, de dos botones con lo que contestar con un “Sí” o un “No” a la pregunta: “¿Te has caído?”. Además, en la parte inferior se situará un temporizador, indicando el tiempo restante para responder -antes de enviar las alertas-.

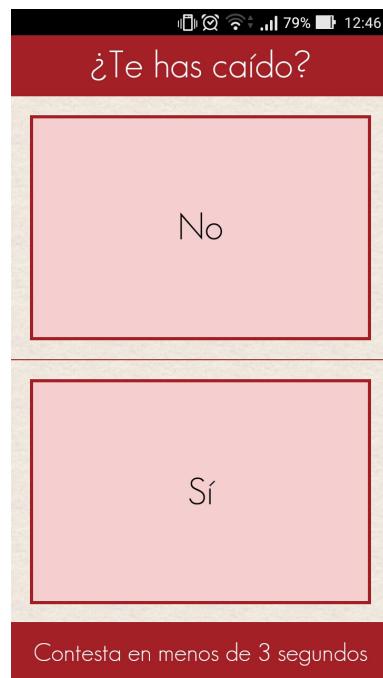


Figura 4.42. Vista de alertas de FallApp.

Como esquema general de toda la interfaz, en la Figura 4.43 se puede ver el flujo de pantallas que seguirá un usuario de FallApp.

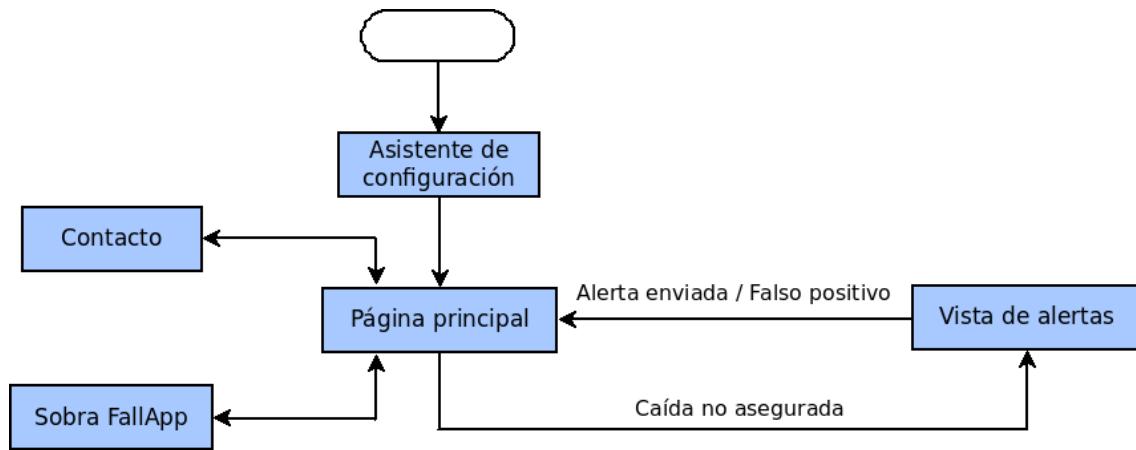


Figura 4.43. Flujo de pantallas en FallApp.

4.9. ESCALABILIDAD EN LA INTERFAZ WEB

Existen multitud de dispositivos Android en el mercado, de diferentes marcas, con **distintos tamaños de pantalla y resoluciones**. La Figura 4.44, por ejemplo, muestra una comparación entre distintas resoluciones encontradas en las pantallas de los smartphones.

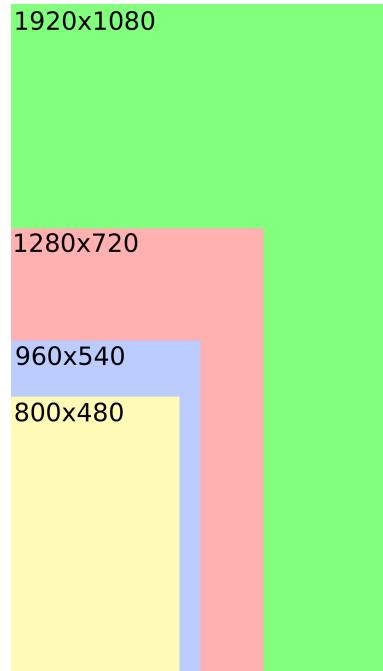


Figura 4.44.- Comparación de las distintas resoluciones de smartphones disponibles en el mercado.

La conjunción de tamaño y resolución desemboca en una **variedad muy alta de densidades de píxeles en una pantalla**, lo que repercute directamente en la visualización de contenido. Una aplicación debería ser capaz de **adaptar su interfaz**, proporcionando un aspecto visual cuidado en cualquier resolución. Aunque existen

multitud de unidades de medida, como los píxeles o los milímetros, Android proporciona un mecanismo que permite conseguir esta adaptación de una forma directa: los **dp**.

Los dp toman como referencia una densidad de píxeles media (MDPI), y se escalan en función de la que tenga la pantalla en la que se visualizan. De esta forma, los elementos se pueden representar de forma aceptable en un amplio rango de alternativas.

Sin embargo, en las tecnologías web no existen los dp, sino que se cuenta con unidades típicas como **píxeles (px)**, **centímetros (cm)** o **porcentajes (%)**. Estas **no responden tan bien ante la variedad de densidades de píxeles**, lo que produce, por ejemplo, que el escalado de los textos sea más complicado.



Figura 4.45. Fallo en el escalado de texto en distintas resoluciones.

Por norma general, a la hora de definir tamaños de bloques resulta conveniente utilizar porcentajes. Este método asegura que el ancho de un contenedor, por ejemplo, ocupará siempre la misma proporción de pantalla sea cual sea el dispositivo en el que se represente; sin embargo, en algunos elementos se optó por definir unas dimensiones fijas en cm, principalmente debido a dos motivos:

- **Modo apaisado:** aunque no se ha elaborado una interfaz acorde al modo apaisado, utilizar los porcentajes producía una distribución en pantalla muy mala al estar el móvil en horizontal; **un porcentaje equivale a un distinto número de píxeles estando el smartphone en vertical u horizontal.**
- **Aparición del teclado:** implica que, al pulsar el usuario un cuadro de texto, el teclado ocupe espacio en pantalla. De esta forma, el 100% de la misma pasa a equivaler a menos píxeles que en cualquier otra situación, produciéndose así un **redimensionado de los bloques**.

Algunos elementos con una o dos dimensiones fijas son la cabecera del asistente y de la pantalla principal, los indicadores inferiores de la ventana actual, y los botones para añadir o quitar números de contacto.

El texto, por su parte, supone un problema aparte. El valor definido por el programador se toma **en base a una cifra de referencia**, variando esta entre los distintos dispositivos. Por lo tanto, la solución encontrada fue **definir un determinado tamaño de fuente por defecto para todo el documento**, de forma que el parámetro *font-size* se adecue en función del mismo.

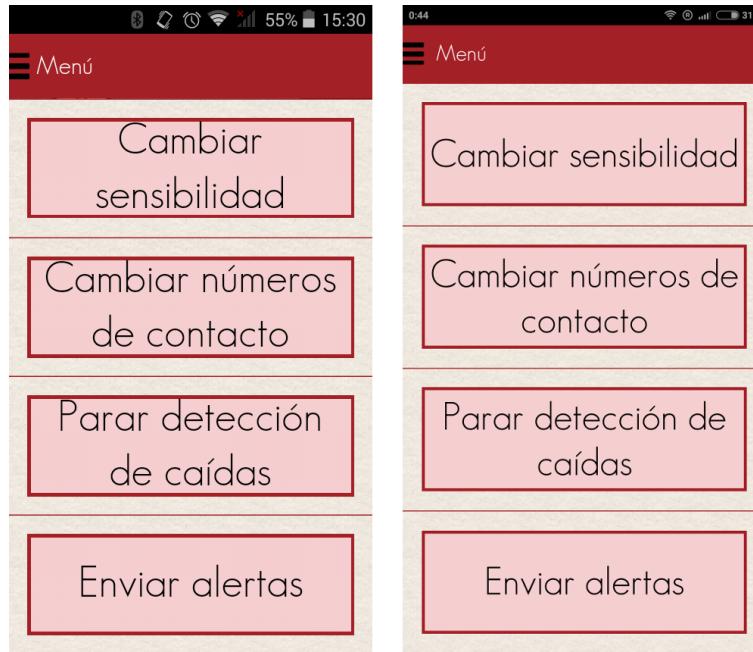


Figura 4.46. Escalado de texto correcto en distintas resoluciones.

Esta alternativa no es ideal y resulta menos adecuada que los dp, pero entra dentro de las opciones disponibles en las tecnologías web.

4.10. PRUEBAS DE RENDIMIENTO

Como se ha comentado en la sección Aplicaciones web, híbridas y nativas, uno de los mayores contras en el desarrollo de las dos primeras es el **rendimiento**.

Independientemente de la eficiencia del código web programado, por definición estas son más lentas que las aplicaciones nativas, ya que **no aprovechan al máximo el hardware del dispositivo ni es código específicamente diseñado para operar en esa plataforma**. Por lo tanto, es labor del desarrollador conseguir, dentro de lo que cabe, un **código limpio** que no repercuta negativamente en el ya de por sí inferior rendimiento.

Con este fin, se han ejecutado varias pruebas que permiten comprobar **cuánto tiempo tardan en ejecutarse algunas instrucciones JavaScript o jQuery**. La elección de buenos selectores, o de un algoritmo eficiente, pueden repercutir en la eficiencia general de la aplicación; por lo tanto, siempre es recomendable escoger la opción que mejor desempeño presente, ya que es cuando crecen los programas cuando se pueden empezar a notar los síntomas de una mala programación.

La forma seleccionada para medir el tiempo empleado en ejecutar código es el método *performance.now()*. Proporciona una precisión de un microsegundo, al contrario de otros métodos como *Date.now*, limitados a una resolución de un milisegundo. Ade-

más, permite ejecutar el mismo código en el navegador incrustado en la aplicación del smartphone, y compararlos con los resultados obtenidos con el homólogo del portátil.

En este caso, se ejecutarán las mismas instrucciones 2000 veces, para que el tiempo obtenido sea relevante.

```

1 var startTime = performance.now();
2 for (var i=0; i<2000; i++) {
3     doSomething();
4 }
5 var endTime = performance.now();
6
7 alert((endTime-startTime) + " ms");

```

Todas las pruebas se realizarán, por un lado, **accediendo al elemento del DOM cada vez que se ejecute un método**; y por otro, **utilizando una variable**. De este modo, también se podrá comprobar cómo repercute este aspecto.

4.10.1. USO DE SELECTORES

Aunque en proporción a otras acciones el uso de un selector resulta casi insignificante, es un **recurso utilizado muy a menudo**, por lo que es un buen punto de partida de cara a conseguir una buena optimización.

Algunos de los selectores disponibles en jQuery se basan en funciones nativas de JavaScript, como el caso de `$(“p”)` y `document.getElementsByTagName()`. Estos son mucho **más rápidos que otros que recorren el DOM buscando coincidencias**, por lo que será la opción preferida en caso de ser posible. Más concretamente, **el uso del selector de id es el que menos tiempo implica**, basándose en la función `document.getElementById()`.

Para que las pruebas sean lo más realistas posibles, se usarán algunos selectores sobre los mismos elementos del HTML.

```

1 <div id="element" class="element" name="element">
2     <p id="text"></p>
3 </div>
4
5
6 startTime = performance.now();
7 for (var i=0; i<2000; i++) {
8     $("#element");
9     //$(".element");
10    //$("#div");
11    //$("#[name='element']");
12    //$("#element").find("p");
13    //$("#element p");
14 }
15 endTime = performance.now();
16 alert((endTime-startTime) + " ms");

```

Los resultados obtenidos se pueden ver en la Tabla 7.

	Aplicación híbrida (ms)		Navegador Firefox (ms)	
	Sin variable	Con variable	Sin variable	Con variable
<code>\$("#element")</code>	25.5019	0.5739	12.4322	0.9402
<code>\$(".element")</code>	46.7809		35.0233	
<code>\$("div")</code>	34.4330		28.9961	
<code>\$("[name='element'])</code>	49.7539		41.8803	
<code>\$("#element").find("p")</code>	60.9410		25.5966	
<code>\$("#element p")</code>	26.6499		34.4804	

Tabla 7.- Tiempo empleado en acceder a un elemento del DOM utilizando distintos selectores.

Así, se ha podido comprobar cuál es el rendimiento real de algunos selectores, más allá de la teoría. Por ejemplo, según [alguna página web](#) consultada, el selector `$("#element p")` es más lento que `$("#element").find("p")`; sin embargo, si bien eso puede ser cierto en sitios mucho mayores -o en el navegador del ordenador, como se ha visto-, ni en el acceso al elemento de esta prueba ni en otras realizadas en la aplicación se ha cumplido.

Los resultados también pueden ayudar a hacerse una idea de la **diferencia existente entre la ejecución de una web en un ordenador y en un smartphone**, aunque este sea de gama alta.

4.10.2. ESTILOS

Si bien la interacción del usuario se consigue usando JavaScript o jQuery, los estilos CSS juegan un **papel importante en cómo cambian los distintos elementos con el uso de la aplicación**.

Por ello, existen instrucciones en jQuery que nos permiten modificar los estilos dinámicamente, como el método `css`; sin embargo, según muchas webs consultadas, **usar JavaScript -o alguno de sus frameworks- implica un mayor consumo de recursos que utilizar directamente la hoja de estilos**. A su vez, y aplicando un razonamiento similar, es menos eficiente cambiar las propiedades CSS de un elemento utilizando el método `css`, que cambiando la clase del mismo.

Por lo tanto, es interesante comprobar cómo repercuten estas acciones en una aplicación híbrida, siempre orientando el desarrollo a conseguir un consumo mínimo de recursos y una experiencia fluida en FallApp. Las dos pruebas realizadas se describen a continuación:

- **Modificación de estilos con jQuery:** como se ha comentado anteriormente, es **más recomendable utilizar clases CSS** para modificar los estilos de un elemento, en lugar de hacerlo utilizando JavaScript o jQuery. Las dos opciones posibles, por tanto, son los dos métodos mostrados a continuación.

```

1  $("#element").css("background-color", "#000000");
2  $("#element").addClass("black");

```

Donde *black* es una clase CSS que realiza la misma modificación que el método jQuery *css*.

```

1 .black {
2     background-color: #000000;
3 }
```

Por lo tanto, el objetivo en ambos casos es cambiar el color de *element* a negro.

- **Concatenación de instrucciones:** algunas de las acciones que más han sido llevadas a cabo en el desarrollo de FallApp es **añadir, quitar y cambiar estilos**.

En jQuery, una vez aplicado un método este devuelve un objeto, de forma que se le puede volver a aplicar otro método. Es lo que se conoce como concatenación o encadenamiento, y es recomendable hacer uso de este recurso ya que es más rápido que utilizar los métodos por separado.

```

1 $("#element").removeClass("white");
2 $("#element").addClass("black");
3
4 $("#element").removeClass("white").addClass("black");
```

	Aplicación híbrida (ms)	Navegador Firefox (ms)		
	Sin variable	Con variable	Sin variable	Con variable
Método <i>css</i>	134.0290	84.8450	57.3049	38.6602
Método <i>addClass</i>	56.2069	18.7399	21.4985	13.1170
Sin concatenación	113.1570	40.4769	31.8906	20.0845
Con concatenación	67.0469	37.1560	24.3373	18.6004

Tabla 8. Tiempo empleado en modificar estilos de un elemento.

4.11. ORGANIZACIÓN DEL CÓDIGO JAVASCRIPT

Una aplicación Cordova genera, automáticamente, un archivo .js que añade funcionalidad a los correspondientes .html y .css. Este viene con una estructura básica, que consta de un objeto con varios métodos; sin embargo, **es muy fácil perder esa estructura y crear funciones globales que ensucian rápidamente el código**.

A lo largo del desarrollo, la simplicidad de los lenguajes JavaScript o jQuery, y el correcto funcionamiento de la aplicación a pesar de ir sumando pequeños fallos, hicieron que el código de los archivos .js de FallApp resultara sucio y desorganizado. De este modo, se organizaba toda la funcionalidad en distintas funciones globales, contando también con bastantes variables de ese tipo.

En estos lenguajes de programación existen **diversos métodos de organización de código**, variando en función del proyecto o las preferencias del autor. Uno de los patrones más sencillos es el ***object literal, utilizado en este desarrollo***.

El patrón *object literal* consiste en **agrupar el código en distintos módulos**, generalmente teniendo en cuenta las funcionalidades que proporcionan. Se le asigna

un objeto a una variable, pudiendo por medio de ella acceder a todos los campos que forman al primero.

Cada módulo consiste en una serie métodos y propiedades que posibilitan el eliminar código desorganizado como las funciones anónimas. Todos los campos son públicos, pudiendo así ser accedidos desde cualquier otro objeto del archivo.

```
1 var functionality = {  
2     variables : {  
3         lastWrongTelNumber : 0,  
4         wrongTelNumber : 0,  
5         errorCode : -1,  
6         stop : false  
7     },  
8     changeSensibility : function() {  
9         ...  
10    },  
11     updateRangeText : function(value) {  
12         ...  
13    },  
14     ...  
15 }
```

Esta forma de organización permite tener un **código claro y limpio**, aumentando la **reusabilidad de los módulos**, la **mantenibilidad**, y siendo más sencillo seguir de forma correcta el Open/Closed Principle.

4.12. DESARROLLO DEL ALGORITMO DE DETECCIÓN DE CAÍDAS

Para conseguir determinar **cuándo se produce una caída**, FallApp hace uso del **acelerómetro del móvil**, cuyo funcionamiento se ha detallado en la sección [El acelerómetro](#). Como se ha comentado con anterioridad, el objetivo de la aplicación es funcionar como un sistema de monitorización, analizando los valores devueltos por este sensor para determinar en qué momento ocurre una anomalía.

4.12.1. TOMA DE DATOS CON EL ACELERÓMETRO

Android proporciona mecanismos para la recogida de estos datos de forma sencilla, abstrayendo al programador de la misma y permitiendo realizar esta acción de forma sencilla. Esta tarea se consigue por medio de la interfaz *SensorEventListener*.

Java, como algunos otros lenguajes orientados a objetos, **no permite la herencia múltiple**, por lo que Android no soporta esta característica. En su lugar, permite una **aproximación a esta idea por medio de las interfaces**. Una interfaz consiste en la declaración de un conjunto de métodos públicos y abstractos que permiten ampliar las características del proyecto. Al contrario de lo que ocurre con las clases, que pueden tener sus propias implementaciones de estos métodos, no están implementados en la

clase padre -sólo definidos-, por lo que siempre deben estar sobreescritos en la clase hija.

En lugar de la palabra reservada *extends*, necesaria en caso de heredar, se utiliza la palabra *implements*, tal y como se puede ver a continuación.

```
1 public class FallService implements SensorEventListener {  
2     ...  
3 }
```

Siendo *SensorEventListener* una interfaz, por tanto, es necesario desarrollar los métodos presentes en ella, dos en este caso: *onAccuracyChanged* y *onSensorChanged*.

- ***onAccuracyChanged***: método al que se llama cuando la precisión del sensor cambia.
- ***onSensorChanged***: el método más importante de la interfaz, ya que contiene gran parte de la lógica del algoritmo. **Este método es llamado cuando el sensor capta nuevos datos**. En función del período de muestreo especificado a la hora de comenzar la recogida de datos, se le llamará con más o menos frecuencia.

Cabe destacar que esta interfaz permite recabar datos de distintos sensores, no sólo del acelerómetro. Por lo tanto, es labor del programador especificar en cuál está interesado por medio del registro de un *listener*. Este se configura para cada sensor en particular, junto con un parámetro comentado anteriormente y que resulta de interés en este proyecto: el período de toma de datos.

La clase *SensorManager*, que junto con la interfaz permite acceder a los sensores del dispositivo, cuenta con unos campos que permiten definir ese período, entre varias opciones disponibles en Android.

- **SENSOR_DELAY_NORMAL**: valor por defecto, es el período más amplio y el adecuado para detectar eventos simples como cambios en la orientación del dispositivo. Su valor, explicado más adelante, es de 3.
- **SENSOR_DELAY_GAME**: con un período más reducido, este parámetro es adecuado para responder ante las exigencias de juegos que hagan uso de sensores. Su valor es 1.
- **SENSOR_DELAY_FASTEST**: el período de toma de datos más corto (y por tanto, que más recoge los datos). Su valor es 0.
- **SENSOR_DELAY_UI**: equivale a un valor intermedio, que se consideró el más adecuado para la aplicación FallApp. Su valor es 2.

Como se puede comprobar, cada campo equivale a un determinado valor, comprendido entre 0 y 3. **Esta cifra permite acceder al número de milisegundos que conforman el período**; es decir, el tiempo que transcurre entre cada muestreo.

El código mostrado a continuación explica lo expuesto anteriormente. Es un fragmento del [código fuente](#) de Android, donde se puede examinar cómo funciona este

sistema operativo a nivel interno y cómo repercuten los valores comentados en el período del sensor. Aunque la versión es la 5.0.1 Lollipop, este método es igual en otras versiones como 4.4 KitKat o 4.1-4.3 Jelly Bean.

```
1 private static int getDelay(int rate) {
2     int delay = -1;
3     switch (rate) {
4         case SENSOR_DELAY_FASTEST;
5             delay = 0;
6             break;
7         case SENSOR_DELAY_GAME;
8             delay = 20000;
9             break;
10        case SENSOR_DELAY_UI;
11            delay = 66667;
12            break;
13        case SENSOR_DELAY_NORMAL;
14            delay = 200000;
15            break;
16    }
17    return delay;
18 }
```

En este *switch*, por tanto, y en función del valor correspondiente a los campos del *delay*, se define el período del sensor. En el caso actual, este sería de **66.667 ms**.

Sin embargo, según se especifica en la [página de desarrolladores](#), **este parámetro sólo es un indicio o una orientación al sistema**, siendo este último el que controlará cada cuánto se toman las medidas. De esta forma, por norma general **el período real de toma de datos será menor que el especificado mediante el campo *SENSOR_DELAY_UI***.

Según las pruebas realizadas con diferentes smartphones, el rango de variación abarca desde los 10ms, hasta aproximadamente 60ms. En este último caso la toma de datos se realiza sin inconvenientes, obteniendo nuevas cifras correspondientes a los 3 ejes del acelerómetro cada vez que es llamado el método *onSensorChanged*; sin embargo, que en último término el sistema regule la velocidad de forma independiente resulta un problema en el primer caso. El sensor devolvía valores aproximadamente 4 veces en 60ms, con el principal inconveniente de que eran repetidos. Este hecho implica registrar entre 3 y 5 datos exactamente iguales, lo que aunque a efectos prácticos no conlleva ningún problema, tampoco implica puntos ventajosos.

4.12.2. ANÁLISIS DE CAÍDAS

Una vez habilitada la toma de datos del acelerómetro con el dispositivo, es posible **analizar los valores devueltos por el mismo tanto en una caída, como en algunas acciones cotidianas**. De esta forma, se puede **buscar puntos diferenciales** entre la primera y el resto de sucesos despreciables por la aplicación: no es labor de FallApp saber qué está haciendo el usuario en cada momento por medio del procesamiento de datos, sino diferenciar todas las acciones que una persona hace en su día

a día de una caída. Por lo tanto, la idea principal es comprobar cómo se comporta el acelerómetro en la misma, y cómo se puede diferenciar.

Aunque Cordova, como se ha explicado en su sección, permite acceder a funciones nativas como el acelerómetro o el GPS, en FallApp esta tarea se desempeña utilizando código nativo. Así, se consideró conveniente **crear una aplicación que posibilitara la toma y guardado de datos**, atendiendo a los criterios expuestos en el apartado anterior.

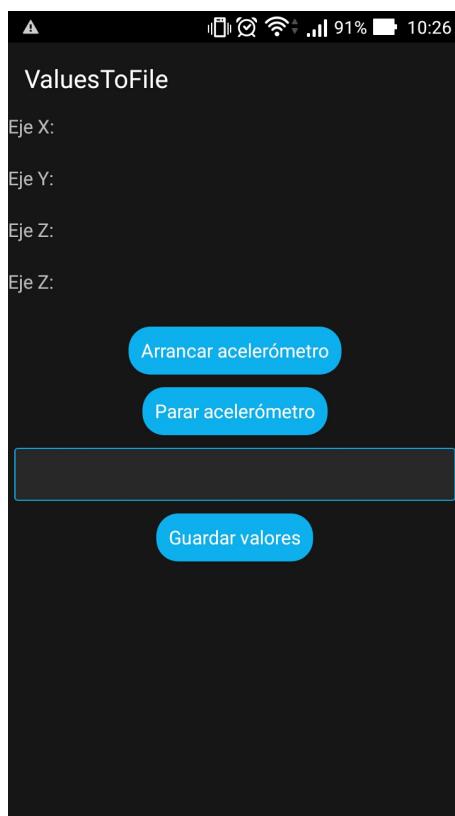


Figura 4.47. Aplicación de recogida y guardado de datos del acelerómetro.

Como se puede ver en la Figura 4.47, esta aplicación sólo permite opciones muy básicas para obtener un fichero .txt con los datos deseados. Utilizando programas como Matlab, se pueden representar y obtener gráficas que ayuden a **entender qué se está registrando**.

Cabe destacar que, aunque el acelerómetro devuelve valores en unidades del Sistema Internacional (SI), m/s^2 , estos **se han representado en función de la aceleración de la gravedad (g)**, siendo así más fácil interpretarlos y determinar como se relacionan con esta última.

A continuación se muestran algunas de las figuras obtenidas. En todas ellas se pueden observar 4 gráficas, correspondientes a los **ejes X, Y, Z, y al módulo de la aceleración**. Este último va a ser el determinante de cara a identificar choques y constancia de valores, como se podrá comprender en la explicación de la Figura 4.50.

Móvil boca arriba reposando en una mesa

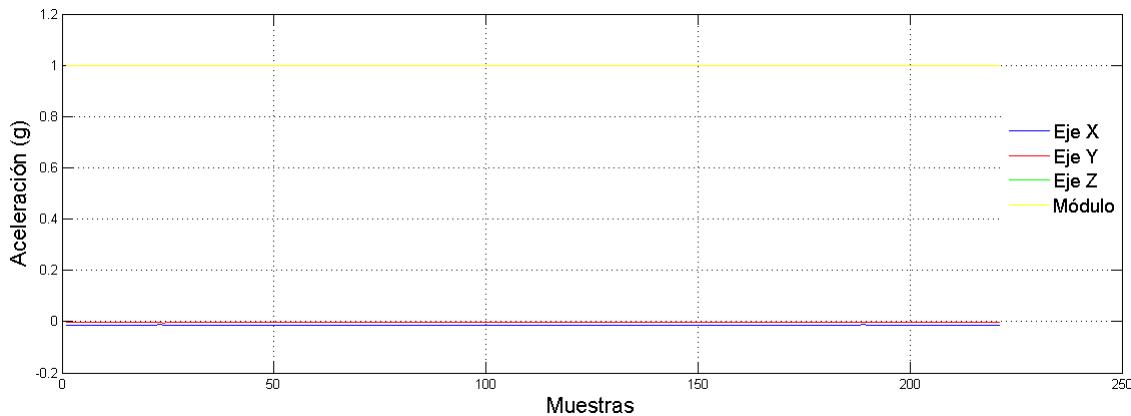


Figura 4.48. Datos obtenidos con el móvil en reposo sobre una superficie plana.

La Figura 4.48 muestra cómo se comporta el acelerómetro al estar el smartphone encima de una mesa. En función de la disposición de ejes planteada en la sección [El acelerómetro](#), este es un ejemplo práctico de lo expuesto con la Figura 3.7.

Los ejes X e Y permanecen muy próximos a 0, manteniendo el Z un valor de 1g. Se comprueba, así, como el móvil está bajo los efectos de la gravedad, sin (idealmente) actuar sobre él ninguna otra fuerza.

Móvil boca arriba inclinado

Con el fin de demostrar de forma práctica lo expuesto en la introducción de este TFG, se ha realizado, esta vez de forma real, la misma prueba que en la Figura 3.8. Los valores obtenidos se pueden observar en la Figura 4.49.

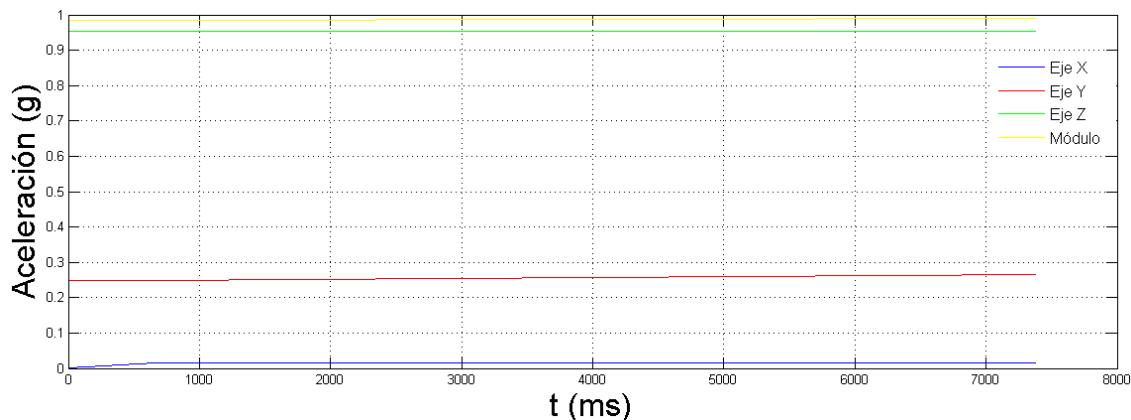


Figura 4.49. Datos obtenidos con el móvil inclinado sobre una superficie plana.

En este caso, los dos ejes entre los que se reparte la fuerza de la gravedad son el Y y el Z, mostrando la aceleración que se puede ver en la imagen. La explicación de por qué ocurre esto se puede leer en la sección [El acelerómetro](#).

Movimientos aleatorios en la mano

Esta gráfica se ha elaborado para plasmar la **importancia del módulo de la aceleración en la detección de impactos**, y la relativa irrelevancia de los valores individuales de cada eje.

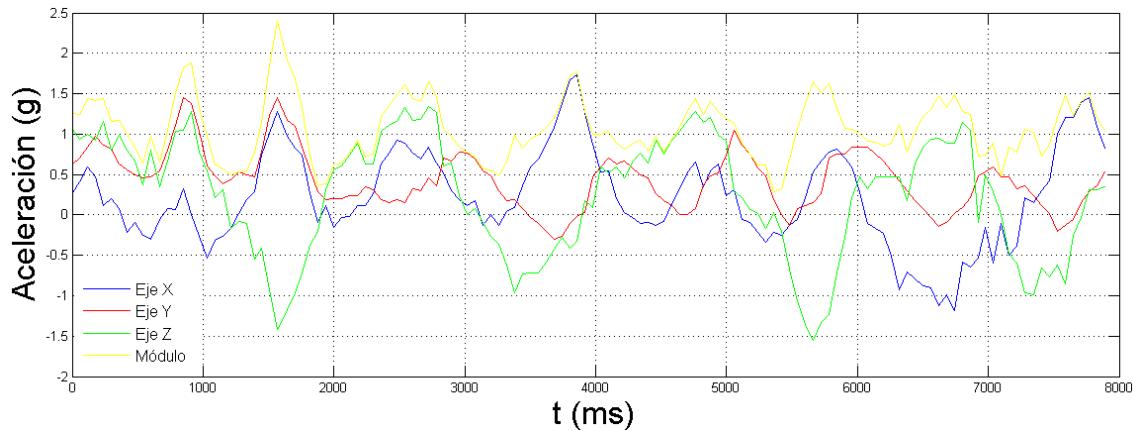


Figura 4.50. Datos obtenidos con el móvil haciendo giros con la mano.

A lo largo de todo el muestreo, se puede comprobar cómo los datos de los ejes fluctúan sin parar debido al movimiento del smartphone. Analizarlos e interpretarlos conllevaría un consumo de recursos innecesario, ya que se ha visto que pueden tomar valores prácticamente aleatorios con movimientos nimios; sin embargo, el módulo tiene un comportamiento diferente. Aunque también sufra variaciones importantes, siempre se encuentra entorno al valor de 1g y presenta cifras, por lo general, más representativos del conjunto en lo que respecta a la detección de impactos.

Mantener el dispositivo quieto en una determinada posición siempre devolverá resultados similares a los de la Figura 4.49, mientras que **las fluctuaciones apreciables en esta se corresponden con cambios de orientación, giros o movimientos**. En las caídas, se produce un **instante en el que el cuerpo golpea contra el suelo**, con la consecuente variación en su velocidad siendo registrada por el acelerómetro como un pico.

Sentarse en una silla

Sentarse en una silla es un acto cotidiano que **puede producir una importante repercusión en la aceleración**, siendo por tanto objeto de estudio.

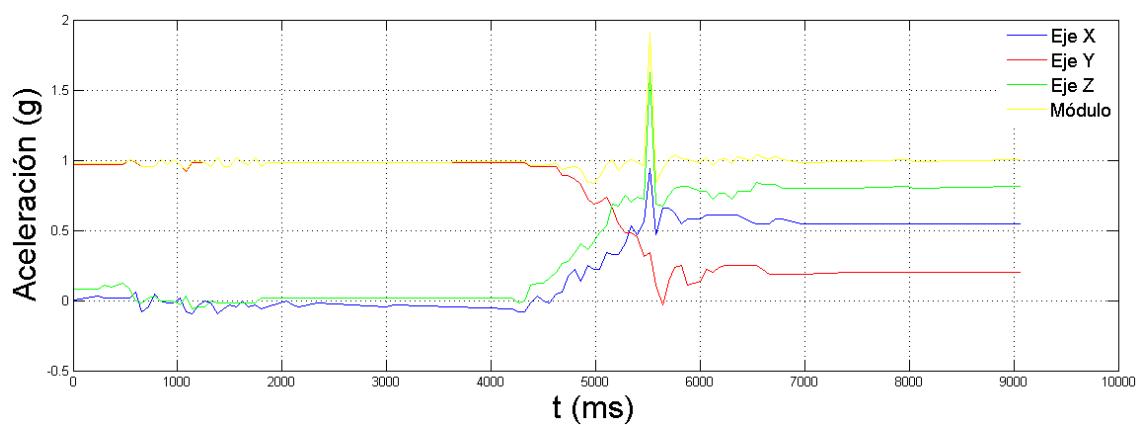


Figura 4.51. Datos obtenidos al sentarse con el móvil en el bolsillo.

La Figura 4.51 corresponde al smartphone guardado en el bolsillo del pantalón, sentándose en una silla. La prominencia del valor Y sobre el resto indica que el móvil

está en vertical, debido a su situación y al estar la persona de pie en ese momento. El pico que se observa aproximadamente en el segundo 5 y medio viene dado por el impacto con la silla. Una vez completado el movimiento, el acelerómetro registra valores constantes.

Al igual que en el ejemplo anterior, los 3 ejes presentan comportamientos que, si bien son útiles de cara a comprender la acción, no lo son tanto al identificar un golpe. El módulo, por su parte, sufre una variación en el momento del impacto con la silla. **El fundamento detrás de esta situación es el mismo que en el caso de una caída: esa variación servirá para determinar en qué momento se produce un impacto de una persona contra el suelo**, ya que el valor devuelto por el sensor en ese momento será sensiblemente superior al resto.

Levantarse de una silla

El acto de levantarse es similar al de sentarse, ya que consiste en volver al estado inicial. En la Figura 4.52 se puede comprobar cómo se produce una transición similar en el momento de incorporarse.

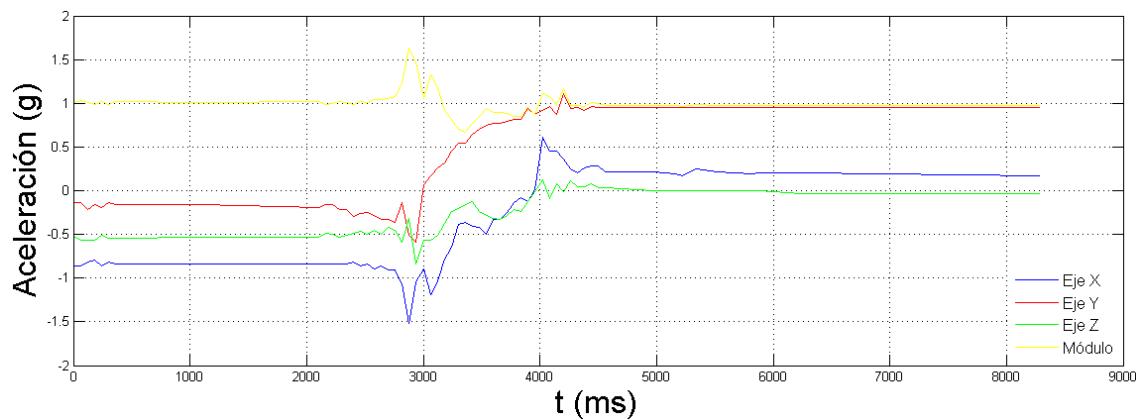


Figura 4.52. Datos obtenidos al incorporarse con el móvil en el bolsillo lateral.

La Figura 4.53, por su parte, muestra el proceso completo.

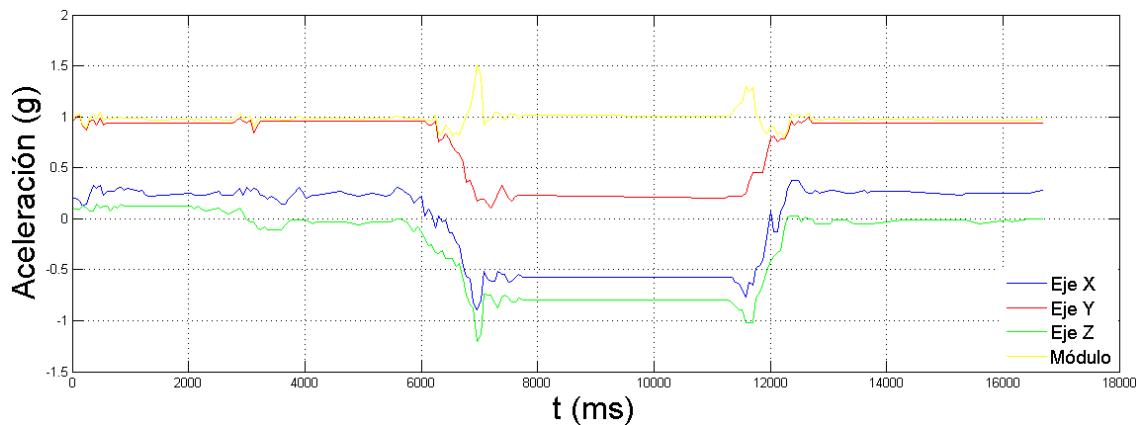


Figura 4.53. Datos obtenidos al sentarse e incorporarse a continuación.

Caminar

Caminar es, en muchos casos, la acción más continuamente realizada por una persona, aunque sea de la tercera edad. No sólo eso, sino que llevar el móvil en el bolsillo al hacerlo desemboca en unos valores de la aceleración relativamente bruscos.

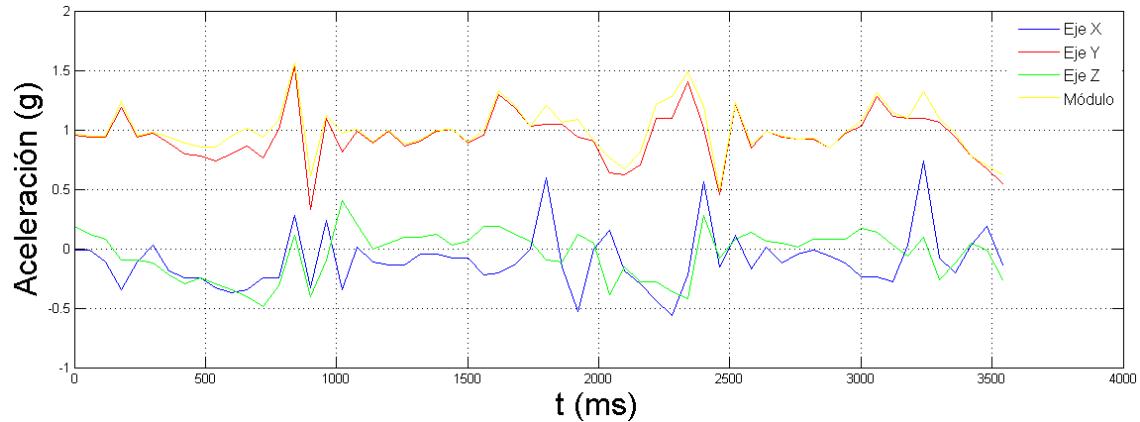


Figura 4.54.- Datos obtenidos al caminar con el smartphone en el bolsillo del pantalón.

La Figura 4.54 representa este proceso. Este registro se ha hecho caminando de forma normal, sin sacudidas ni acelerones. Una persona anciana, en principio, llevará un ritmo lento y pausado, por lo que caminar no debería producir falsos positivos. Se puede comprobar cómo el módulo se acota dentro de un intervalo relativamente comedido, por lo que un umbral más alto evitaría alertas innecesarias.

Trotar

Un trote suave constituye una particularidad del ejemplo anterior llevado al extremo.

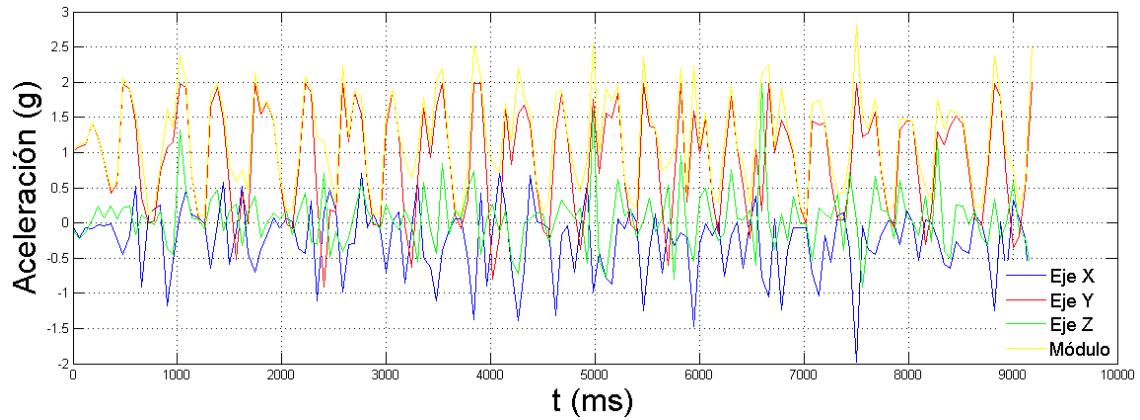


Figura 4.55. Datos obtenidos al trotar con el smartphone en el bolsillo del pantalón.

El patrón encontrado en este caso es similar, con la particularidad de que los valores son más bruscos debido al contoneo del smartphone en el bolsillo.

Giro de muñeca

Teniendo en cuenta que **el smartphone es un dispositivo que se maneja con las manos**, es interesante comprobar cómo responde este ante algunos movimientos concretos, además de los aleatorios mostrados en la Figura 4.50.

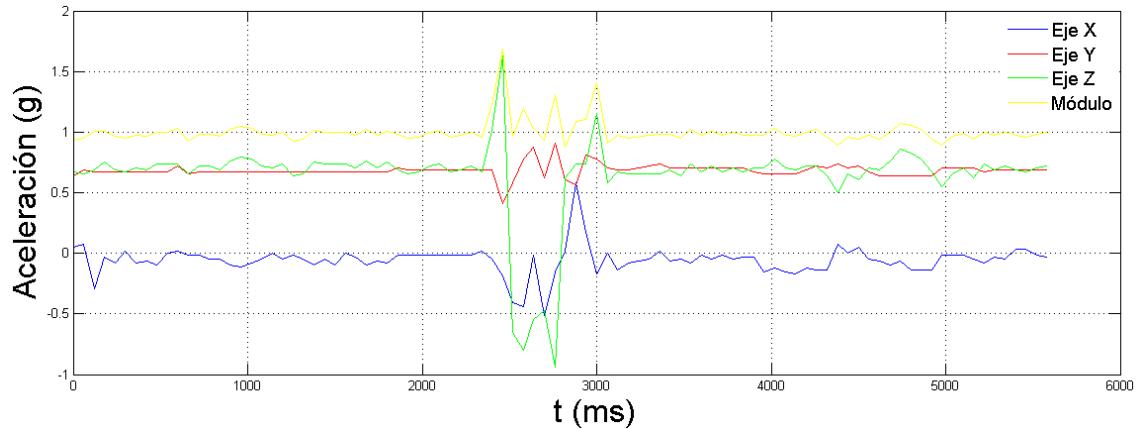


Figura 4.56. Datos obtenidos con un giro de muñeca.

En este caso, la gráfica responde a un giro de muñeca, con su consecuente repercusión en la aceleración por los cambios en las fuerzas. Como se ha comentado con anterioridad, el módulo es el valor más representativo, mostrando una perturbación al realizar este movimiento.

La Figura 4.57, por su parte, muestra un caso más extremo. Se trata, de nuevo, de un giro -esta vez más brusco- del smartphone, pasando de estar en horizontal a estar inclinado. Se puede comprobar como en este caso, el pico en el módulo es considerablemente superior a lo visto en otras pruebas. **Este pico es el que da a entender que ha habido un impacto**, aunque no haya existido ninguno en este caso: tal y como se ha explicado en la sección El acelerómetro, este determina la aceleración a partir de la fuerza a la que está sometido el móvil, que no tiene por qué ser necesariamente producida por una caída. Por lo tanto, y aunque ese cambio repentino puede ser un muy buen indicador de que está ocurriendo un evento que tal vez sea de interés, **por si mismo constituye un mecanismo demasiado simple como para ser fiable**.

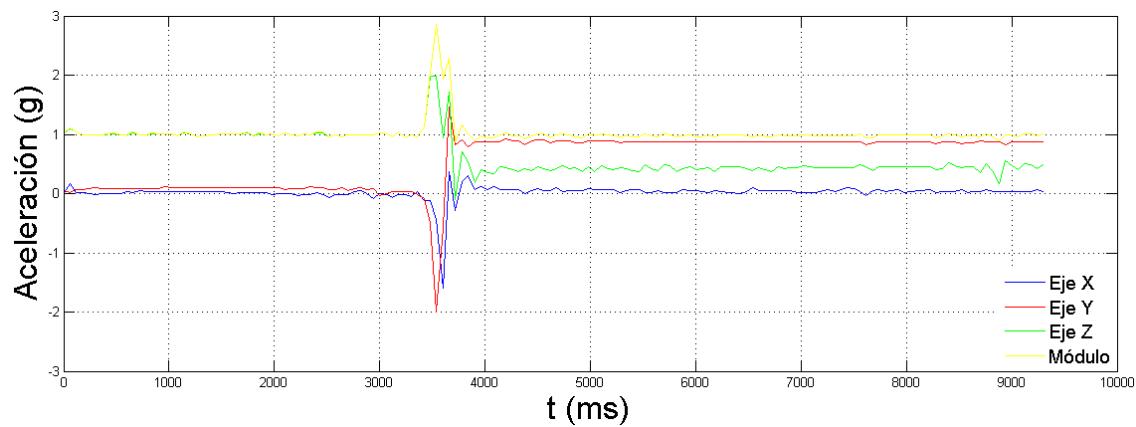


Figura 4.57. Datos obtenidos con un giro de muñeca brusco.

Esto da pie a la siguiente sección, en la que se describe el algoritmo de detección de caídas en su conjunto.

4.12.3. FUNDAMENTO DEL ALGORITMO

Teniendo en cuenta la **limitación de un pico en el módulo de la aceleración para determinar caídas**, fue necesario buscar alguna otra característica que, en su conjunto, permitiera determinar cuando tenía lugar este evento.

Existen numerosos estudios al respecto en Internet, como [Wearable device for real-time monitoring of human falls](#), o [Human Fall Detection Using 3-Axis Accelerometer](#). En algunos de ellos se distingue entre distintos tipos de caídas -hacia atrás, de lado, etc-, con análisis y funciones mucho más complejos que los que conciernen a este TFG, aunque es bastante común que **el algoritmo empleado no sea público**. En todo caso, dado el objetivo de FallApp, es necesario utilizar un mecanismo que, sin ser tan preciso, permita obtener cierto grado de fiabilidad.

Muchos de los estudios y proyectos examinados, como [esta tesis doctoral](#), coinciden en la **existencia de varias etapas en las caídas. Un patrón** que, aunque con la particularidad de que cada incidente de este tipo es único, se repite en todo el proceso:

- **Caída libre:** cuando el móvil está inmóvil encima de la mesa, o siendo sujeto en la mano, la aceleración que está sufriendo -como se puede comprobar en las gráficas expuestas- es 1g. Así ocurre con todos los objetos en reposo de la Tierra, sin contar aquellos que estén sometidos, además, a otras fuerzas; sin embargo, existen casos en los que se puede predecir, así como comprobar de forma práctica, que **la aceleración del smartphone tiende a 0g**: la caída del mismo.

En las situaciones comentadas, aunque la gravedad ejerce su fuerza sobre el smartphone -con sentido al centro de la Tierra- y, por tanto, a lo que tendería el móvil es a caer, este está apoyado sobre **una superficie que lo sostiene**. De esta forma, y como se explicó en el [apartado correspondiente](#), existe una fuerza opuesta que mantiene el objeto en su lugar.

Sin embargo, **cuando el móvil cae desde una cierta altura, deja de disponer de una superficie que lo sostenga**, tendiendo entonces a un valor 0g. ¿Cómo es esto posible, teniendo en cuenta que para dejar de sentir los efectos de la gravedad habría que hallarse a unos **6 millones de kilómetros de la Tierra**?

El efecto producido cuando el móvil cae al suelo se conoce como **caída libre**, haciendo referencia este término a que **el objeto sólo se ve afectado por la fuerza de la gravedad** -y ya no por la fuerza de la superficie-. Está muy **relacionado con el concepto de ingravidez** que se consigue, por ejemplo, en la ISS: a la altura a la que orbita (unos 400km) la gravedad es aproximadamente de 0.9g, y sin embargo se habla de que los tripulantes experimentan la gravedad cero. Lo que ocurre en realidad es la aplicación del efecto tratado; nave y pasajeros “caen” a la misma velocidad, por lo que estos últimos no cuentan con ninguna superficie que los sostenga. Otro ejemplo se encuentra en la caída de un ascensor con una persona dentro, donde este “no tiene” peso al desplazarse junto con el primero.

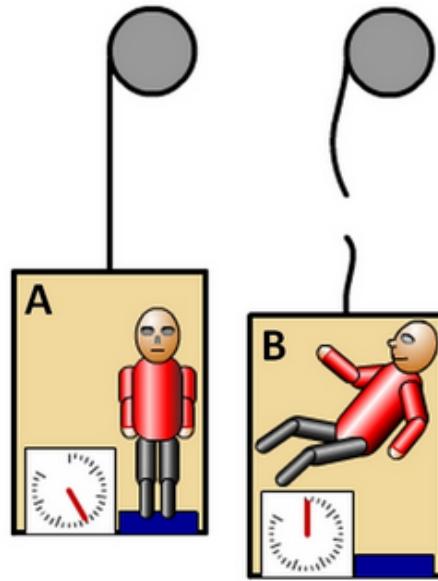


Figura 4.58.- Esquema de una persona con peso en un ascensor (izquierda) y en caída libre (derecha).

La etapa de la caída de una persona, sin embargo, no es ideal. **No existe, por norma general, un período de tiempo en el que el módulo sea 0**; lo que es más, normalmente no llega a alcanzar ese valor. Una caída del smartphone, por su parte, se puede acercar de forma considerable, siendo esta una diferencia fundamental en la detección de falsos positivos.

En la Figura 4.59 se pueden ver los datos obtenidos en una caída de un móvil al suelo desde una mesa.

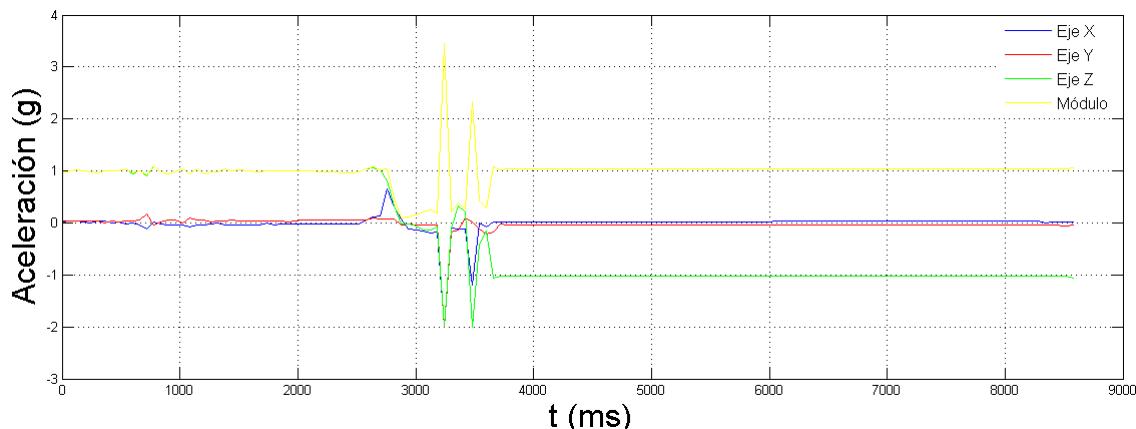


Figura 4.59. Caída de un smartphone al suelo desde una mesa.

- **Impacto:** el momento del impacto, con su consecuente repercusión en el módulo de la aceleración, es el **evento más determinante en todo el proceso**. Se corresponde, como su propio nombre indica, con el instante en el que la persona -el smartphone- golpea el suelo, sufriendo una alteración en las fuerzas que repercuten en este parámetro. En función de si tiene lugar o no, se analizarán más datos para llegar a tomar posteriormente una decisión.

- **Constancia de los valores:** después del impacto, tiene lugar una etapa en la que la persona permanece **inmóvil en el suelo**. Teniendo en cuenta que FallApp está orientada a personas de tercera edad, este tiempo puede llegar a ser bastante elevado, ya que este grupo de la población no suele disponer de la vitalidad, agilidad y energía como para reaccionar de forma inmediata.

Estas etapas se han corroborado con las pruebas empíricas realizadas con la aplicación de registro de valores, encontrando el mismo patrón en todas ellas. Algunas de las gráficas obtenidas se pueden ver a continuación:

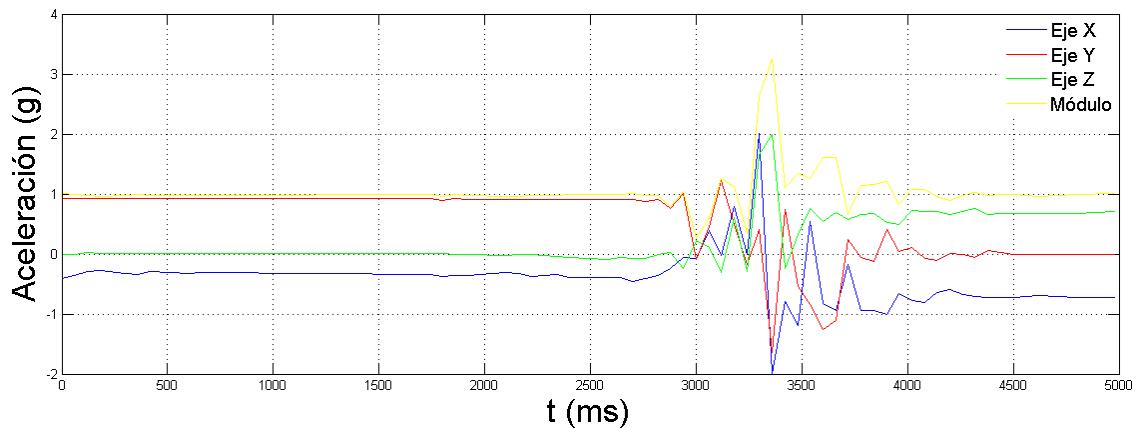


Figura 4.60. Caída hacia delante con el móvil en el bolsillo del pantalón.

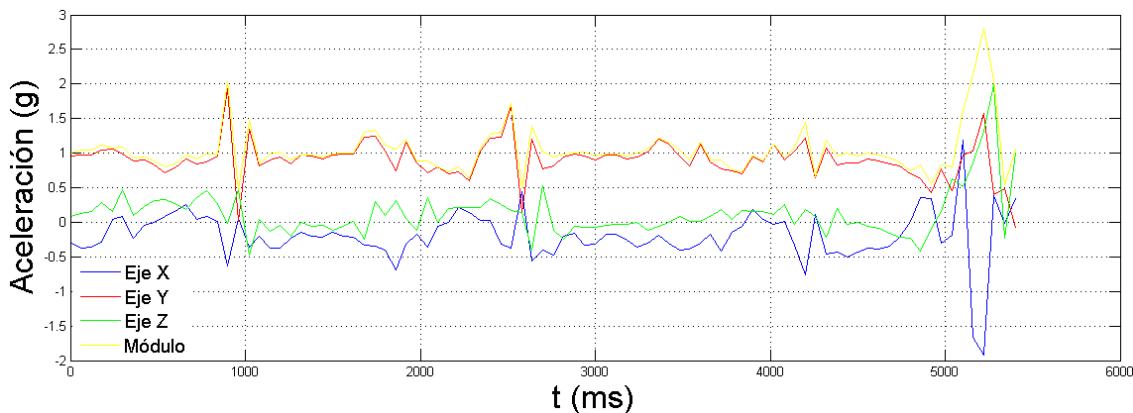


Figura 4.61. Datos obtenidos al caminar y caer hacia delante.

En la Figura 4.60 se puede observar una caída con sus correspondientes etapas: caída libre a los 3 segundos; impacto a los 3.4; y constancia a partir del 4.2, aproximadamente.

Por su parte, la 4.61 representa los valores obtenidos al caminar, y producirse una caída a partir de los 4500 ms. Se puede ver, así, como el pico producido por el incidente es **considerablemente superior al resto**.

Teniendo en cuenta las fases de la caída, actuando como un conjunto pueden servir como un buen indicador para detectar esta clase de incidentes. El desencadenante, como se ha comentado, es el impacto, el valor que sobrepase un determinado umbral definido. A continuación, se determinará si los valores recogidos por el acelerómetro son constantes, en cuyo caso se podrá tomar la decisión de enviar las alertas.

Uno de los mayores problemas de una aplicación de este ámbito es no sólo no enviar una alerta en caso de ser necesario, sino **enviar demasiadas**: los falsos positivos. Hay dos eventos importantes que presentan una similitud considerable y, por tanto, deberían ser diferenciados: la caída del smartphone, y la caída del usuario. Sus particularidades se pueden ver en la Tabla 9.

	Caída de smartphone	Caída de usuario
Caída libre	Más acusada, por norma general valores próximos a 0	Menos acusada, los valores no se aproximan tanto a 0
Impacto	Sí	Sí
Constancia	Sí, valores del eje Z próximos a 1g	Sí

Tabla 9.- Comparativa de características entre una caída del smartphone y de un usuario.

Una de las diferencias más destacables consiste en la última etapa. Las caídas del smartphone al suelo implican que, la inmensa mayoría de las ocasiones, **el dispositivo quede en horizontal sobre el suelo**, ya sea boca arriba o boca abajo. Este hecho supone que, además de permanecer el módulo constante, los valores correspondientes al eje Z son muy próximos a 1g, tal y como muestra la Figura 4.48. En lo que respecta a la caída de un usuario, **es complicado que el móvil quede totalmente horizontal después de un incidente**, por lo que esta diferenciación se utiliza para limitar los falsos positivos de este tipo.

El flujo de decisión del algoritmo se puede ver en el SDL mostrado en la Figura 4.62.

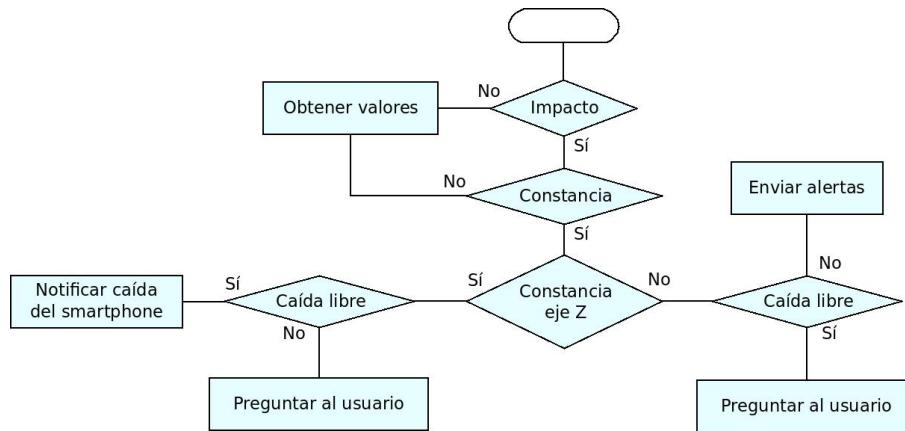


Figura 4.62. Flujo de decisión del algoritmo de detección de caídas.

4.12.4. IMPLEMENTACIÓN DEL ALGORITMO

El diagrama presente en la figura anterior responde al código programado en el servicio Android. Este es el componente de la aplicación que se ejecuta en segundo plano, donde se registra el *listener* y, por tanto, se pone en marcha la toma de datos.

Como se explicó en la sección [Toma de datos con el acelerómetro](#), el método *onSensorChanged* es llamado cada vez que se obtienen nuevos valores del sensor. En base a las pruebas realizadas, el instante de muestreo puede producirse entre unos 15 y unos

60 ms, produciéndose repetición de cifras en el primer caso. de todas formas, en el algoritmo no influye este hecho ya que lo importante, en todo caso, sería que el refresco se produjera con muy poca frecuencia; manteniendo este parámetro -nuevos valores cada 60 ms-, tener repeticiones es irrelevante.

El instante del impacto, como se ha comentado, es un evento que determina qué acción ejecutar. Eso implica que, **en función de ese parámetro, el análisis que hay que realizar cada período varía**. El mecanismo utilizado para conseguir esto de una forma elegante -siguiendo el Open/Closed Principle- es el **patrón estado**.

Java no permite definir una tabla hash en la que a cada clave le corresponda una función, por lo que se optó por hacer uso de una interfaz que posibilitara un resultado similar.

```

1 interface Action {
2     void runAction();
3 }
4
5 HashMap<String, Action> actionMap = new HashMap<String, Action>();

```

En función del estado, cuyo valor será la clave de la tabla hash, se ejecutará una función a otra, pasando por todos ellos si se detectan las condiciones de una caída. La Figura 4.63 ilustra este comportamiento por medio de una máquina de estados.

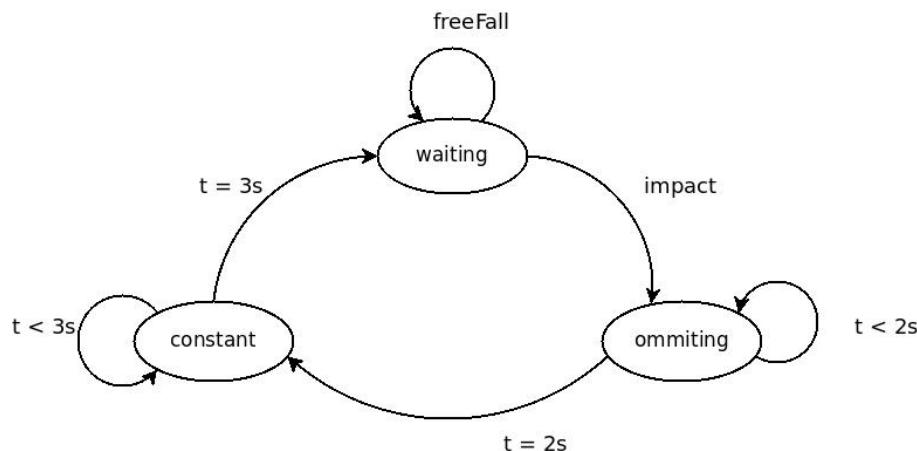


Figura 4.63. Máquina de estados del algoritmo de detección de caídas.

- **Estado “waiting”:** estado **por defecto**, en el que se valoran los datos obtenidos para comprobar si alguno se puede considerar como caída libre o como impacto. Tal y como se ha comentado en la [sección anterior](#), este último hecho es el **factor que desencadena el análisis posterior** y, por tanto, produce el cambio de estado.

Los datos se van procesando en tiempo real, lo que permite realizar algunas comprobaciones que permitan **saber si el smartphone se adecua a las condiciones de la aplicación**. Este, dentro de lo que cabe, **debería estar siempre en el bolsillo del usuario**, ya que en caso contrario no tendría sentido la monitorización. Por lo tanto, se comprueba si el móvil está en posición horizontal -en la mayoría de ocasiones, encima de una mesa-, notificando entonces un recordatorio del buen uso de FallApp.

- **Estado “omitting”:** el momento del impacto no es un sólo punto a lo largo de todo el muestreo, sino que consiste en un intervalo de tiempo en el que los datos son turbulentos (como se puede ver en las figuras del apartado anterior). Por lo tanto, **pasar a la siguiente etapa no es inmediato**, sino que es necesario esperar a que los valores vuelvan a ser fiables. Es necesario, de este modo, **configurar unos segundos en los cuales no se procesen ni se analicen**, siendo esta la labor del estado actual.

Para ello se utiliza un **temporizador**, proporcionado por la clase *CountDownTimer*. En el momento en el que cuente el tiempo definido, se volverán a analizar valores y se pasará al siguiente estado.

- **Estado “constant”:** el último estado consiste en el **análisis de la constancia de los datos**, determinando si el smartphone está inmóvil después del impacto. En el momento en que finaliza el estado “omitting”, se pone en marcha otro temporizador que abarca el período en el que interesa comprobar si los datos son constantes. Durante ese intervalo, se registran los mismos en un vector, analizándolos cuando haya pasado el tiempo estipulado.

Llegados a este punto, se dispone de la suficiente información como para determinar una acción a ejecutar como resultado. El ciclo de trabajo se puede ver en los SDL de las Figuras 4.64, 4.65, 4.66 y 4.67

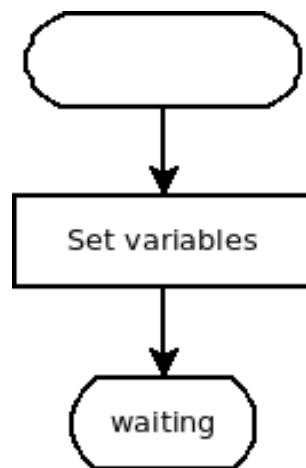


Figura 4.64. Comienzo del algoritmo de detección de caídas.

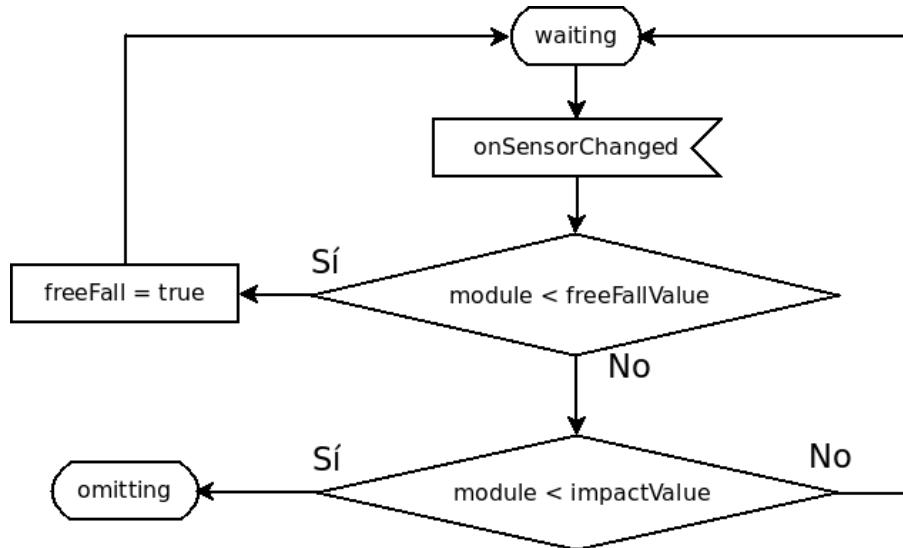


Figura 4.65. SDL del estado “waiting”.

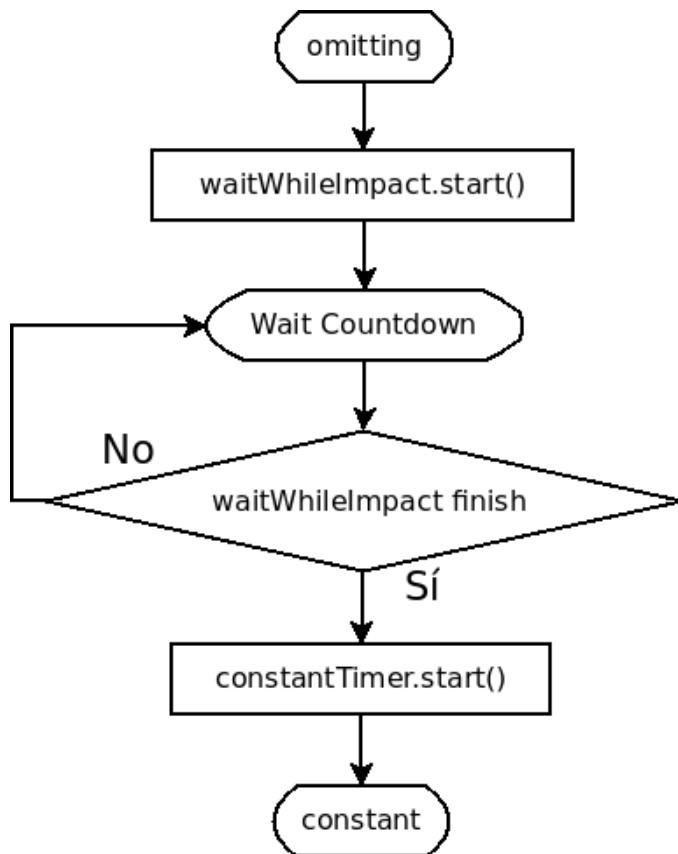


Figura 4.66. SDL del estado “omitting”.

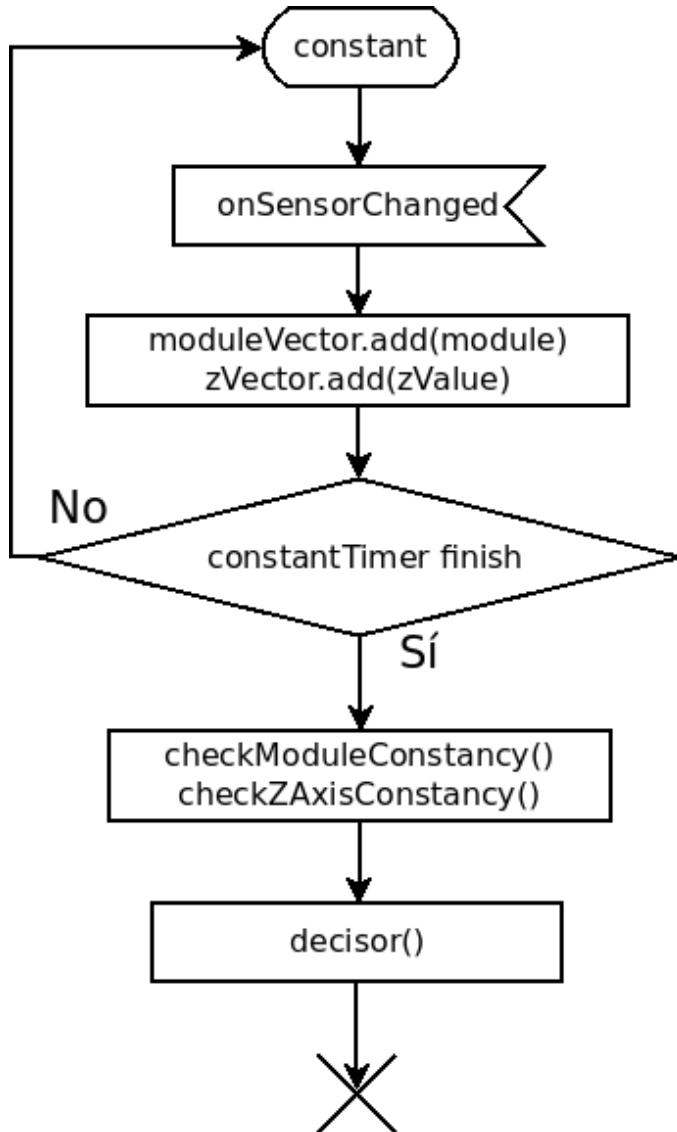


Figura 4.67. SDL del estado “constant”.

La elaboración de estos SDL resulta compleja, ya que engloba tanto el paso por los distintos estados como las temporizaciones de los timers; sin embargo, la clase *CountDownTimer* encapsula la creación de un nuevo hilo en segundo plano, lo que posibilita que la ejecución de la acción correspondiente al estado y del temporizador se realice de forma simultánea. De esta forma, es complicado plasmar esa simultaneidad en los diagramas, aunque se consideró más conveniente hacerlo de esta forma debido a la estrecha relación existente entre todos los procesos.

4.12.5. ALERTAS

En caso de que la aplicación tome la decisión de enviar alertas, se pondrá en marcha otro temporizador que, durante los 15 minutos siguientes, intentará enviar - cada 20 segundos- los SMS informando de la caída. La repetición periódica de esta tarea permite dar tiempo al servicio de localización para posicionar el

dispositivo, así como **minimizar las posibilidades de no enviar los avisos por no haber conexión en ese momento**.

Se enviarán, como máximo, **3 mensajes de texto**, en función de la cobertura móvil y de GPS disponible durante ese período de tiempo:

- **Mensaje informativo de la caída:** normalmente, el primer SMS que se envía. Contiene el **nombre del usuario** que ha sufrido el incidente y **la hora** a la que ha ocurrido el mismo.
- **Mensaje indicando la posición aproximada:** en caso de estar activado el servicio de localización, es habitual que la primera posición que se obtiene sea muy aproximada, con un margen de error bastante amplio. De todas formas, una caída se considera un accidente lo suficientemente importante como para asumir el gasto de un SMS extra informando sobre la **localización aproximada**, aunque sea solo orientativa.
Junto con la latitud y la longitud, se envía un **enlace de Google Maps** con el punto exacto indicado por el smartphone. Estos dos valores pueden no ser fácilmente interpretados, por lo que una **representación gráfica** podría resultar más conveniente.
- **Mensaje indicando la posición exacta:** cuando la **precisión de la localización sobrepasa un determinado umbral**, se envía el último SMS indicando los mismos parámetros que en el mensaje anterior.

Cabe destacar que la posición del usuario se puede obtener de **dos maneras distintas**: utilizando el **receptor GPS** del smartphone o la **red móvil**. El primer caso es el más recomendado, ya que es el que posibilita una localización más precisa. El segundo, por su parte, puede ser de utilidad en casos en los que la recepción GPS no sea buena, aunque su precisión sea muy inferior.

Teniendo en cuenta la dependencia de la cobertura para enviar las alertas, se pueden dar varios escenarios:

- **Hay cobertura móvil y el servicio de localización está activado:** se enviarán, en principio, los 3 mensajes de texto explicados anteriormente. En caso de **contar con la posición precisa antes de enviar el SMS con la aproximada**, se omitirá esta última alerta. De igual forma, si se dispone de **información de localización antes de enviar el SMS informativo**, se omitirá este último.
- **Hay cobertura móvil pero no está activado el servicio de localización:** en este caso, **se enviará únicamente el SMS informativo**, ya que no se podrá acceder a la posición del smartphone.
- **No hay cobertura móvil:** no se puede enviar ningún SMS sin cobertura, por lo que **se seguirá comprobando la conexión cada 20 segundos durante los próximos 15 minutos**.

En la Figura 4.68 se puede ver un SDL donde se aclara y representa de forma gráfica el proceso. Una vez se ha enviado un mensaje de cualquiera de los 3 tipos, se garantiza

mediante una variable *booleana* que no se reenvíe el mismo SMS. De esta forma, no se enviarán mensajes repetidos cada intervalo de tiempo.

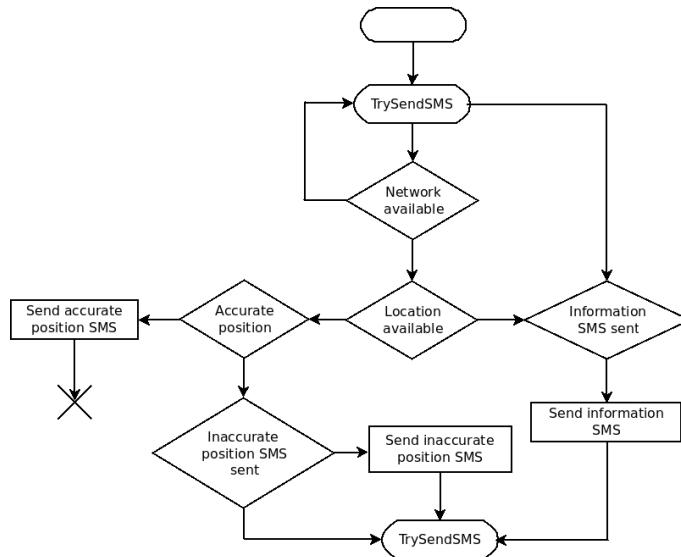


Figura 4.68. SDL del envío de alertas.

Cabe destacar que, en el momento de **detectar una caída**, el móvil comienza a **vibrar y a sonar** y no dejará de hacerlo hasta pasados 20 minutos, a menos que la alarma se detenga.

Para poder hacer uso de todos los mecanismos comentadas, como enviar SMS, vibrar o localizar el smartphone, es necesario definir los permisos correspondientes en el archivo “AndroidManifest.xml”. De este modo, al instalar la aplicación, el usuario será informado sobre **qué funciones de su smartphones requiere FallApp** para desempeñar su labor correctamente.

4.12.6. SERVICIO EN SEGUNDO PLANO

Como se ha comentado en numerosas ocasiones a lo largo de toda la memoria, el algoritmo de detección de caídas corre en segundo plano gracias a un **servicio**. De esta forma, aunque el usuario esté fuera de la aplicación o con el móvil bloqueado, se seguirán procesando los valores devueltos por el acelerómetro.

La forma de definir un elemento de este tipo en este caso es la siguiente:

```

1 public class FallService extends Service implements SensorEventListener {
2     ...
3 }
  
```

Se puede ver como extiende de la clase “Service”, a la vez que implementa la interfaz “SensorEventListener”. De este modo, **reúne las funcionalidades que proporcionan ambas**, permitiendo tomar datos de los sensores en *background*.

La clase “FallService” suele implementar varios métodos abstractos heredados de “Service”: *onCreate*, *onStartCommand* y *onDestroy*. Su ciclo de vida se puede revisar

en la sección [Servicios](#). Cada uno de ellos responde a los requisitos del algoritmo, siendo especialmente importante el segundo.

El método *onStartCommand* retorna un valor que define **cómo se comporta el servicio si es cerrado**. Aunque existen [varios disponibles](#), el más adecuado en este caso es *START-STICKY*.

De esta forma, si el servicio es destruido **no se guarda el último intent** recibido -que en este caso, no se necesita-, además de que el sistema podrá revivirlo si tiene memoria disponible.

4.13. COMUNICACIÓN ENTRE LA INTERFAZ WEB Y EL NÚCLEO ANDROID

Llegados a este punto, se han explicado tanto las **pautas seguidas en la elaboración de la interfaz** utilizando tecnologías web, como el **algoritmo de detección de caídas** programado en Java. Ambas partes, sin embargo, no tienen una comunicación inmediata entre ellas; si bien la transferencia de información entre clases Java es directa utilizando, por ejemplo, *Intents*, entre los ficheros HTML, CSS y JavaScript y las clases Java se requiere de **una serie de pasos intermedios**.

4.13.1. ACTIVIDADES Y SU CORRESPONDIENTE INTERFAZ

En una aplicación Android, una actividad consiste en una clase Java con su correspondiente *layout*. Este *layout* es un fichero XML que define la interfaz, conteniendo botones, cuadros de texto o cualquier tipo de elementos. De esta forma, desde este se puede llamar a métodos implementados en la clase, proporcionando Android mecanismos para permitir que el traspaso de información en una aplicación sea sencilla.

El mecanismo que utilizar Cordova, por su parte, funciona de forma diferente. Las clases Java no cuentan con su correspondiente *layout*, sino que lo que el usuario ve son la **conjunción de los archivos HTML, CSS y JavaScript**. La parte visible de la actividad reside en ellos, con una diferencia importante: mientras que en una aplicación nativa la funcionalidad la proporciona la clase, en este caso lo hace JavaScript.

	Aplicación nativas	Aplicación Cordova
Interfaz	Layout	Archivos web
Funcionalidad	Clase Java	JavaScript
Comunicación	Intents, Broadcasters	Plugins

Tabla 10. Diferencias entre las actividades en las aplicaciones nativas e híbridas.

Sin embargo, aunque no se necesite una clase Java para añadir la funcionalidad, **cualquier aplicación creada con esta plataforma cuenta con una**, por defecto nombrada “CordovaApp.java”. Su función es la de **lanzar la interfaz**, hacer la llamada al archivo HTML que corresponda. Por defecto, como se ha comentado en la sección [Config.xml](#), este archivo será el index.html.

En el caso de FallApp, la idea es **lanzar el asistente de configuración la primera vez que se ejecute la aplicación**, abriendo la página principal el resto de

ocasiones.

En el archivo “AndroidManifest.xml” se especifica **qué actividad se abre por defecto**, la que se conoce como *launcher activity*. Inicialmente y en el caso de Cordova, se trata de “CordovaApp.java”. Sin hacer ningún cambio al respecto, siempre se ejecutará esta clase y, por tanto, se lanzará el index.html. Teniendo en cuenta este desarrollo, este comportamiento no es viable. La solución consiste en el uso de las “**SharedPreferences**”, **un mecanismo de guardado permanente de datos** utilizado por Android.

Aunque en el caso de necesitar una base de datos más compleja es recomendado el uso de SQLite, las “SharedPreferences” constituyen una buena alternativa en caso de necesitar **guardar datos simples** de la aplicación. Por medio del sistema clave-valor, se pueden almacenar **variables que persistan aún habiendo eliminado la aplicación del segundo plano**.

Lanzar una actividad u otra, por tanto, consistiría en registrar una determinada cadena en función de cuál haya que abrir en cada momento. De esta forma, desde una sola clase Java es posible **llamar a diferentes archivos web** en función de las necesidades del momento. Se evita, por tanto, la creación de otras clases innecesarias, así como la modificación del “AndroidManifest.xml”

Una vez lanzada la interfaz, la interacción del usuario se recogerá por medio de eventos o funciones JavaScript, pudiendo así manipular el DOM. Sin embargo, manteniendo esta diferenciación en lo que a lenguajes se refiere, la funcionalidad se limita a modificar esa clase de aspectos, **sin poder interactuar dinámicamente con el código nativo**.

4.13.2. PLUGIN APIS

Como se explicó en la sección [Plugins, añadiendo funcionalidad](#), existen tres tipos de plugins en Cordova. En este apartado, resultan de interés aquellos que permiten acceder al código nativo del dispositivo.

Esta plataforma proporciona de forma oficial una serie de plugins que permiten acceder al hardware del smartphone, así como actuar de puente entre la interfaz web y el código nativo. Las posibilidades del software de Apache no se limitan a los disponibles en su página, sino que **se pueden desarrollar plugins específicos para cada aplicación**. De este modo, y al igual que los oficiales, constituyen el nexo entre los archivos web y las clases Java.

Esta herramienta de desarrollo de aplicaciones híbridas proporciona una abstracción que permite, utilizando **una función JavaScript, una clase y un método Java**, comunicar de forma bidireccional ambas partes. Una vez se realiza la llamada desde la interfaz, el código nativo responde por medio de una función de callback, ejecutando en la primera un método u otro en función de si la operación se completó con éxito o no.

La función JavaScript, cuyos parámetros se explican a continuación, se puede ver en la siguiente línea de código:

```
1 cordova
2 .exec(<successFunction>, <failFunction>, <service>, <action>, [<args>]);
```

- <successFunction>: función de callback que se ejecuta si la llamada al código nativo se realiza con éxito.
- <failFunction>: función de callback que se ejecuta si la llamada al código nativo no se completa con éxito.
- <service>: clase Java, que extiende de *CordovaPlugin*, con la que se pretende comunicar.
- <action>: cadena que sirve para identificar que acción o fragmento de código Java se quiere ejecutar. En la implementación de este mecanismo en FallApp, se verá cómo se llama a la misma clase desde distintas funciones JavaScript, por lo que este campo servirá para identificarlas y actuar en consecuencia.
- [<args>]: parámetro opcional, consiste en los argumentos que se envían al código nativo en formato JSON.

Por lo tanto, cuando desde JavaScript se ejecuta la llamada a `cordova.exec`, la interfaz se comunica con la clase Java cuyo nombre se especifica en el campo <*service*>. Las acciones a ejecutar en cada momento, por su parte, vendrían determinadas por el campo <*action*>.

En el caso de FallApp, se creó una clase llamada “LinkToService”, cuya labor es procesar todas las llamadas al código nativo que se hagan desde cualquier archivo JavaScript. En función de cuál sea el valor del campo *action*, se actuará de una manera u otra.

Dichas llamadas, detalladas en función del *action*, se muestran a continuación:

- **startService**: llamada realizada al finalizar el asistente de configuración. Obtiene las variables que requiere el servicio, lo inicia y lanza la página principal.

Además, en este proceso se le muestra al usuario una **ventana emergente** en caso de no tener activadas todas las modalidades del servicio de localización.



Figura 4.69. Ventana emergente para arrancar el servicio de localización.

- **getVariables:** transmite los parámetros introducidos por el usuario a la página principal. Así, se pueden **representar el nombre y los números de contacto en la cabecera del menú lateral.**



Figura 4.70. Parámetros mostrados en la cabecera del menú lateral.

- **updateSensibility:** permite **actualizar el valor de la sensibilidad** apretando el botón correspondiente en la página principal.
- **updateTelNumbers:** misma acción con los números de contacto.
- **stopService:** **parar el servicio** desde la opción de la página principal.
- **reStartService:** **arrancar de nuevo el servicio** desde la página principal.
- **sendAlert:** permite **llamar al método de envío de alertas** dentro del servicio. Se hace esta llamada al pulsar el botón correspondiente en la pantalla principal.
- **returnToMainPage:** posibilita **volver a la pantalla principal** después de haber sido lanzada la vista de alertas.

4.14. DIAGRAMAS

Tras explicar la funcionalidad de FallApp y cuál es su comportamiento, se consideró adecuado elaborar diagramas de distinto tipo que los describan de forma gráfica.

En primer lugar, se muestran diagramas de caso de uso de la aplicación. En ellos se detalla la interacción del usuario con los distintos componentes de la misma, como con la página principal o el servicio de detección de caídas.

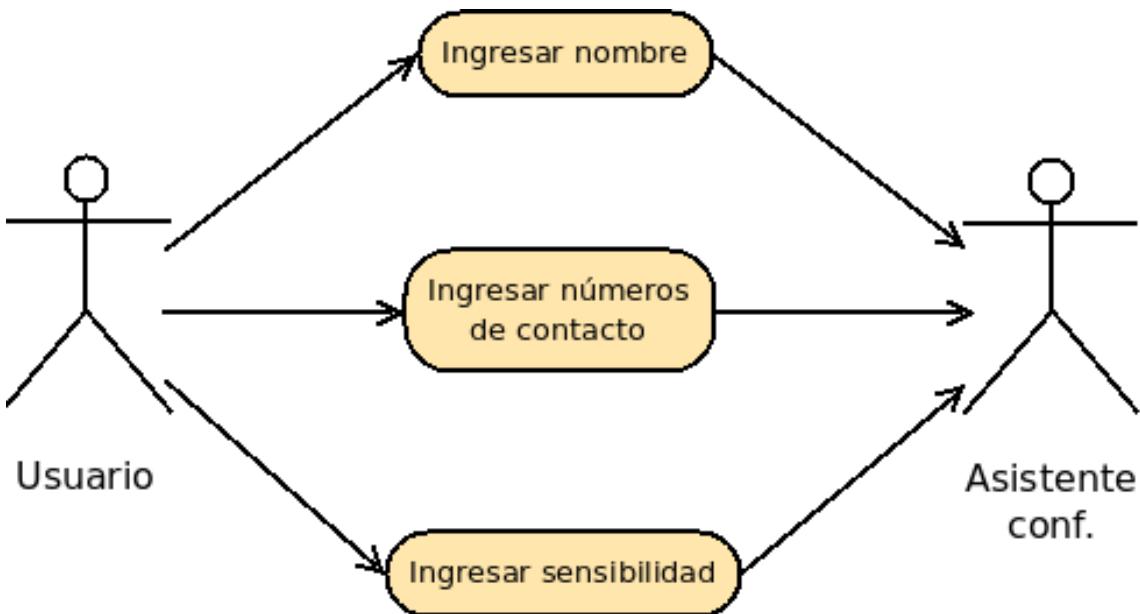


Figura 4.71. Caso de uso del asistente de configuración.

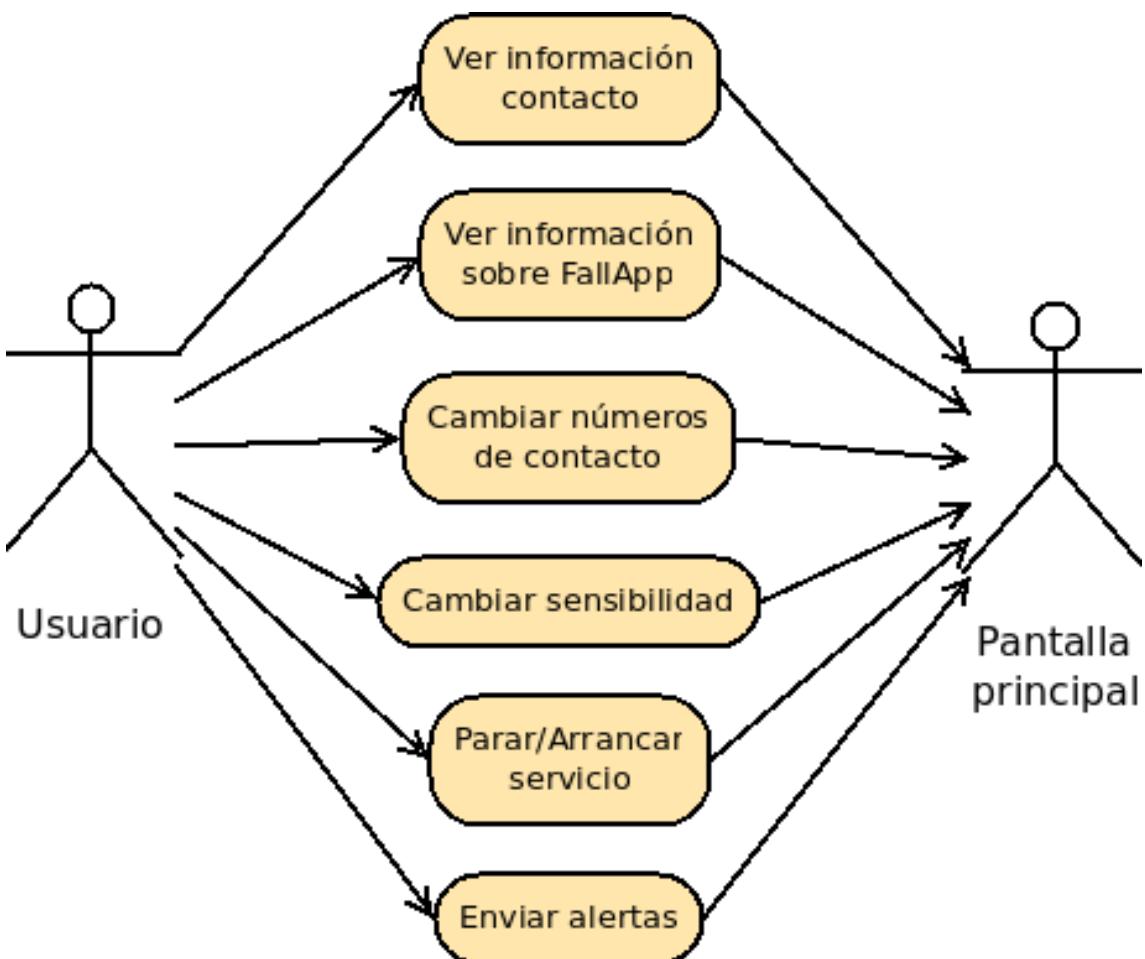


Figura 4.72. Caso de uso de la pantalla principal.

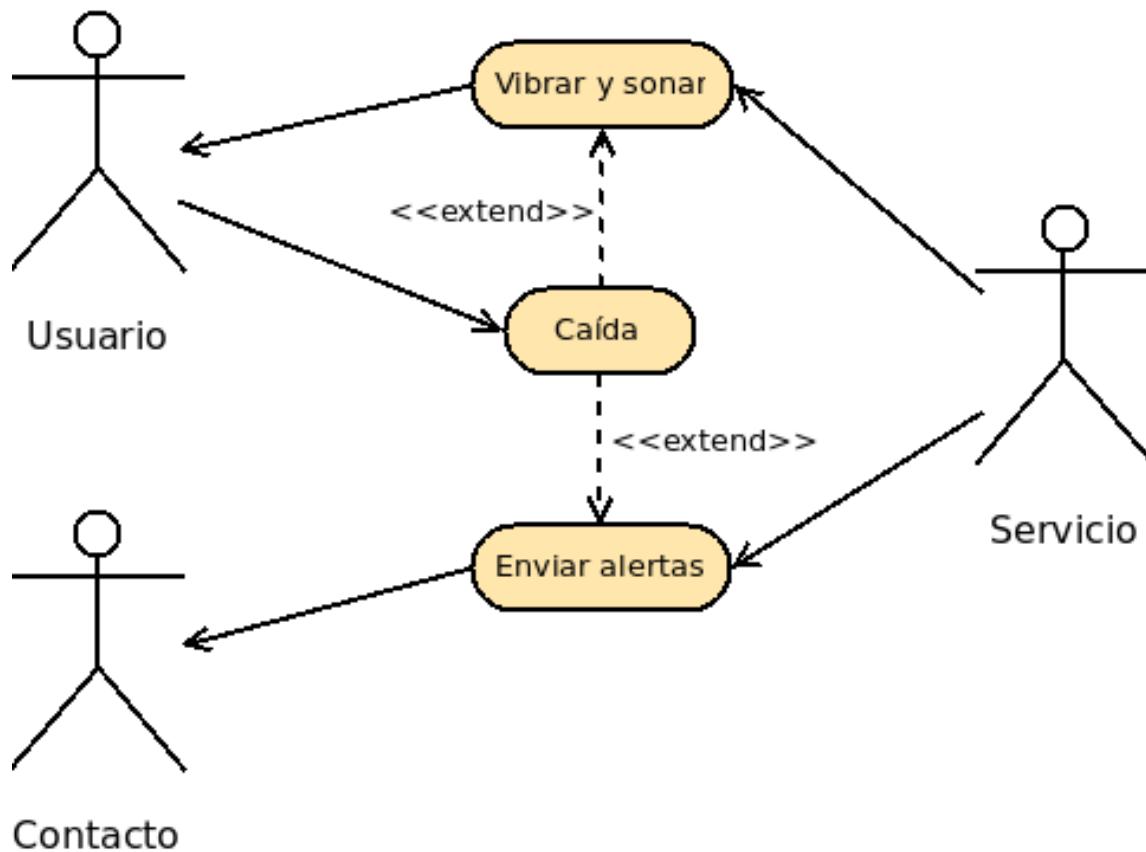


Figura 4.73. Caso de uso de una caída.

Los diagramas de clases UML representan las clases disponibles y su interacción. En la Figura 4.74 se puede ver el relativo al núcleo de FallApp, las clases Java con código nativo que la componen.

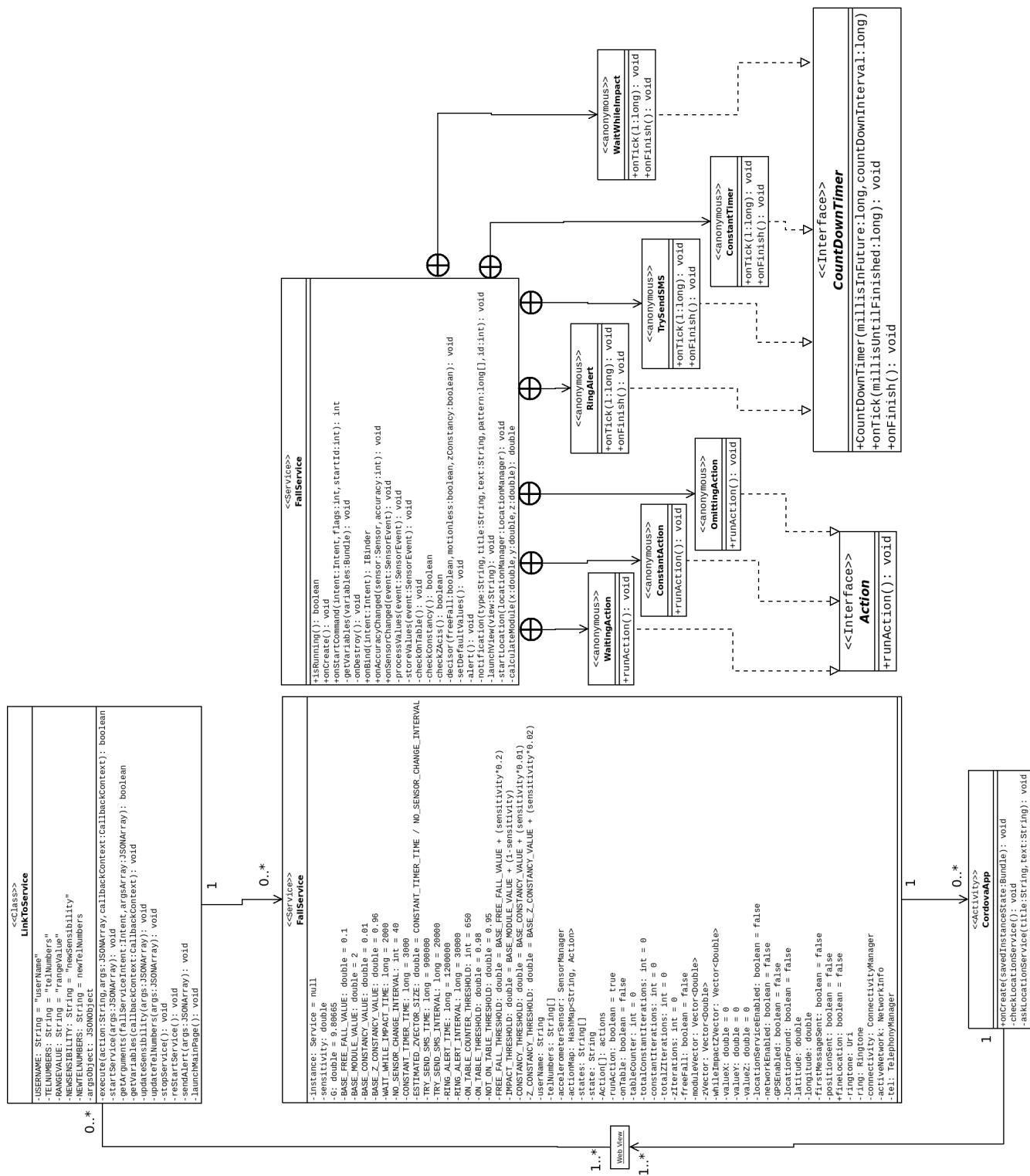


Figura 4.74. Diagrama de clases UDL de FallApp.

La interfaz web se ha representado como un objeto, ya que se consideró conveniente detallar la interacción de la misma con el resto del clases Java. Sin embargo, y teniendo en cuenta que esta consiste en un conjunto de páginas web dinámicas, se optó por hacer uso de las posibilidades que brindan los diagramas para representar tanto su estructura, como el dinamismo que proporciona JavaScript.

En este último caso, se utilizó el mismo método que el visto en la Figura 4.74: los diagramas de clases UDL. De esta forma, para cada archivo .js, es posible indicar los distintos objetos creados, así como las variables y métodos de los que disponen. Aunque este lenguaje no responde exactamente a la filosofía orientada a objetos, elaborar estos esquemas supone una forma rápida y directa de comprobar qué forma la interfaz web.

Se expone, en primer lugar, todo lo relativo al asistente de configuración, continuando con la página principal y la vista de alertas.

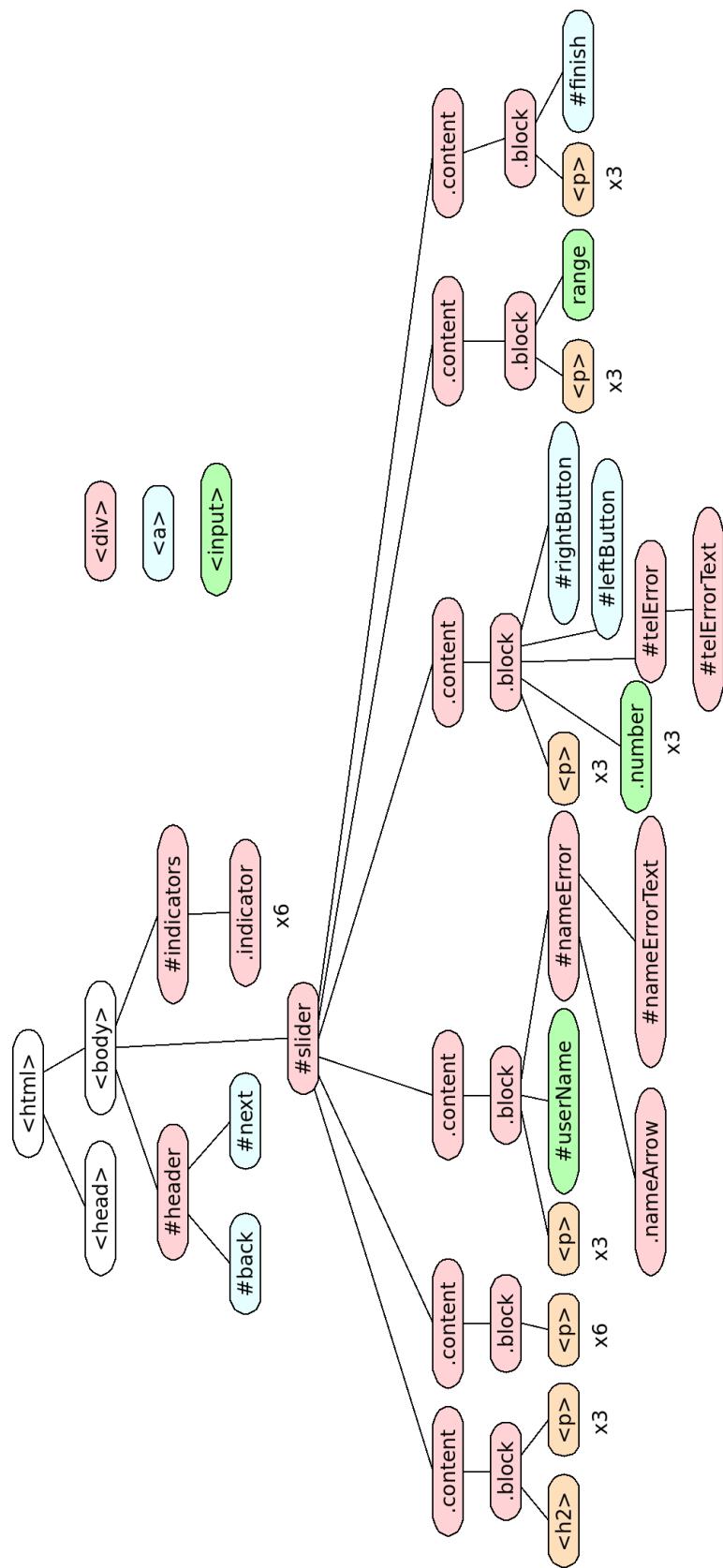


Figura 4.75. Estructura web del asistente de configuración.

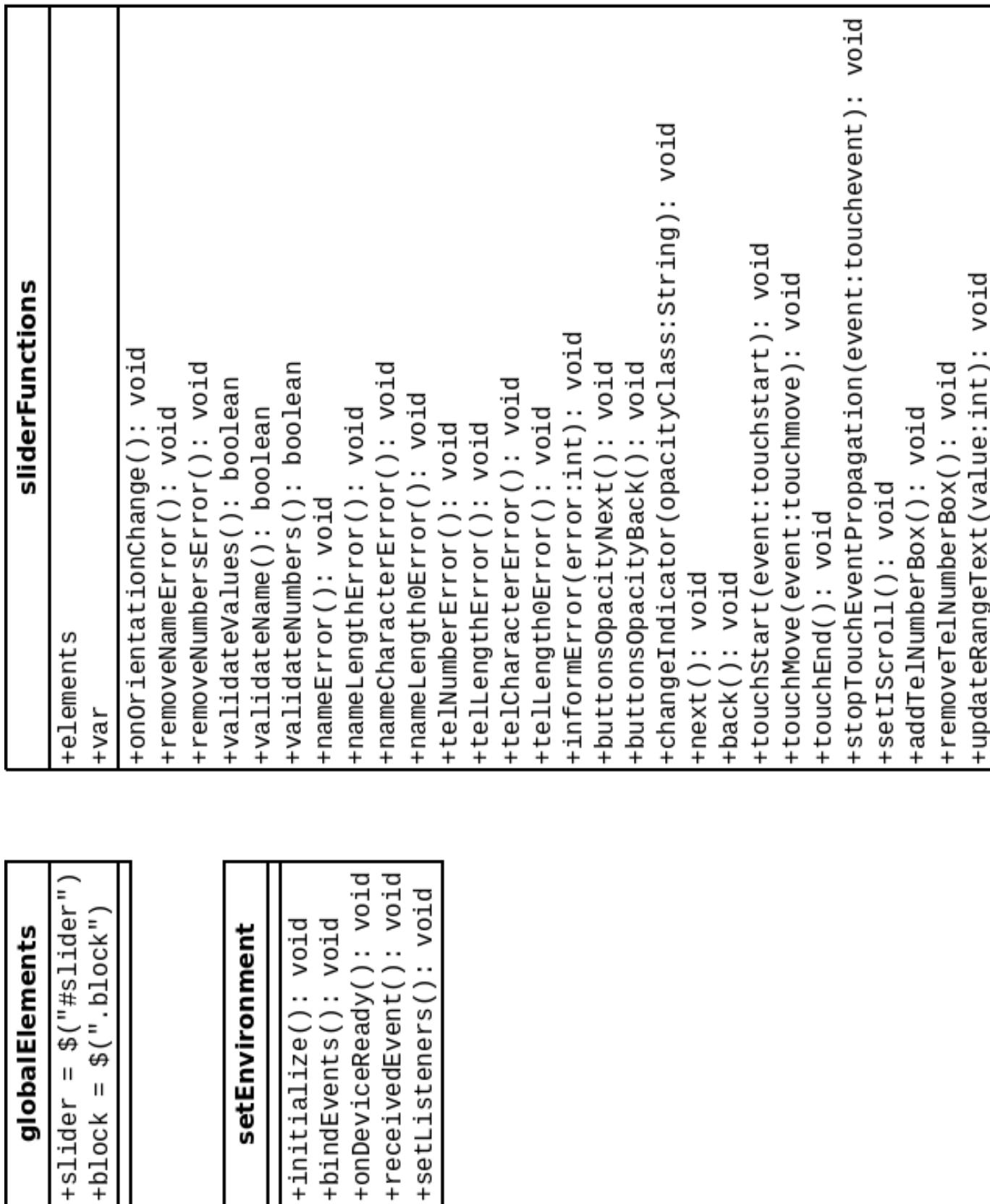


Figura 4.76. Estructura JavaScript del asistente de configuración.

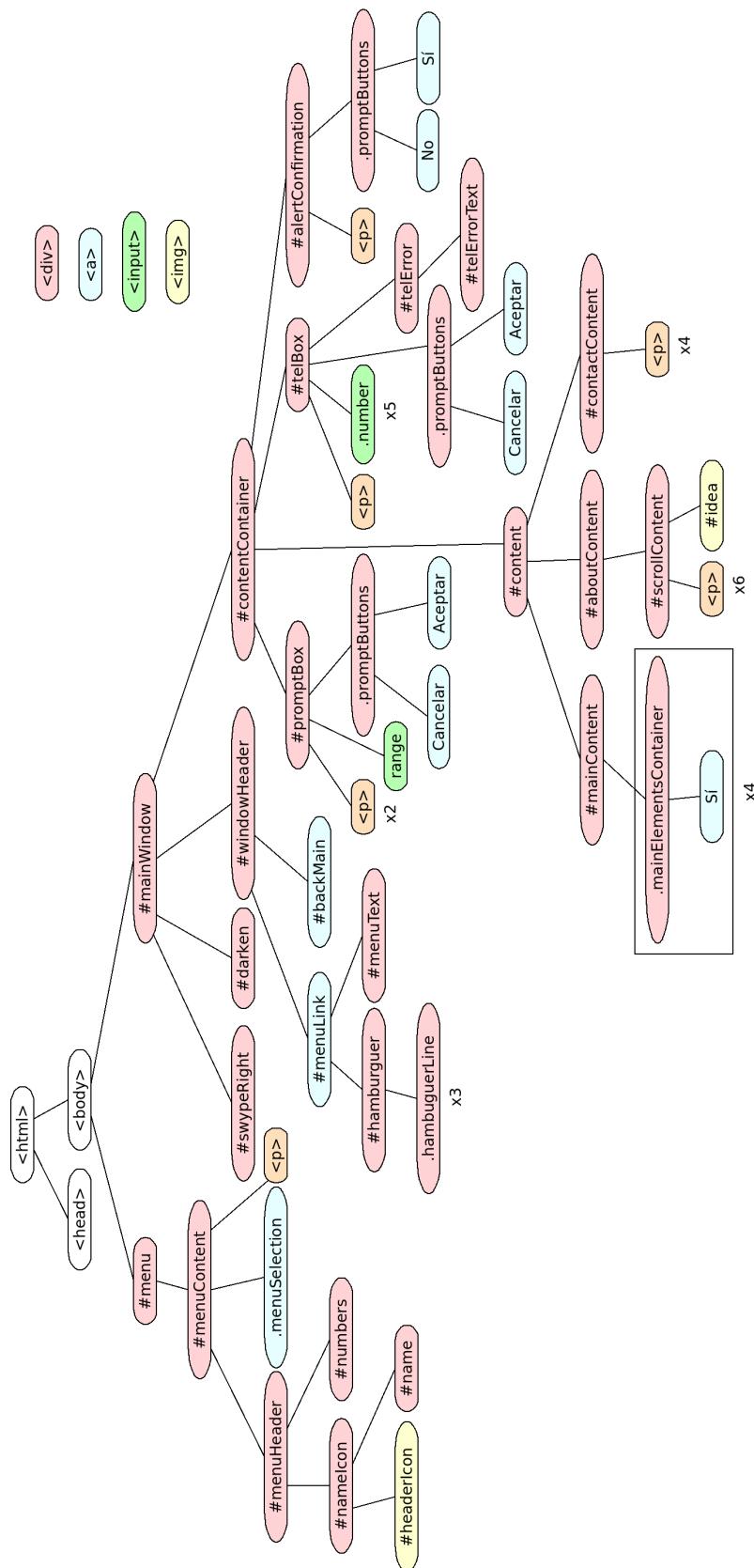


Figura 4.77. Estructura web de la página principal.

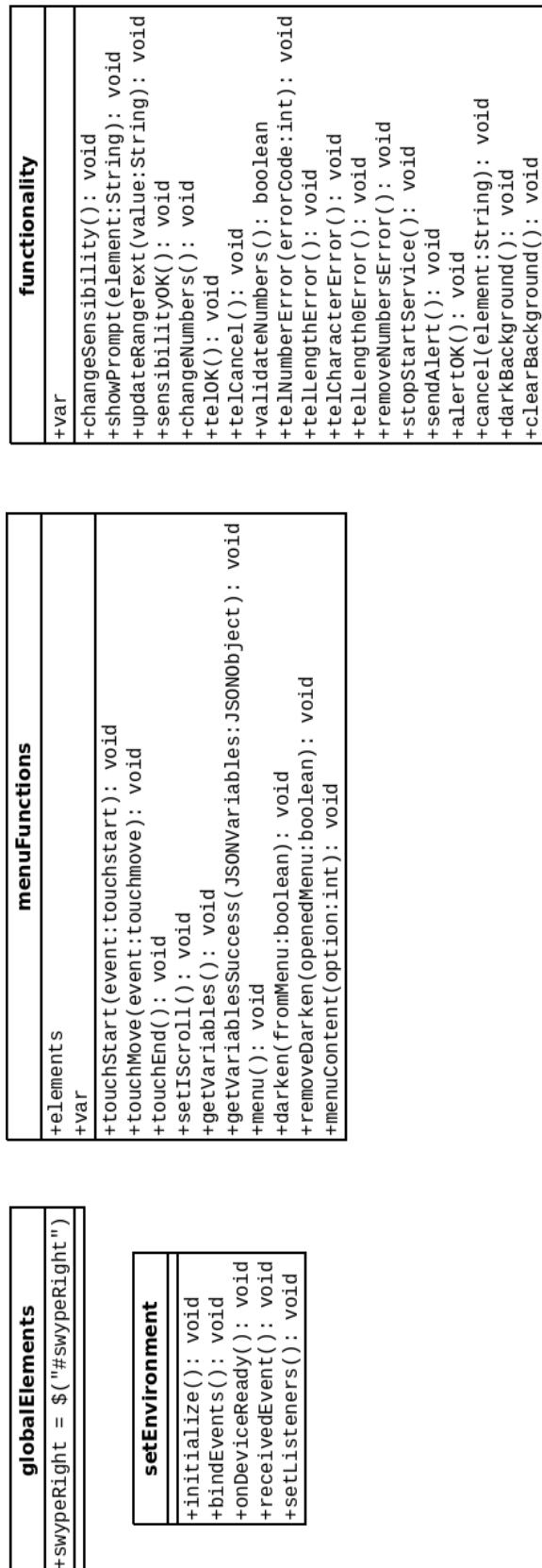


Figura 4.78. Estructura JavaScript de la página principal.

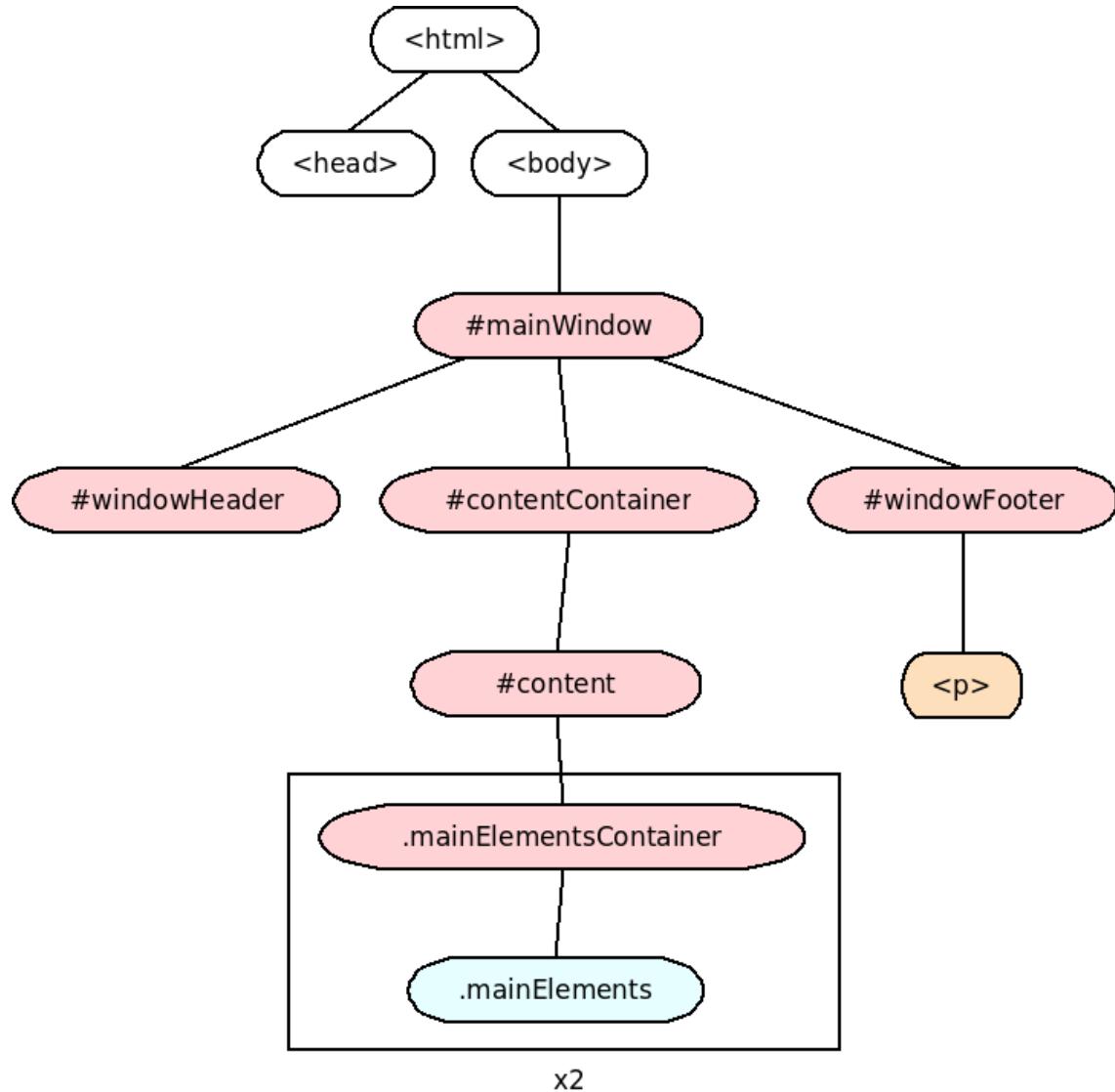


Figura 4.79. Estructura web de la vista de alertas.

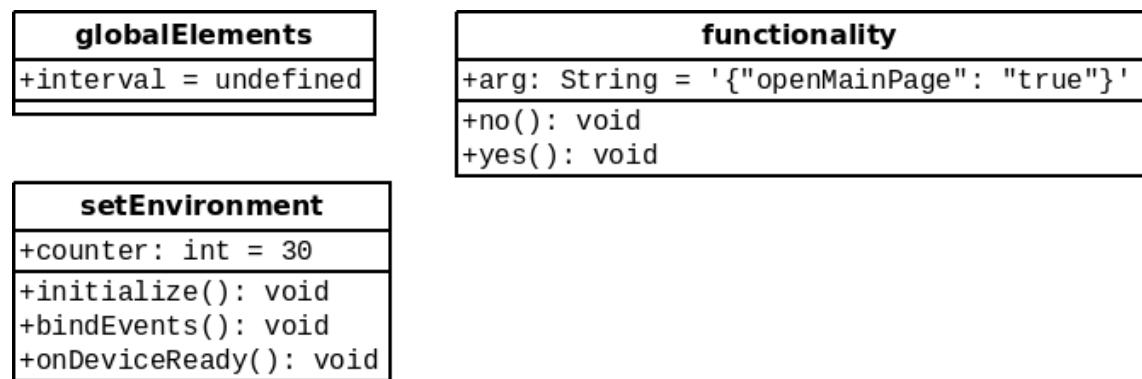


Figura 4.80. Estructura JavaScript de la vista de alertas.

4.15. POSIBLES MEJORAS

Aunque FallApp es una aplicación funcional, existe una gran cantidad de posibilidades que pueden **mejorar** tanto sus opciones, como su interfaz. En este apartado se tratarán algunos aspectos que, por falta de tiempo, **no han podido ser implementados** e incrementarían la calidad de la aplicación.

- **Añadir un *splashscreen*:** un aspecto puramente estético, Cordova permite añadir un splashscreen de una forma sencilla. Hacer uso del mismo podría mejorar la primera impresión al abrir FallApp por primera vez.
- **Interfaz para el modo apaisado:** la interfaz web elaborada en esta aplicación, si bien funciona y responde en cualquier orientación del dispositivo, **pierde mucha calidad con el móvil en horizontal**.

La única funcionalidad añadida al respecto es la transición del slider al girar el smartphone en el asistente de configuración. Por el desplazamiento en píxeles que va sufriendo este elemento al avanzar por las pantallas, cambiar entre los dos modos implicaba un descoloque de las mismas. Por lo tanto, haciendo uso del evento “orientationchange” se corrige el posicionamiento.

Sin embargo, una buena opción habría sido **reestructurar cómo se muestran los distintos bloques del .html**. La alternativa ideal habría sido intentar utilizar la regla `@media`, que permite definir distintos parámetros y propiedades en función de aspectos como el ancho o el alto de la ventana.

- **Añadir la opción de enviar mensajes de Whatsapp:** aunque se ha intentado implementar, parece ser que no se puede mandar un mensaje de Whatsapp de forma automática a un contacto. Sí es posible abrir la conversación con el mismo, pero esa funcionalidad no resulta de utilidad para enviar alertas.

Por lo tanto, este sería un buen añadido, de haber dispuesto de más tiempo para investigar al respecto y buscar una buena solución.

- **Añadir la opción de regular el tiempo de espera antes de enviar una alerta:** en la vista de alertas, se espera siempre un tiempo de 30 segundos antes de enviarlas. Sería interesante que el usuario pudiera **especificar cuánto quiere que dure este intervalo**, adecuándolo así a su estado físico y sus necesidades.
- **Añadir números desde los contactos:** actualmente, FallApp permite añadir números de teléfono a mano. Sería interesante que se llevara al usuario a una pantalla de contactos, donde pudiera escogerlos de forma más cómoda.
- **Mejorar las ventanas emergentes de la pantalla principal:** aunque cumplen su cometido, el pop-up que se abre al intentar modificar los números de contacto no permite ni añadirlos ni quitarlos. Esta opción habría sido posible, ya que no había más que copiar la implementación existente en el asistente de configuración; sin embargo, **estéticamente el resultado no era bueno**.

Al ser una ventana emergente y, por tanto, no ocupar toda pantalla, se encontraron problemas con la asignación del espacio al añadir cajas para introducir los nuevos teléfonos. Se podría haber optado por ocupar la pantalla completa, pero

no se quiso “romper” el estilo de los pop-ups. De este modo, esta mejora tiene mucho potencial y no conllevaría demasiado esfuerzo.

- **Almacenar datos:** FallApp desecha los valores del acelerómetro a medida que los va analizando -excepto en determinadas situaciones puntuales-. Podría ser interesante **almacenarlos utilizando SQLite**, de forma que se puedan consultar en un futuro para, por ejemplo, **comprobar tendencias o elaborar estadísticas**.
- **Elaborar gráficas:** los datos almacenados también serían una buena fuente para la **representación de datos**. Combinar esta opción con la anterior, con el objetivo de observar tendencias o comprobar qué se ha hecho durante el día, es una idea con potencial. Además, se podrían **revisar las caídas o los falsos positivos** con el fin de mejorar el algoritmo.
- **Utilizar broadcasters:** aunque este es un aspecto en el que no se ha profundizado, según la información buscada por internet los *broadcasters* podrían ser un buen mecanismo para posibilitar la comunicación entre clases. En lugar de la manera expuesta, en este caso se conseguiría -en principio- mayor optimización y simplicidad en la misma.
- **Usar redes neuronales:** las redes neuronales son un mecanismo que se originó a mediados del siglo XX. Simulan el comportamiento del cerebro humano, siendo un sistema capaz de aprender. De esta forma, podrían recoger datos a partir del uso de la aplicación que haga el usuario, mejorando a lo largo del tiempo y afinando la detección de caídas, minimizando el número de falsos positivos.

Aunque el objetivo no sería que el anciano se dejase caer para que FallApp aprenda qué implica ese acto, se podría **combinar con el algoritmo diseñado**. Así, inicialmente sería este último el que se encargue de detectar incidentes, mejorando indefinidamente gracias al complemento de las redes neuronales.

5. Pruebas de funcionamiento

En este apartado se detallará **cómo se comporta la aplicación ante determinadas situaciones**. Servirá para comprobar la fiabilidad del algoritmo, así como para corroborar el correcto funcionamiento de FallApp.

Cabe destacar que, en las pruebas que requieren tirar o dejar caer el smartphone, se ha situado una manta en el suelo con el fin de que este no sufra daños. Este hecho puede amortiguar el impacto y hacer que los valores devueltos por el acelerómetro sean, por tanto, menores.

5.1. CAIDA DEL SMARTPHONE DESDE UNA MESA

- **Descripción:** el smartphone caerá al suelo desde una mesa, con una altura de 75cm.
- **Salida esperada:** la aplicación debería detectar una caída sólo del móvil y mostrar una notificación indicándolo.
- **Salida obtenida:** tras dejar caer el móvil en repetidas ocasiones, se ha detectado una caída en la mayoría de ellas. Los errores en esta prueba vienen dados, en su práctica totalidad, por el giro del smartphone al caer.

Si el móvil cae girando sobre sí mismo, el acelerómetro registra valores que hacen que **no se cumpla el requisito de caída libre**. Como se comentó en la sección [Uso de sensores](#), este inconveniente podría ser solucionado por el giroscopio -si bien no se sabe con exactitud al no haber hecho el estudio pertinente-.

	Valores esperados	Valores obtenidos
freeFall	true	true
impact	true	true
constancy	true	true
zConstancy	true	true

Tabla 11. Comparación de resultados al dejar caer el smartphone desde una mesa.

5.2. CAÍDA DEL SMARTPHONE AL SACARLO DEL BOLSILLO

- **Descripción:** el smartphone caerá al suelo al sacarlo del bolsillo, desde una altura de aproximadamente 1m.
- **Salida esperada:** la aplicación debería detectar una caída sólo del móvil y mostrar una notificación indicándolo.
- **Salida obtenida:** tras dejar caer el móvil en repetidas ocasiones, se ha determinado en todas ellas una caída del smartphone, si bien es cierto que depende en gran medida de cómo se produzca esta.

	Valores esperados	Valores obtenidos
freeFall	true	true
impact	true	true
constancy	true	true
zConstancy	true	true

Tabla 12. Comparación de resultados al dejar caer el smartphone desde el bolsillo.

5.3. CAÍDA DEL SMARTPHONE GIRANDO SOBRE SÍ MISMO

- **Descripción:** se dejará caer el smartphone al suelo, haciendo que gire sobre sí mismo en lugar de caer desplazándose a lo largo de un sólo eje. La altura será de aproximadamente 1m.
- **Salida esperada:** la aplicación tendrá problemas para determinar el instante de caída libre debido a las rotaciones del móvil. Por lo tanto, se darán falsos positivos que derivarán en el envío -mostrando previamente la vista de alertas- de los mensajes.
- **Salida obtenida:** tras dejar caer el móvil 10 veces, en 3 se determinó caída libre y, por tanto, se detectó la caída del smartphone. Las 7 restantes, el campo correspondiente a la trayectoria hasta el suelo no tuvo un valor correcto; de esta forma, se lanzó la vista de alertas al decidir la aplicación en función de las variables de la Tabla 13.

	Valores esperados	Valores obtenidos
freeFall	false	false
impact	true	true
constancy	true	true
zConstancy	true	true

Tabla 13.- Comparación de resultados al dejar caer el smartphone rotando sobre sí mismo.

5.4. CAMINAR

- **Descripción:** se caminará a un ritmo normal con el smartphone en el bolsillo.
- **Salida esperada:** el móvil en ningún momento obtendrá valores constantes del acelerómetro. De este modo, aunque se detecten impactos -producidos por el movimiento rítmico al desplazarse- no se enviará ninguna alerta. En este caso concreto, es complicado que se sobrepase su umbral de detección.
- **Salida obtenida:** tras caminar varios minutos con el dispositivo en el bolsillo derecho del pantalón, en ningún momento se obtuvieron picos que sobrepasaran el umbral de detección de impactos. Por lo tanto, ni siquiera fue necesario comprobar la constancia posterior de los datos.

5.5. TROTAR

- **Descripción:** se correrá a un ritmo lento con el smartphone en el bolsillo. Esta prueba resulta interesante para comprobar cómo se comporta la aplicación ante algunos casos de uso “extremos” en la vida del anciano.
- **Salida esperada:** al igual que en el caso anterior, no se deberían determinar valores constantes, evitando así el envío de alertas.
- **Salida obtenida:** FallApp detectó caídas libres e impactos debido al movimiento; sin embargo, el análisis posterior de los datos determinó que el móvil no estaba inmóvil, por lo que el comportamiento fue el esperado.

5.6. SENTARSE EN UNA SILLA

- **Descripción:** el hecho de sentarse produce una repercusión en el módulo de la aceleración que se puede considerar un impacto. Aunque el umbral se debería rebasar sólo en casos mucho más bruscos, como dejarse caer sobre el asiento en lugar de sentarse, un falso positivo en este caso podría derivar en el envío de alertas, por lo que se consideró una buena prueba.
- **Salida esperada:** no se debería sobrepasar el umbral de impacto, por lo que ni siquiera se tendría que llegar a etapas posteriores del análisis.
- **Salida obtenida:** no se detectaron falsos positivos.

	Valores esperados	Valores obtenidos
freeFall	-	-
impact	false	false
constancy	-	-
zConstancy	-	-

Tabla 14. Comparación de resultados al sentarse en una silla.

5.7. LEVANTARSE DE UNA SILLA

- **Descripción:** acción muy parecida a sentarse, con una descripción análoga.
- **Salida esperada:** no se debería sobrepasar el umbral de impacto, por lo que ni siquiera se tendría que llegar a etapas posteriores del análisis.
- **Salida obtenida:** no se detectaron falsos positivos.

	Valores esperados	Valores obtenidos
freeFall	-	-
impact	false	false
constancy	-	-
zConstancy	-	-

Tabla 15. Comparación de resultados al levantarse de una silla.

5.8. LLEVAR EL MÓVIL A LA OREJA

- **Descripción:** teniendo en cuenta que el smartphone se usa con las manos, se consideró interesante realizar alguna prueba de una acción cotidiana realizada con el mismo. Por lo tanto, se realizará en repetidas ocasiones el movimiento de llevar el móvil a la oreja.
- **Salida esperada:** no se debería sobrepasar el umbral de impacto, por lo que ni siquiera se tendría que llegar a etapas posteriores del análisis. Aún así, podría darse el caso si se realiza el gesto muy rápidamente.
- **Salida obtenida:** en las pruebas “normales” no se detectaron falsos positivos, determinando un impacto cuándo se mueve muy rápido el dispositivo. De todas formas, el análisis posterior de constancia no determinó ausencia de movimiento debido a los ligeros temblores de la mano. Aunque podría darse el caso, se considera una opción muy poco común.

	Valores esperados	Valores obtenidos
freeFall	-	-
impact	false	false
constancy	false	false
zConstancy	false	false

Tabla 16. Comparación de resultados al llevar el móvil a la oreja.

5.9. CAMINAR Y CAER

- **Descripción:** una de las pruebas más importantes, donde el usuario camina y se cae accidentalmente hacia delante. Otras caídas hacia atrás o hacia los lados se pueden englobar en esta prueba, al ser análogos los valores devueltos por el acelerómetro.
- **Salida esperada:** todo el algoritmo se ha diseñado con el fin de evitar malas interpretaciones en este caso, por lo que se debería enviar una alerta aunque sea después de mostrar la vista de alertas.
- **Salida obtenida:** se ha detectado la caída en gran parte de las ocasiones, probando también casos donde esta no fuera ideal.

Así, se intentó caer de la forma más “fluida” posible, intentando simular la caída libre. Detectando ese parámetro, se lanzó la vista de alertas y se envió la alerta pasados 30 segundos.

Lo mismo ocurre en el caso de que, al caer, el smartphone quede casualmente totalmente plano sobre el suelo, detectando constancia en una superficie horizontal.

	Valores esperados	Valores obtenidos
freeFall	false	false
impact	true	true
constancy	true	true
zConstancy	false	false

Tabla 17. Comparación de resultados al caminar y caerse hacia delante.

6. Planificación

Esta sección recoge un diagrama de Gantt, donde se representa de forma gráfica la planificación y las tareas en las que consiste el presente TFG. Cabe destacar que tanto estas, como los plazos, corresponden a un proyecto **desde el punto de vista de una empresa**, no de un alumno de Grado.

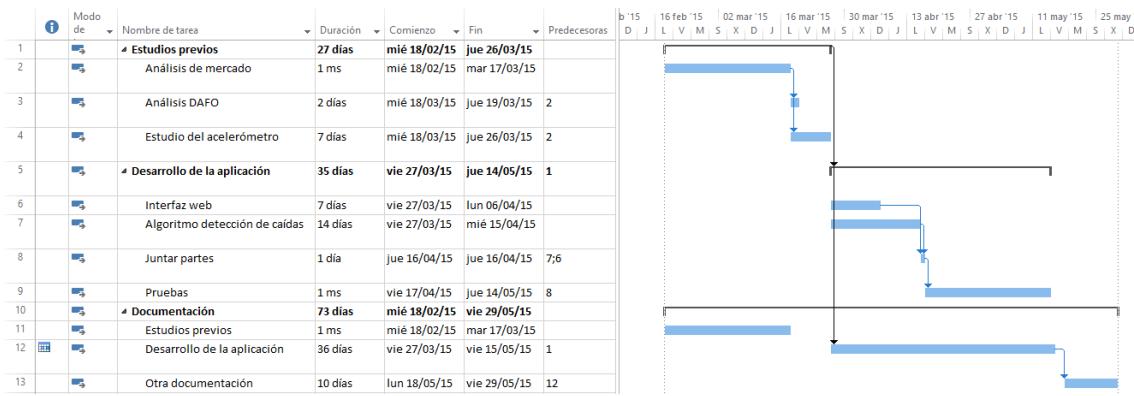


Figura 6.81. Diagrama de Gantt del desarrollo de FallApp.

7. Presupuesto

En este apartado, se expone un posible presupuesto aproximado del desarrollo de FallApp. Al igual que en la sección anterior, corresponde a un proyecto de empresa.

Concepto	Unidades	Precio unitario (€/h)	Duración (h)	Total
Desarrollador web	1	15	56	840
Ingeniero software (Android)	1	35	112	3920
Subtotal personal de desarrollo				4760
IntelliJ IDEA	2	481.58	-	481.58
Distribución Linux	2	0	-	0
Subtotal licencias de software				481.58
Ordenador portátil	2	331.95	-	663.90
Smartphone	2	269	-	538.00
Subtotal hardware				1200.90
				Subtotal
				6443.48
Beneficio (25 %)				1610.86
IVA (21 %)				1353.10
TOTAL				9407.44

Tabla 18. Presupuesto del desarrollo de FallApp.

8. Conclusiones

Cordova ha resultado ser una herramienta muy interesante para el desarrollo de aplicaciones móviles. Permite abstraer al desarrollador, permitiéndole programar usando lenguajes web y encapsulando los archivos en una aplicación Android. De esta forma, se pueden conseguir diseños sencillos y funcionales en -en principio- poco tiempo.

Sin embargo, tras completar el TFG y después de haber experimentado tanto con esta plataforma como con Android, no se considera la opción más adecuada. Es de suponer, no obstante, que los programadores web la valorarán sobremanera, ya que les permitiría aplicar sus conocimientos sin necesidad de tocar una sola línea de código nativo.

Los inconvenientes encontrados a día de hoy consisten en dos puntos bien diferenciados:

- **Interfaz dificultosa y poco profesional:** en este caso concreto, **no se disponía de conocimientos previos en lo que respecta a los lenguajes web**. Por lo tanto, fue necesario aprender no sólo cómo funciona una aplicación Android, sino las pautas, buenas prácticas y metodología propia de HTML, CSS y JavaScript. Esto requirió una **inversión de tiempo inicial** considerable, ya que se pretendió diseñar una interfaz que proporcionara la **mejor experiencia de usuario posible**. La estructura en sí puede no resultar complicada, pero los detalles son los que marcan la diferencia; de esta forma, se intentaron perfeccionar aspectos como el oscurecimiento de la página principal al abrir el menú, o el uso de los eventos de toque para conseguir el desplazamiento entre pantallas. Es decir, se trató de que **FallApp se pareciera lo máximo posible a una aplicación nativa**.

La pregunta en este caso es: **¿si la intención es que parezca a una aplicación nativa, por qué no desarrollarla directamente como tal?** A lo largo de este proyecto se han tenido que crear otras aplicaciones, como la que guardaba los valores del acelerómetro en un archivo txt. En ellas, la interfaz se elaboró en un breve período de tiempo, consiguiendo resultados relativamente profesionales; los botones y los cuadros de texto, por ejemplo, se adecuaban automáticamente a la interfaz del smartphone, tal y como se pude ver en la Figura 4.47. Sin embargo, en una interfaz web **todo debe ser programado desde cero**, definiendo las dimensiones de los elementos, el color de fondo, el grosor de los bordes, etc. Aunque la personalización es total, el tiempo empleado en conseguir unos resultados bonitos es mucho mayor.

Además, y esto es algo muy subjetivo ya que depende sobremanera del gusto y de la capacidad de diseño, **el resultado final no es todo lo bueno que podría haber sido**. Se ha conseguido que FallApp sea una aplicación relativamente bonita, pero **el aspecto no es muy profesional**. No sólo eso, sino que presenta algunos aspectos incoherentes, como el texto *placeholder* no centrado verticalmente en el cuadro de introducción del nombre. En el *layout* de una aplicación nativa, sin embargo, se consiguió el objetivo de una forma mucho más sencilla y satisfactoria. Se nota cuándo una plataforma está perfectamente integrada, siendo este uno de esos casos. Si bien muchos aspectos de la interfaz no se han

intentado conseguir por este medio (como el menú lateral y el fondo oscurecido), es muy probable que sea mucho más fácil lograr mejores efectos empleando menos recursos. De todas formas, dado que no se ha realizado una comparativa, es sólo una opinión poco fundamentada (aunque seguramente no esté desencaminada).

Otro aspecto desfavorable es el rendimiento. La red está plagada de artículos discutiendo el tema, pero en función de la experiencia con FallApp, se puede asegurar que este parámetro es **claramente inferior al nativo**. Aún habiendo aplicado todas las buenas prácticas explicadas, se percibe *lag* en las transiciones en determinados casos. Este inconveniente sería admisible, personalmente, en móviles de gama baja con poca potencia; sin embargo, se han realizado pruebas con **distintos smartphones de gama alta** con idénticos resultados, por lo que se puede deducir que **no es un problema de potencia, sino de optimización**.

El funcionamiento, además, ha presentado **incoherencias puntuales**, como la necesidad de pulsar 2 veces en algún botón para que este lo detectara. La experiencia web no ha sido del todo mala, pero sí inferior a la nativa.

- **Comunicación entre elementos de la aplicación:** como se ha explicado en su [correspondiente apartado](#), la transferencia de datos entre la interfaz web y el núcleo Java no es inmediata. La comunicación mediante intents y broadcasters sigue siendo viable entre las distintas clases, pero entre JavaScript y el código nativo es necesario utilizar plugins.

En la propia página de Apache Cordova existen tutoriales que guían a lo largo del proceso, por lo que este en sí no resulta complicado. Sin embargo, se convierte en un procedimiento muy tedioso, que implica utilizar demasiado código para lograr objetivos muy simples: la llamada al método necesario desde el archivo .js, las funciones de *callback* correspondientes, la clase Java, la acción y los parámetros en formato JSON. Esto desemboca en que **la compartición de variables entre componentes implica la existencia de numerosos pasos intermedios**, en lugar de añadirse directamente como parámetros en los intents.

En aplicaciones que no dispongan de un servicio, o que no exijan demasiada comunicación entre clases, este mecanismo puede no constituir un inconveniente. En FallApp, no obstante, supuso **demasiado código y demasiadas complicaciones innecesarias**. Como se ha comentado en el punto anterior, en lo que respecta a este aspecto también habría sido mejor crear una aplicación nativa.

La conclusión extraída del trabajo con Cordova es que, en general, constituye una **idea y unas posibilidades extraordinarias**. No obstante, y es una opinión subjetiva, **de momento no está al nivel de una aplicación nativa**: a lo largo del desarrollo se produjeron fallos en principio inexplicables, que se solucionaban con, por ejemplo, una reinstalación. Este hecho hace que la plataforma pierda fiabilidad, además de tiempo que debería emplearse en mejorar la aplicación, no en solucionar errores incoherentes.

Como se ha explicado en la sección [Aplicaciones web, híbridas y nativas](#), sin embargo, cuenta con otras ventajas interesantes con respecto a las últimas. Su uso, en general, puede resultar muy satisfactorio si el objetivo es un **diseño simple y una funcionalidad sencilla**, más aún si se **aprovechan conocimientos web de un desarrollador**.

A modo de resumen, se puede destacar que las aplicaciones híbridas todavía tienen camino por delante, presentando algunas incoherencias que ensombrecen la experiencia. Cuentan, aún así, con un futuro prometedor, sobre todo si se puede **conseguir que la estabilidad y fiabilidad de Cordova se equipare a la experiencia al programar con el lenguaje nativo.**

A pesar de todas las dificultades comentadas, **la elección de utilizar esta plataforma no pudo haber sido mejor.** En lugar de aprender un sólo método de programación de aplicaciones, se han aprendido tres. No sólo eso, sino que los conocimientos en los ámbitos de las tecnologías web son muy útiles a día de hoy, y no se habrían adquirido de no haber optado por esta alternativa.

En lo que respecta al servicio, la experiencia ha sido menos enriquecedora. No porque fuera menos interesante, sino porque el lenguaje Java era ya conocido.

El uso de sensores y la captación de sus datos, por su parte, constituyó una parte crucial en el TFG: fue necesario analizarlos, y comparar los valores obtenidos con distintos móviles, para **encontrar un buen algoritmo de detección de caídas.** No sólo eso, sino que dada la función del acelerómetro, resultó imprescindible repasar conceptos de física con el objetivo de entender exactamente cómo se comporta.

Para acabar, y como conclusión general, el uso de Cordova y la creación de una aplicación híbrida fueron muy adecuados desde el punto de vista didáctico. No obstante, no lo fueron desde el punto de vista comercial, derivando en demasiadas complicaciones innecesarias. Teniendo en cuenta la orientación de este proyecto, por tanto, se puede afirmar que **la elección resultó más que apropiada.**

Referencias

- [1] "PhoneGap o Apache Cordova", [en línea]. Disponible en la web: <http://www.campusmpv.es/recursos/post/PhoneGap-o-Apache-Cordova-que-diferencia-> (último acceso febrero 2015)
- [2] "Página oficial", [en línea]. Disponible en la web: http://cordova.apache.org/docs/en/4.0.0/guide_overview_index.md.html#Overview (último acceso junio 2015)
- [3] "Apache Cordova, una interfaz para dominarlos a todos", [en línea]. Disponible en la web: <http://qbitera.com/apache-cordova-una-interfaz-para-dominarlos-a-todos/> (último acceso febrero 2015)
- [4] "Vídeo fundador PhoneGapSpain", [en línea]. Disponible en la web: <https://www.youtube.com/watch?v=AC1f5mgPwfQ> (último abril 2015)
- [5] "Evento deviceready", [en línea]. Disponible en la web: http://cordova.apache.org/docs/en/4.0.0/cordova_events_events.md.html#deviceready (último acceso junio 2015)
- [6] "Tutorial: How To Write a PhoneGap plugin for Android", [en línea]. Disponible en la web: (último acceso 2015)
- [7] "http://devgirl.org/2013/07/17/tutorial-how-to-write-a-phonegap-plugin-for-android/", [en línea]. Disponible en la web: (último acceso junio 2015)
- [8] "Cómo hacer un menú estilo Facebook", [en línea]. Disponible en la web: <http://www.phonegapspain.com/tutorial/como-hacer-un-menu-estilo-facebook/> (último acceso mayo 2015)
- [9] "FastClick, cómo acelerar la navegación de tu app", [en línea]. Disponible en la web: <http://www.phonegapspain.com/tutorial/fast-click-como-acelerar-la-navegacion> (último acceso mayo 2015)
- [10] Tirado Núñez, María jesus. "Aplicación para detección de caídas", [en línea]. Disponible en la web: <http://www.giro.infor.uva.es/proyectos/IG-SEP09-18.pdf> (último acceso junio 2015)
- [11] Blanco, Ramiro; Hoyos, Alejandra. "Sistema de detección de caída en personas de la tercera edad para uso en centros geriátricos", [en línea]. Disponible en la web: <http://repository.javeriana.edu.co/bitstream/10554/7041/1/tesis486.pdf> (último acceso junio 2015)
- [12] "ABC del acelerómetro", [en línea]. Disponible en la web: <http://5hertz.com/tutoriales/?p=228> (último acceso junio 2015)

- [13] Varios autores. “Wearable device for real-time monitoring of human falls”, [en línea]. Disponible en la web: http://www.researchgate.net/profile/Chi-Shih_Chao/publication/222579089_Wearable/ (último acceso junio 2015)
- [14] “Human Fall Detection Using 3-Axis Accelerometer”, [en línea]. Disponible en la web: (último acceso junio 2015)
- [15] “Así funcionan las tripas de tu móvil: el acelerómetro”, [en línea]. Disponible en la web: <http://www.eldiario.es/hojaderouter/tecnologia/acelerometro-funciones-giroscopio.html> (último acceso junio 2015)
- [16] “¿Cómo funciona el acelerómetro de nuestro móvil?”, [en línea]. Disponible en la web: (último acceso junio 2015)
- [17] “Accelerometers”, [en línea]. Disponible en la web: <http://www.explainthatstuff.com/accelerometers.html> (último acceso junio 2015)
- [18] “La diferencia entre giroscopio y acelerómetro”, [en línea]. Disponible en la web: (último acceso junio 2015)
- [19] “Accelerometers and how they work”, [en línea]. Disponible en la web: <http://www2.usfirst.org/2005comp/Manuals/Acceler1.pdf> (último acceso junio 2015)
- [20] “¿A cuántas 'g' sometemos nuestro cuerpo en actividades cotidianas como toser o sentarnos en una silla?”, [en línea]. Disponible en la web: <http://www.xatakaciencia.com/fisica/a-cuantas-g-sometemos-nuestro-cuerpo-en-act> (último acceso junio 2015)
- [21] “Accelerometers: What They Are & How They Work”, [en línea]. Disponible en la web: <http://www.livescience.com/40102-accelerometers.html> (último acceso julio 2015)
- [22] “Caídas en los ancianos: causas, consecuencias y prevención”, [en línea]. Disponible en la web: <http://escuela.med.puc.cl/publ/boletin/osteoporosis/CaidasAncianos.html> (último acceso junio 2015)
- [23] Fundación Andaluza de Servicios Sociales; Iavante. “Eres grande, cuídate”, [en línea]. Disponible en la web: http://www.iavante.es/portal3d/accion_social/precan/esp/doc/dossier_PRECAN.pdf (último acceso 2015)
- [24] “Pérdida de equilibrio y caídas de personas mayores”, [en línea]. Disponible en la web: http://www.mapfre.es/salud/es/cinformativo/Intentando_no_caer.shtml (último acceso junio 2015)

- [25] Dr. Adolfo Moyano Crespo. “Las caídas en los adultos mayores y sus consecuencias”, [en línea]. Disponible en la web: http://www.sanatorioallende.com/web/ES/las_caidas_en_adultos_mayores_y_sus_cons (último acceso junio 2015)
- [26] “Las temidas caídas”, [en línea]. Disponible en la web: mayores.consumer.es/documentos/mayores/atender_necesidades/caidas.php (último acceso junio 2015)
- [27] “Caídas en los ancianos, ¿por qué son tan habituales?”, [en línea]. Disponible en la web: <http://www.hola.com/salud/2014021269665/caidas-personas-mayores/> (último acceso junio 2015)
- [28] “Página oficial Sense4care”, [en línea]. Disponible en la web: <http://www.sense4care.com/en> (último acceso junio 2015)
- [29] “Página oficial Vigi’Fall”, [en línea]. Disponible en la web: <http://www.vigilio.fr/#> (último acceso junio 2015)
- [30] “Dispositivo de alarma móvil GEA”, [en línea]. Disponible en la web: (último acceso <http://www.tecnalia.com/es/defensaseguridad/dispositivo-de-alarma-movil-gea.htm> 2015)
- [31] Google Play. “Fade: fall detector”, [en línea]. Disponible en la web: <https://play.google.com/store/apps/details?id=com.itec.falldetector&hl=es> (último acceso junio 2015)
- [32] “Web Apps Are the Future”, [en línea]. Disponible en la web: <http://www.sitepoint.com/long-live-web-app/> (último acceso junio 2015)
- [33] “Living with Web Apps”, [en línea]. Disponible en la web: <http://paul.kinlan.me/living-with-web-apps/> (último acceso junio 2015)
- [34] “Mobile: Native Apps, Web Apps, and Hybrid Apps”, [en línea]. Disponible en la web: <http://www.nngroup.com/articles/mobile-native-apps/> (último acceso junio 2015)
- [35] “Web Sites VS. Web Apps: What the Experts Think”, [en línea]. Disponible en la web: <http://www.visionmobile.com/blog/2013/07/web-sites-vs-web-apps-what-the-experts> (último acceso junio 2015)
- [36] “Mobile Website vs. Mobile App: Which is Best for Your Organization?”, [en línea]. Disponible en la web: <http://www.hwsolutions.com/services/mobile-web-development/mobile-website-vs-a> (último acceso junio 2015)
- [37] “Aplicaciones web vs. aplicaciones nativas vs. aplicaciones híbridas”, [en línea]. Disponible en la web: <http://blogthinkbig.com/aplicaciones-web-nativas-hibridas/> (último acceso junio 2015)

- [38] Kseso. “Text-align: Justify y RWD”, [en línea]. Disponible en la web: <http://ksesocss.blogspot.com/2013/06/text-align-justify-y-rwd.html> (último acceso abril 2015)
- [39] Kseso. “Mantener relación de aspecto con Css”, [en línea]. Disponible en la web: <http://ksesocss.blogspot.com/2013/08/aspect-ratios-css.html> (último acceso abril 2015)
- [40] Kseso. “Background 100 %. 2 técnicas”, [en línea]. Disponible en la web: <http://ksesocss.blogspot.com/2012/04/background-100-21-tecnicass-3-al-viejo.html> (último acceso abril 2015)
- [41] Kseso. “Display: inline-block y sus espacios en blanco de separación”, [en línea]. Disponible en la web: <http://ksesocss.blogspot.com/2012/03/display-inline-block-y-sus-empeno-en.html> (último acceso abril 2015)
- [42] Kseso. “Z-index en Css: el apilamiento de cajas y sus valores negativos”, [en línea]. Disponible en la web: <http://ksesocss.blogspot.com/2012/05/z-index-en-css-el-apilamiento-de-cajas.html> (último acceso abril 2015)
- [43] “A Complete Guide to Flexbox”, [en línea]. Disponible en la web: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/> (último acceso abril 2015)
- [44] “Usando las cajas flexibles CSS”, [en línea]. Disponible en la web: https://developer.mozilla.org/es/docs/Web/Guide/CSS/Cajas_flexibles (último acceso abril 2015)
- [45] “Animación con Keyframes en CSS3”, [en línea]. Disponible en la web: <http://www.cristalab.com/tutoriales/animacion-con-keyframes-en-css3-c1131441/> (último acceso abril 2015)
- [46] “An Introduction To CSS3 Keyframe Animations”, [en línea]. Disponible en la web: <http://www.cristalab.com/tutoriales/animacion-con-keyframes-en-css3-c1131441/> (último acceso abril 2015)
- [47] “Detecting de completion of a transition”, [en línea]. Disponible en la web: https://developer.mozilla.org/en-US/docs/Web/Guide/CSS/Using_CSS_transitions#Detection (último acceso abril 2015)
- [48] “Animaciones CSS3 vs Animaciones jQuery”, [en línea]. Disponible en la web: <http://www.desarrolloweb.com/articulos/animaciones-css3-animaciones-jquery.html> (último acceso abril 2015)
- [49] “What font size attribute should be used for text scaling and why?”, [en línea]. Disponible en la web: <http://www.oodlestechnologies.com/blogs/What-font-size-attribute-should-be-used> (último acceso mayo 2015)

- [50] "How we learned to leave default font-size alone and embrace the em", [en línea]. Disponible en la web: <https://www.filamentgroup.com/lab/how-we-learned-to-leave-body-font-size-alone/> (último acceso mayo 2015)
- [51] "What is jQuery?", [en línea]. Disponible en la web: <http://www.jquery-tutorial.net/introduction/what-is-jquery/> (último acceso abril 2015)
- [52] "The difference between 'return false;' and 'e.preventDefault();'", [en línea]. Disponible en la web: <https://css-tricks.com/return-false-and-prevent-default/> (último acceso abril 2015)
- [53] "Let's Play With Hardware-Accelerated CSS", [en línea]. Disponible en la web: <http://www.smashingmagazine.com/2012/06/21/play-with-hardware-accelerated-css/> (último acceso marzo 2015)
- [54] "Funciones callback (Javascript y Jquery)", [en línea]. Disponible en la web: <http://www.abcdiseño.com/funciones-callback/> (último acceso abril 2015)
- [55] "jQuery, optimizando para aumentar el rendimiento", [en línea]. Disponible en la web: <http://www.arumeinformatica.es/blog/jquery-optimizando-para-aumentar-el-rendimiento> (último acceso abril 2015)
- [56] "Ámbitos y Alcance en Javascript", [en línea]. Disponible en la web: <http://pensamientoobjetivo.blogspot.com.es/2009/09/ambitos-y-alcance-en-javascript.html> (último acceso abril 2015)
- [57] "Performance.now()", [en línea]. Disponible en la web: <https://developer.mozilla.org/en-US/docs/Web/API/Performance/now> (último acceso mayo 2015)
- [58] "Temporizadores en JavaScript", [en línea]. Disponible en la web: <https://geekytheory.com/tempORIZADORES-en-javascript/> (último acceso mayo 2015)
- [59] "Detect viewport orientation", [en línea]. Disponible en la web: <http://stackoverflow.com/questions/4917664/detect-viewport-orientation-if-orientation> (último acceso mayo 2015)
- [60] "Hash Tables in Javascript", [en línea]. Disponible en la web: http://www.mojavelinux.com/articles/javascript_hashes.html (último acceso abril 2015)
- [61] "Introducción a los servicios en Android", [en línea]. Disponible en la web: <http://www.androidcurso.com/index.php/tutoriales-android/38-unidad-8-servicios> (último acceso abril 2015)
- [62] "Utilizar servicios (Service e IntentService) en Android", [en línea]. Disponible en la web:

<http://gpmess.com/blog/2014/08/14/utilizar-servicios-service-e-intentservice-en>
(último acceso abril 2015)

- [63] “Services”, [en línea]. Disponible en la web:
<http://developer.android.com/guide/components/services.html#Lifecycle>
(último acceso abril 2015)
- [64] “Android Development Tutorials”, [en línea]. Disponible en la web:
<http://www.vogella.com/tutorials/android.html> (último acceso abril 2015)
- [65] “How to call a method stored in a HashMap?”, [en línea]. Disponible en la web:
[http://stackoverflow.com/questions/4480334/how-to-call-a-method-stored-in-a-has](http://stackoverflow.com/questions/4480334/how-to-call-a-method-stored-in-a-hash)
(último acceso abril 2015)
- [66] “Sensores en Android: Acelerómetro”, [en línea]. Disponible en la web:
<https://sekthdroid.wordpress.com/2013/03/12/sensores-en-android-acelerometro/>
(último acceso junio 2015)
- [67] “SensorManager”, [en línea]. Disponible en la web:
<http://developer.android.com/reference/android/hardware/SensorManager.html#regi>
(último acceso junio 2015)
- [68] “How to Check for Network Availability in Android App”, [en línea]. Disponible en la web:
<http://inducesmile.com/android-game-development/how-to-check-for-network-availa>
(último acceso mayo 2015)
- [69] “Localización geográfica en Android”, [en línea]. Disponible en la web:
<http://www.sgoliver.net/blog/localizacion-geografica-en-android-i/>
(último acceso mayo 2015)
- [70] “LocationStrategies”, [en línea]. Disponible en la web:
<http://www.sgoliver.net/blog/localizacion-geografica-en-android-i/>
(último acceso mayo 2015)
- [71] “Android’s Notification Center”, [en línea]. Disponible en la web:
<http://www.sgoliver.net/blog/localizacion-geografica-en-android-i/>
(último acceso junio 2015)
- [72] “Android Notification Sound”, [en línea]. Disponible en la web:
<http://stackoverflow.com/questions/15809399/android-notification-sound>
(último acceso mayo 2015)
- [73] “Calling startActivity() from outside of an Activity context”, [en línea]. Disponible en la web:
<http://stackoverflow.com/questions/3918517/calling-startactivity-from-outside-o>
(último acceso junio 2015)