

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN



TRABAJO FIN DE MÁSTER

MÁSTER EN TELEMEDICINA Y BIOINGENIERÍA

Aplicación móvil para el registro, monitorización
y refuerzo de estilo de vida saludable

Yaiza García Martín-Mantero

2011

UNIVERSIDAD POLITÉCNICA DE MADRID

**ESCUELA TÉCNICA SUPERIOR DE INGENIEROS
DE TELECOMUNICACIÓN**

Dpto. de Tecnología Fotónica
Grupo de Bioingeniería y Telemedicina

TRABAJO FIN DE MÁSTER

MÁSTER EN TELEMEDICINA Y BIOINGENIERÍA

Aplicación móvil para el registro, monitorización
y refuerzo de estilo de vida saludable

Yaiza García Martín-Mantero

2011

TRABAJO FIN DE MÁSTER

Título: Aplicación móvil para el registro, monitorización y refuerzo de estilo de vida saludable

Autor: Yaiza García Martín-Mantero

Tutor/a: D. José Manuel Iniesta Chamorro

Tribunal:

Presidente: D. Enrique J. Gómez Aguilera

Vocal: D^a. M^a Elena Hernando Pérez

Vocal secretario: D. Javier Serrano Olmedo

Suplente: D. Pedro J. Benito Peinado

Fecha de lectura:

Calificación:

Resumen:

Este trabajo de fin de máster presenta una aplicación desarrollada para sistemas operativos móviles Android cuyo objetivo es fomentar un estilo de vida saludable mediante el registro e interpretación de la señal de acelerometría recogida por el sensor incluido en el dispositivo. Además de ello, se produce una realimentación de información al usuario para que éste sea consciente en todo momento de sus progresos. La evolución del usuario puede consultarse de acuerdo a tres condiciones: el peso, el IMC (Índice de Masa Corporal) y la actividad realizada. Para conocer la actividad realizada, se ha llevado a cabo un estudio con su correspondiente clasificación en el que se han recogido los datos del acelerómetro durante hora y media mientras un individuo realizaba los tres tipos de actividades disponibles (reposo, andando, corriendo). Estos datos se analizaron mediante técnicas estadísticas para determinar los umbrales que caracterizaban cada una de las actividades. Estos valores son los que se incluyen en la aplicación final para conocer el ejercicio del usuario.

Palabras clave: Android, acelerómetro, preprocesamiento de señal, IMC (Índice de Masa Corporal)

Title: Mobile application for registration, monitoring and reinforcement of healthy lifestyle

Summary:

This project presents an application developed for Android mobile operating system with the aim to promote a healthy lifestyle by recording and interpreting the signal collected by the accelerometer sensor embedded in the device. Moreover, there is a feedback information to the user in order to have information about its progress. The evolution of the user can be found according to three criteria: weight, BMI (Body Mass Index) and the activity. For the activity, has conducted a study with the corresponding classification in which the data were collected from the accelerometer for an hour and a half while the individual conducting the three types of activities available (rest, walking, running). These data were analyzed using statistical techniques to determine the thresholds that characterize each of the activities. These values are included in the final application to meet the exercise of the user.

Keywords: Android, accelerometer, signal preprocessing, BMI (Body Mass Index)

Tabla de contenido

Tabla de ilustraciones	vii
1. Introducción.....	1
1.1. Justificación del trabajo	1
1.2. Estructura del documento	1
1.3. Objetivos	2
2. Antecedentes.....	5
2.1. Problema de salud a tratar. Estilos de vida saludable (dieta y ejercicio). Monitorización de actividad física.....	5
2.2. Aplicaciones de promoción de estilos de vida saludable y seguimiento de actividad física en dispositivos móviles.....	6
2.2.1. Fitness Tour	6
2.2.2. Nike+GPS	7
2.2.3. Adidas MiCoach	8
2.3. Preprocesamiento de acelerometría en dispositivos móviles	8
2.4. Análisis estadístico de los datos	11
2.4.1. Distribución Normal.....	11
2.4.2. Técnicas de comprobación de normalidad	12
2.4.3. Contraste de hipótesis de igualdad de medias.....	13
2.5. Sistemas operativos para móviles	13
3. Materiales.....	17
3.1. Plataforma Software utilizado.....	17
3.1.1. Plataforma Android.....	17
3.1.2. Lenguajes de programación	27
3.1.3. Eclipse	28
3.1.4. SQLite	29
3.1.5. AndroidPlot.....	29
3.1.6. SPSS.....	29
3.2. Hardware utilizado	29
4. Metodología de trabajo.....	33
4.1. Especificación de requisitos	33
4.2. Fase de análisis.....	33
4.2.1. Diagrama de Casos de Uso	33
4.2.2. Descripción de Casos de Uso	34
4.3. Fase de diseño.....	35

4.3.1.	Diagramas de Secuencia.....	35
4.3.2.	Estructura de la aplicación	39
4.4.	Detalles de implementación	41
4.4.1.	Visión General.....	42
4.4.2.	Persistencia de los datos	43
4.4.3.	Clasificación.....	43
4.4.4.	Página inicial.....	53
4.4.5.	Menú.....	54
4.4.6.	Configuración.....	55
4.4.7.	Play.....	57
4.4.8.	Evolución	64
4.4.9.	Consejos	70
4.4.10.	Ayuda	71
5.	Manual de usuario	73
6.	Conclusiones y Propuestas de Mejora	81
6.1.	Conclusiones	81
6.2.	Propuestas de Mejora.....	82
7.	Bibliografía.....	83

Tabla de ilustraciones

Ilustración 2.1.- Tabla de estado nutricional.....	6
Ilustración 2.2.- Arquitectura de Fitness Tour.....	7
Ilustración 2.3.- Clasificación de las técnicas de preprocesado.....	9
Ilustración 2.4.- Resultados cualitativos de las métricas del dominio del tiempo.....	10
Ilustración 2.5.- Resultados cualitativos de las métricas del dominio de la frecuencia.....	10
Ilustración 2.6.- Resultados cualitativos de las métricas del dominio de las cadenas.....	11
Ilustración 2.7.- Distribución Normal.....	12
Ilustración 2.8.- Función de una distribución normal.....	12
Ilustración 2.9.- Intención de compra según plataforma.....	14
Ilustración 2.10.- Tabla comparativa de sistemas operativos móviles	15
Ilustración 3.1.- Arquitectura de Android.	18
Ilustración 3.2.- Ciclo de vida de una actividad.	22
Ilustración 3.3.- Ciclo de vida de un servicio.	24
Ilustración 3.4.- Jerarquía de una interfaz de usuario.	26
Ilustración 3.5.- Tabla comparativa de las librerías de gráficos actuales.....	29
Ilustración 3.6.- Tamaño del dispositivo HTC Desire HD.....	30
Ilustración 3.7.- Tamaño de la pantalla del HTC Desire HD.....	30
Ilustración 3.8.- Ejes de medida del acelerómetro.	31
Ilustración 4.1.- Diagrama de casos de uso de la aplicación.	34
Ilustración 4.2.- Diagrama de secuencia "Configurar Datos"	36
Ilustración 4.3.- Diagrama de secuencia "Modificar Peso"	37
Ilustración 4.4.- Diagrama de secuencia "Lectura e Interpretación de la Señal de Acelerometría".	38
Ilustración 4.5.- Diagrama de secuencia "Consultar Evolución".....	38
Ilustración 4.6.- Diagrama de secuencia "Leer Consejos/Ayuda".....	39
Ilustración 4.7.- Esquema general de la aplicación.....	42
Ilustración 4.8.- Datos recogidos en reposo.	46
Ilustración 4.9.- Datos recogidos andando.....	47
Ilustración 4.10.- Datos recogidos corriendo.	47
Ilustración 4.11.- Resumen del procesamiento de los casos.....	47
Ilustración 4.12.- Pruebas de normalidad.	48
Ilustración 4.13.- Contraste de hipótesis de igualdad de medias.....	49
Ilustración 4.14.- Contraste de igualdad de medias por parejas.	50
Ilustración 4.15.- Valores medios y límites de los intervalos para cada una de las actividades.	51
Ilustración 4.16.- Representación de la media de la variable media_min respecto a los tres grupos de actividad.....	52
Ilustración 4.17.- Representación de la media de la variable desvT_min respecto a los tres grupos de actividad.....	52
Ilustración 4.18.- Pantalla inicial en bruto.....	53
Ilustración 4.19.- Pantalla inicial en detalle.....	53
Ilustración 4.20.- Diseño en bruto de la pantalla del menú.	54
Ilustración 4.21.- Diseño detallado de la pantalla del menú.....	54
Ilustración 4.22.- Diseño en bruto de la pantalla de configuración.....	56
Ilustración 4.23.- Diseño detallado de la pantalla de configuración.....	56

Ilustración 4.24.- Pantalla en bruto de la interfaz general de evolución.....	64
Ilustración 4.25.- Pantalla en detalle de la interfaz general de evolución.....	64
Ilustración 4.26.- Pantalla en bruto de consejos.....	70
Ilustración 4.27.- Pantalla en detalle de consejos.....	70
Ilustración 5.1.- Pantalla inicial de la aplicación.....	73
Ilustración 5.2.- Pantalla del menú de la aplicación.....	73
Ilustración 5.3.- Pantalla inicial de configuración.....	74
Ilustración 5.4.- Selección de la fecha de nacimiento.....	74
Ilustración 5.5.- Selección del género.....	74
Ilustración 5.6.- Pantalla con todos los datos rellenados.....	74
Ilustración 5.7.- Mensaje de datos guardados.....	75
Ilustración 5.8.- Pantalla de la interfaz general de evolución.....	75
Ilustración 5.9.- Diálogo para la modificación del peso.....	76
Ilustración 5.10.- Pantalla de la evolución del peso.....	76
Ilustración 5.11.- Pantalla de la evolución del IMC.....	77
Ilustración 5.12.- Pantalla real de la evolución de la actividad.....	77
Ilustración 5.13.- Pantalla real de consejos.....	78
Ilustración 5.14.- Pantalla real de ayuda.....	78
Ilustración 5.15.- Notificación en la barra del dispositivo móvil.....	79
Ilustración 5.16.- Mensaje como notificación.....	79

1. Introducción

En este capítulo se va a realizar una breve introducción al presente trabajo fin de máster, justificando el por qué realizar un proyecto de estas características, la estructura que va a presentar y los objetivos que se pretenden cumplir.

1.1. Justificación del trabajo

El teléfono móvil se ha convertido en un periodo de pocos años en una parte fundamental de la sociedad, llegando incluso al hecho de que en España existen más líneas de telefonía móvil que habitantes (1). Lejos de soportar únicamente comunicaciones por voz, estos dispositivos cuentan con arquitecturas hardware internas sofisticadas, y presentan la posibilidad de extender sus funciones como pueden ser la localización por GPS, el e-mail, el organizador e incluso la sincronización con otros dispositivos externos. Las unidades avanzadas, pueden incluso ir equipadas con una serie de sensores internos, como los acelerómetros de tres dimensiones (2).

En la actualidad, la población mundial se ve afectada por un creciente número de enfermedades de aparición relativamente reciente, que se están convirtiendo rápidamente en un serio problema desde el punto de vista de la salud pública (3). Las enfermedades cardiovasculares son un claro ejemplo de este tipo de condiciones, que han experimentado un aumento significativo en cuanto a su incidencia en la población general. Las principales causas de dichas enfermedades son la diabetes, la hiperlipidosis y el síndrome metabólico. En las últimas décadas, la progresión del número de casos ha sido geométrica. Además, dicha incidencia se ha extendido a los segmentos más jóvenes de la población.

Para la prevención de las causas de enfermedades cardiovasculares mencionadas anteriormente, se recomienda un estilo de vida saludable, mediante una dieta equilibrada, un régimen de ejercicio razonable y la supresión del tabaco, entre otras consideraciones. El papel que juegan las nuevas tecnologías en la consecución de este estilo de vida saludable ha pasado a un primer plano. Con la información obtenida de los sensores, se pueden ofrecer servicios con valor añadido a los usuarios de los dispositivos móviles. Por ejemplo, analizando los datos del acelerómetro, se puede saber si el usuario está llevando a cabo una actividad física como correr o andar. Esta información puede recogerse durante un periodo de tiempo, ya sea una semana o un mes, con el objetivo de conocer rutinas o hábitos del usuario.

Con todo lo anterior se establece el objetivo principal del presente trabajo: *Crear un sistema de monitorización y reconocimiento de actividad física basándose en los datos obtenidos por el acelerómetro incluido en el teléfono móvil.*

1.2. Estructura del documento

Este documento se va a estructurar en distintos apartados como se comenta a continuación:

En el primer capítulo o *Introducción*, como su propio nombre indica, se realiza una introducción al presente trabajo, justificando el por qué de la realización del mismo, su estructura, y los objetivos que pretende cumplir.

A continuación, en el segundo tema, se realiza un *Estado del arte* para conocer las distintas opciones de desarrollo existentes y elegir así la más adecuada para implementar la aplicación.

En el capítulo de *Material y métodos*, se explican tanto el software como el hardware empleados en la realización del presente trabajo fin de máster.

La metodología empleada y la descripción de las distintas fases del desarrollo del proyecto: análisis, diseño e implementación se encuentran en el capítulo cuarto, *Metodología de trabajo*. Los *resultados* de aplicar dicha metodología, se muestran en el quinto capítulo.

A continuación se mencionan las *conclusiones* obtenidas al realizar el trabajo, así como las posibles *mejoras* que podrían llevarse a cabo.

Para finalizar, se muestra el listado de *referencias bibliográficas* consultadas en la realización del trabajo.

1.3. Objetivos

El objetivo de este Trabajo de Fin de Máster (TFM) es el diseño e implementación de una aplicación móvil para el registro y monitorización de la actividad física empleando como dispositivo de medición el acelerómetro incluido en el teléfono móvil. La aplicación analizará los datos recogidos e informará al usuario para sea consciente de su evolución mediante gráficas y mensajes para provocar así que éste modifique sus hábitos hacia un estilo de vida saludable.

El objetivo principal puede desglosarse en objetivos específicos que en cierto modo representan las diferentes funcionalidades a desarrollar en el presente TFM:

- Determinar el problema de salud a tratar.
- Estudio de mercado para determinar para qué sistema operativo móvil desarrollar la aplicación.
- Estudio de la plataforma de desarrollo para Android.
- Revisión de las librerías disponibles para realizar gráficos con Android y estudio de sus características para determinar cuál se ajusta más a los objetivos deseados.
- Estudio de las técnicas de preprocesado de señales de acelerometría.
- Evaluación del coste computacional de las técnicas empleadas en el tratamiento de señales de acelerometría para determinar cuáles son las más viables para su implementación en dispositivos móviles.
- Diseño del proceso de registro de la actividad física mediante el empleo del acelerómetro incluido en el dispositivo móvil.
- Realización de una clasificación para determinar las características de la señal que determinan el tipo de actividad realizada. Se valorarán tres actividades distintas:
 - Reposo
 - Actividad moderada, que se corresponde con andar.
 - Actividad intensa, que se corresponde con correr.
- Repaso a las técnicas de análisis estadístico de los datos.

- Familiarización con el programa estadístico SPSS para poder analizar los datos que darán lugar a la clasificación de actividades.
- Diseño e implementación de una forma de aconsejar al usuario sobre su estilo de vida en cuanto a la actividad física se refiere.
- Diseño e implementación de gráficas que permitan al usuario ver la evolución de su peso y de su Índice de Masa Corporal (IMC). Para que esto funcione correctamente, se debe permitir que el usuario introduzca manualmente su peso semanalmente.
- Diseño e implementación de una base de datos que de soporte a los datos necesarios para cumplir los objetivos propuestos.

2. Antecedentes

En este apartado se van a desarrollar los conocimientos adquiridos durante la búsqueda bibliográfica. Debido a la amplitud del mismo y para facilitar su lectura, se muestra a continuación un breve esquema de los puntos a tratar en él:

- 1) Exposición del problema de salud a tratar. Rol que desempeña la promoción de estilos de vida saludable (dieta y ejercicio). Actividad física y posibilidades actuales para su monitorización.
- 2) A continuación se hace un breve repaso de algunas de las aplicaciones para dispositivos móviles que fomentan un estilo de vida saludable y que realizan un seguimiento de la actividad física del usuario.
- 3) Dentro de las diferentes técnicas empleadas para determinar la actividad física realizada una de las más extendidas es el análisis de datos de acelerometría. Por ello, se especificarán algunas técnicas de preprocesado de la señal de acelerometría. Además de esto, se estudia si dichas técnicas son idóneas para su implementación en dispositivos móviles teniendo en cuenta sus costes de almacenamiento y consumo de recursos.
- 4) También se comentarán técnicas utilizadas en la literatura para seleccionar los umbrales que determinen el tipo de actividad física realizada por el usuario en cada momento utilizando los datos del acelerómetro.
- 5) Por último se hace una breve reseña de los principales sistemas operativos para móviles existentes en la actualidad con una mayor relevancia en el mercado.

2.1. Problema de salud a tratar. Estilos de vida saludable (dieta y ejercicio). Monitorización de actividad física.

En la actualidad, la población mundial se ve afectada por un creciente número de enfermedades de aparición relativamente reciente, que se están convirtiendo rápidamente en un serio problema desde el punto de vista de la salud pública (3). Las enfermedades cardiovasculares son un claro ejemplo de este tipo de condiciones, que han experimentado un aumento significativo en cuanto a su incidencia en la población general. Las principales causas de dichas enfermedades son la diabetes, la hiperlipidosis y el síndrome metabólico. En las últimas décadas, la progresión del número de casos ha sido geométrica. Además, dicha incidencia se ha extendido a los segmentos más jóvenes de la población.

Para la prevención de las causas de enfermedades cardiovasculares mencionadas anteriormente, se recomienda un estilo de vida saludable, mediante una dieta equilibrada, un régimen de ejercicio razonable y la supresión del tabaco, entre otras consideraciones.

Existen varias formas para medir si una persona lleva un estilo de vida saludable o no. Una de ellas es mediante el IMC (Índice de Masa Corporal, IBM en inglés, Index Body Mass). El valor de este índice se divide en rangos (ver Ilustración 2.1), según el exceso de masa corporal del individuo (4). Es más, conociendo el IMC de un individuo, puede determinarse si este sufre riesgo de contraer alguna enfermedad. Algunas condiciones ligadas al sobrepeso incluyen: muerte prematura, enfermedades cardiovasculares, alta tensión, osteoartritis, algunos tipos de cáncer y diabetes.

BMI	Nutritional status
Below 18.5	Underweight
18.5–24.9	Normal weight
25.0–29.9	Pre-obesity
30.0–34.9	Obesity class I
35.0–39.9	Obesity class II
Above 40	Obesity class III

Ilustración 2.1.- Tabla de estado nutricional.

2.2. Aplicaciones de promoción de estilos de vida saludable y seguimiento de actividad física en dispositivos móviles.

El teléfono móvil se ha convertido en un periodo de pocos años en una parte fundamental de la sociedad, llegando incluso al hecho de que en España existen más líneas de telefonía móvil que habitantes (1). Lejos de soportar únicamente comunicaciones por voz, estos dispositivos cuentan con arquitecturas hardware internas sofisticadas, y presentan la posibilidad de extender sus funciones como pueden ser la localización por GPS, el e-mail, el planificador e incluso la sincronización con otros dispositivos externos. Las unidades avanzadas, pueden incluso ir equipadas con una serie de sensores internos, como los acelerómetros de tres dimensiones (2).

En la actualidad, el papel que juegan las nuevas tecnologías en la consecución de este estilo de vida saludable ha pasado a un primer plano. Con la información obtenida de los sensores, se pueden ofrecer servicios con valor añadido a los usuarios de los dispositivos móviles. Por ejemplo, analizando los datos del acelerómetro, se puede saber si el usuario esté llevando a cabo una actividad física como correr o andar. Esta información puede recogerse durante un periodo de tiempo, ya sea una semana o un mes, con el objetivo de conocer rutinas o hábitos del usuario.

Aunque el mercado de aplicaciones para móviles que monitorizan la actividad física es elevado, el de aquellas con el objetivo de fomentar un estilo de vida saludable son más bien escasas. A continuación se comentan y analizan algunas aplicaciones relevantes de seguimiento de actividad física existentes.

2.2.1. Fitness Tour

Se trata de una aplicación (5) desarrollada para smartphones basados en la plataforma Android centrada en el dominio de la asistencia sanitaria. Dicha aplicación cuenta con las siguientes características:

- Generación aleatoria de ejercicios.
- Verificación segura de si se ha completado una serie de ejercicios

- Interfaz de red social que permite a los usuarios invitar a amigos para que vean sus marcas o para que participen en competiciones organizadas.

El principal objetivo de esta aplicación es que niños y adolescentes adopten un estilo de vida más activo para combatir así el problema de la obesidad.

La aplicación hace una lectura del ritmo cardiaco del usuario y recoge las calorías quemadas, enviando los datos a un servidor remoto. En la Ilustración 2.2 puede verse la arquitectura de esta aplicación. En ella se observa como la aplicación consta de dos componentes principales:

- Módulo cliente, que corre en el *smartphone*.
- Módulo servidor, que corre en un servidor remoto.

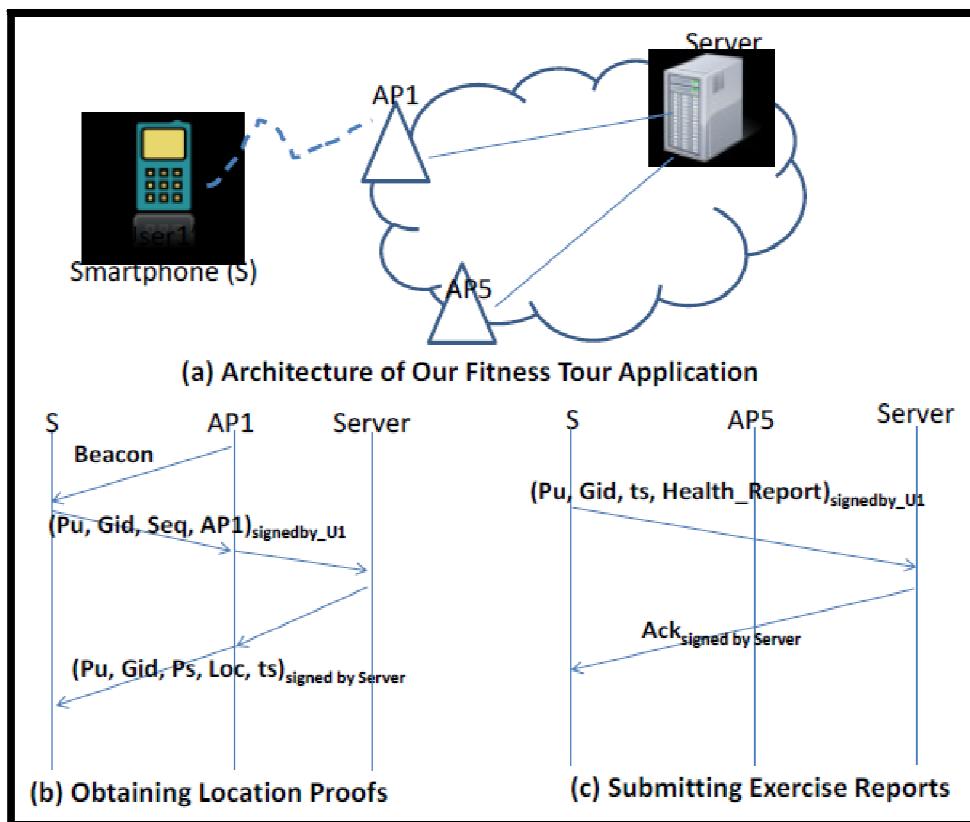


Ilustración 2.2.- Arquitectura de Fitness Tour.

2.2.2. Nike+GPS

Se trata de una aplicación pensada para los usuarios de iPod Touch o iPhone 3GS o 4 que utiliza la antena de geolocalización de estos dispositivos para rastrear el recorrido se puede planificar previamente, la distancia y la velocidad media del atleta (6) .

La aplicación permite al usuario saber en tiempo real el crono que han marcado otros individuos, compartir los datos propios online -gracias a la conexión 3G del iPhone y con una WiFi en el caso del Touch-. Del mismo modo, si se adjunta el correspondiente sensor, el gadget también controlará el ritmo cardiaco y las calorías.

Entre otras funciones más útiles que la sincronización de datos con Twitter o Facebook se encuentra la posibilidad de encontrar amigos que estén corriendo en ese mismo momento, la posibilidad de gestionar las listas de reproducción fácilmente e, incluso la voz del entrenador.

2.2.3. Adidas MiCoach

Esta aplicación (7), permite seleccionar un plan para comenzar a entrenar. Los planes disponibles son: entrenamiento para hombres y mujeres, correr, fútbol, tenis, fútbol americano y baloncesto.

Una vez elegido el plan, se empieza con el entrenamiento. Los entrenamientos de miCoach se basan en cuatro zonas de colores personalizadas para el usuario y su plan cardiovascular o de flexibilidad y resistencia.

miCoach comprueba los entrenamientos cardiovasculares con comentarios audibles en tiempo real.

Otra característica de esta aplicación, es que se sincroniza online para que se pueda comprobar las estadísticas, introducir información sobre la flexibilidad y fuerza, realizar un seguimiento del progreso y obtener comentarios de todos los entrenamientos.

Tras sincronizar un entrenamiento, se obtiene una puntuación que relaciona las estadísticas con los parámetros del entrenamiento, la amplitud de las zonas, el equipo que utilizado y otros factores.

2.3. Preprocesamiento de acelerometría en dispositivos móviles

Al contrario de lo que ocurre con otros sensores que proporcionan valores que pueden emplearse directamente en el proceso de inferencia del contexto, las señales procedentes de los acelerómetros requieren de una fase compleja de preprocesado para poder caracterizar la actividad física que un usuario realiza en un periodo concreto de tiempo (2).

La necesidad de extraer las características clave de la señal que permitan avanzar en los algoritmos de preprocesado para obtener información útil del contexto, ha llevado al desarrollo de una amplia variedad de aproximaciones algorítmicas. Estas aproximaciones convierten o transforman las señales de entrada en diferentes dominios de representación.

Como se observa en la Ilustración 2.3, es posible clasificar las técnicas disponibles de preprocesamiento de señales de sensores en tres dominios:

- Dominio del tiempo
- Dominio de la frecuencia
- Dominios de representaciones discretas

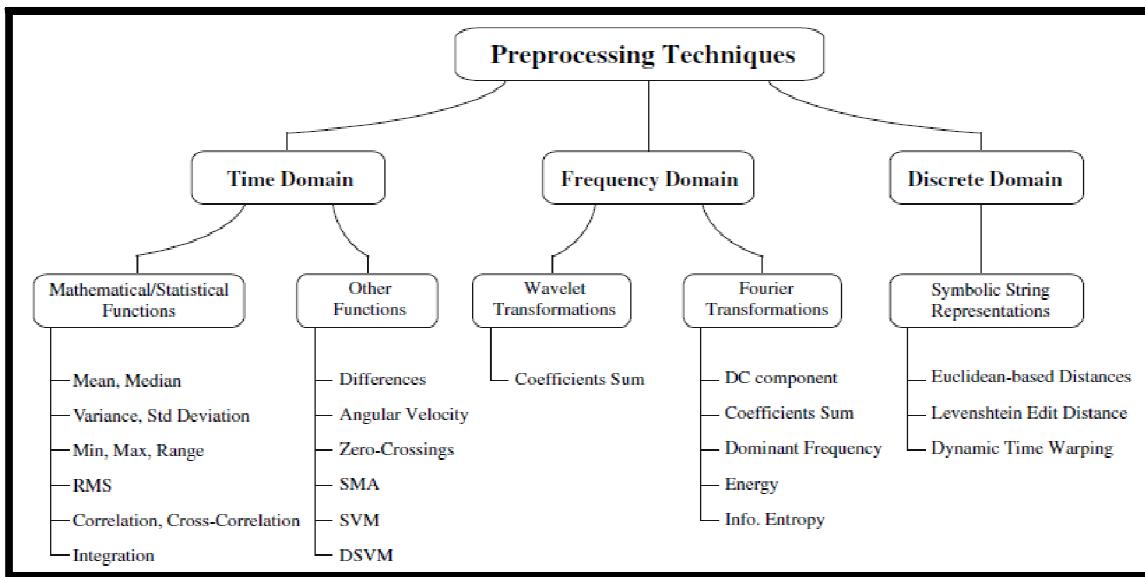


Ilustración 2.3.- Clasificación de las técnicas de preprocesado.

Todas estas técnicas, presentan costes computacionales y de almacenamiento diferentes, lo que las hace más o menos idóneas para ser implementadas en dispositivos móviles como los *smartphones* que presentan recursos limitados. A continuación, se compararán dichas técnicas para saber cuáles de ellas deberían usarse en dispositivos móviles y cuáles no.

La complejidad de los algoritmos se va a estudiar en función de las operaciones que realiza para evitar así las posibles diferencias entre procesadores, compiladores, etc. En el cálculo del coste computacional, se incluyen el número de sumas, multiplicaciones, y otras operaciones lógicas y aritméticas. Para simplificar los cálculos, se asume que las operaciones se realizan en punto flotante (32-bits de precisión simple). En algunos casos, existe la necesidad de un paso inicial de normalización para transformar los tres vectores de entrada de enteros (uno por cada eje del acelerómetro) en un único vector en punto flotante. Este coste computacional, no se tiene en cuenta a la hora de calcular el coste total de la función.

Además del coste operacional, también se han visto los requerimientos de memoria y almacenamiento para mantener las variables temporales y los datos de entrada de la señal.

En las siguientes tres tablas (Ilustración 2.4, Ilustración 2.5 e Ilustración 2.6), se muestran los resultados cualitativos de los dos estudios mencionados anteriormente, empleando las siguientes etiquetas:

- *Very low*: Si sólo requiere de un número de operaciones con una relación lineal con el número de muestras de entrada. Estas operaciones suelen ser sumas y restas aritméticas.
- *Low*: Si el número de operaciones sigue una relación lineal con el número de muestras de entrada, incluyendo multiplicaciones y divisiones.
- *Medium*: Incluye métricas cuadráticas en términos de su entrada.
- *High*: Incluye técnicas que requieren un número de operaciones mayor de una frontera asintótica cuadrática.

Conforme a estos valores, se determina y las características deben implementarse o no en un dispositivo móvil. Se etiqueta cada métrica con tres posibles etiquetas:

- *Yes*: Indica que la métrica es idónea para implementarla en un dispositivo móvil.
- *No*: Indica que no lo es.
- *Moderate*: Indica que la métrica emplea un nivel medio de recursos, por lo que en caso de implementarse, deberá hacerse con precaución.

En general, las métricas en el dominio del tiempo (Ilustración 2.4) están dominadas por el coste de la normalización y ninguna de ellas contiene operaciones aritméticas complejas como funciones trigonométricas y logarítmicas. Se puede decir que, en general, estas métricas son buenos candidatos para su implementación en dispositivos móviles.

Time-domain metric	Comp. cost	Storage req.	Precision	Mobile device
Mean	Very low	Very low	Single/int	Yes
Std. deviation	Very low	Very low	Double/single	Yes
Median	Medium	Very low	Single/int	Yes
Range	Very low	Very low	Single/int	Yes
Maximum	Very low	Very low	Single/int	Yes
Minimum	Very low	Very low	Single/int	Yes
RMS	Very low	Very low	Double/single	Yes
Integration	Very low	Very low	Double/single	Yes
Correlation	Medium	Low	Double/single	Moderate
Cross-correlation	Medium	Low	Double/single	Moderate
Differences	Very low	Very low	Single/int	Yes
Zero-crossings	Very low	Very low	Single/int	Yes
SMA	Low	Low	Single/int	Yes
SVM	Low	Low	Double/single	Yes
DSVM	Low	Low	Double/single	Yes

Ilustración 2.4.- Resultados cualitativos de las métricas del dominio del tiempo.

En la Ilustración 2.5, se muestran los resultados cualitativos para el dominio de la frecuencia. Se puede observar como las métricas del domino de la frecuencia son computacionalmente más caras que las del tiempo. Con la excepción de las transformadas Wavelets, el resto se basan en el espectro de la señal, por lo que presentan valores medios-altos de coste computacional. Esto hace que las Wavelets sean las únicas recomendadas para implementar en dispositivos móviles.

Frequency-domain metric	Comp. cost	Storage req.	Precision	Mobile device
Energy	Medium	Low	Double/single	Moderate
Entropy	High	Low	Double/single	No
Coeff. sum	Medium	Low	Double/single	Moderate
Dominant freq.	Medium	Low	Double/single	Moderate
Wavelet (H_2 and coeff. sum)	Low	Low	Double/single	Yes

Ilustración 2.5.- Resultados cualitativos de las métricas del dominio de la frecuencia.

En la Ilustración 2.6, se muestran los resultados en el dominio de las cadenas. Observando dichos resultados, se ve como las técnicas implementadas con programación dinámica (Levenshtein y DTW), presentan un mayor coste que la de la distancia Euclídea, por lo que esta última es la más idónea para implementar en un dispositivo móvil.

String-domain metric	Comp. cost	Storage req.	Precision	Mobile device
Minimum distance	Low	Low	Int	Yes
Levenshtein	Medium	Medium	Int	Moderate
DTW	Medium	Medium	Int	Moderate

Ilustración 2.6.- Resultados cualitativos de las métricas del dominio de las cadenas.

2.4. Análisis estadístico de los datos

En este apartado se comentan los métodos empleados en el análisis estadístico de los datos llevado a cabo para determinar el umbral que va a determinar el tipo de actividad que realiza el usuario.

2.4.1. Distribución Normal

La distribución de probabilidad conocida como distribución normal es, por la cantidad de fenómenos que explica, la más importante de las distribuciones estadísticas (8).

La distribución de probabilidad normal y la curva normal que la representa, tienen las siguientes características (ver Ilustración 2.7):

- La curva normal tiene forma de campana y un solo pico en el centro de la distribución. De esta manera, la media aritmética, la mediana y la moda de la distribución son iguales y se localizan en el pico. Así, la mitad del área bajo la curva se encuentra a la derecha de este punto central y la otra mitad está a la izquierda de dicho punto.
- La distribución de probabilidad normal es simétrica alrededor de su media.
- La curva normal desciende suavemente en ambas direcciones a partir del valor central. Es asintótica, lo que quiere decir que la curva se acerca cada vez más al eje X pero jamás llega a tocarlo. Es decir, las "colas" de la curva se extienden de manera indefinida en ambas direcciones.

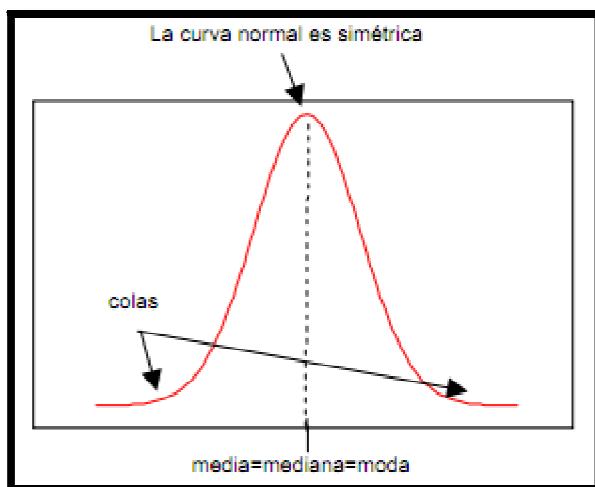


Ilustración 2.7.- Distribución Normal.

Formalmente (9), se dice que una variable aleatoria X , tiene una distribución normal con parámetros μ y σ , donde $-\infty < \mu < \infty$ y $0 < \sigma$, si la función de distribución de probabilidad de X es:

$$f(x; \mu, \sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-(x-\mu)^2/(2\sigma^2)}$$

Ilustración 2.8.- Función de una distribución normal.

2.4.2. Técnicas de comprobación de normalidad

Kolmogorov-Smirnov

En estadística, la prueba de Kolmogórov-Smirnov (también prueba K-S) es una prueba no paramétrica que se utiliza para determinar la bondad de ajuste de dos distribuciones de probabilidad entre sí.

Conviene tener en cuenta que la prueba Kolmogórov-Smirnov es más sensible a los valores cercanos a la mediana que a los extremos de la distribución.

La distribución de los datos F_n para n observaciones y_i se define como:

$$F_n(x) = \frac{1}{n} \sum_{i=1}^n \begin{cases} 1 & \text{si } y_i \leq x, \\ 0 & \text{alternativa.} \end{cases}$$

Shapiro-Wilk

En estadística, el Test de Shapiro-Wilk (10), se usa para contrastar la normalidad de un conjunto de datos. Se plantea como hipótesis nula que una muestra x_1, \dots, x_n , proviene de una población normalmente distribuida. Se considera uno de los test más potentes para el contraste de normalidad, sobre todo para muestras pequeñas ($n < 30$).

El estadístico del test es:

$$W = \frac{\left(\sum_{i=1}^n a_i x_{(i)} \right)^2}{\sum_{i=1}^n (x_i - \bar{x})^2}$$

donde:

- $x(i)$ es el número que ocupa la i -ésima posición en la muestra.
- $\bar{x} = (x_1 + \dots + x_n) / n$ es la media muestral.
- Las constantes a_i se calculan como:

$$(a_1, \dots, a_n) = \frac{m^\top V^{-1}}{(m^\top V^{-1} V^{-1} m)^{1/2}}$$

donde

$$m = (m_1, \dots, m_n)'$$

siendo m_1, \dots, m_n son los valores medios del estadístico ordenado, de variables aleatorias independientes e idénticamente distribuidas, muestreadas de distribuciones normales. V es la matriz de covarianzas de ese estadístico de orden.

2.4.3. Contraste de hipótesis de igualdad de medias

Prueba de Kruskal-Wallis

Se trata de una técnica no paramétrica para decidir si un conjunto de k muestras provienen o no de la misma población (11). Esta prueba se puede describir así: supongamos que tenemos k muestras de tamaños N_1, N_2, \dots, N_k , siendo el tamaño total de todas las muestras juntas $N = N_1 + N_2 + \dots + N_k$. Supongamos además que los datos de todas las muestras conjuntas se ordenan y que las sumas de los órdenes para las k muestras son R_1, R_2, \dots, R_k , respectivamente. Si definimos el estadístico:

$$H = \frac{12}{N(N+1)} \sum_{j=1}^k \frac{R_j^2}{N_j} - 3(N+1)$$

entonces se puede demostrar que la distribución muestral de H tiene casi distribución chi cuadrado con $k-1$ grados de libertad, siempre y cuando N_1, N_2, \dots, N_k sean al menos 5.

La prueba H , ofrece un método no paramétrico en el *análisis de varianza* para clasificaciones simples o experimentos con un factor, y es posible hacer generalizaciones.

HSD de Tukey

Se le conoce como método de la diferencia honestamente significativa de Tukey (12). Todas las comparaciones se refieren a una misma diferencia mínima, o sea utiliza un sólo valor con el cual se comparan todos los posibles pares de medias. Numerosos autores coinciden en afirmar que la prueba es apropiada para realizar comparaciones por pares, por lo que tiene una gran aceptación.

Games-Howell

Se trata de un método muy similar al de Tukey (12). Se basa en la distribución del rango estudentizado y en un estadístico t en el que, tras estimar las varianzas poblacionales suponiendo que son distintas, se corrigen los grados de libertad mediante la ecuación de Welch.

2.5. Sistemas operativos para móviles

Un sistema operativo móvil, es el sistema operativo que controla un dispositivo móvil, al igual que los ordenadores utilizan Windows o Linux entre otros. Sin embargo, los sistemas operativos móviles son bastante más simples y están más orientados a la conectividad inalámbrica, los formatos multimedia para móviles y las diferentes maneras de introducir información en ellos.

A medida que los teléfonos móviles crecen en popularidad, los sistemas operativos con los que funcionan adquieren mayor importancia.

La compañía Nielsen (13) ha realizado un estudio sobre la intención de compra de los usuarios de móviles en función de sus sistemas operativos. Los resultados pueden verse en el siguiente gráfico, según el cual, se observa un crecimiento de la plataforma Android, mientras que el resto de sistemas operativos presentan una ligera bajada.

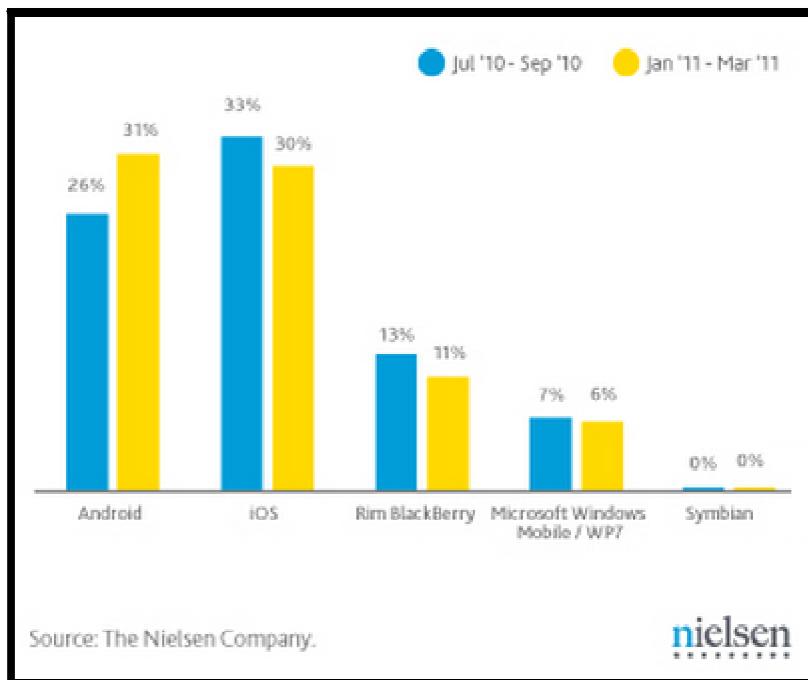


Ilustración 2.9.- Intención de compra según plataforma.

A continuación se discuten los principales aspectos de las plataformas de desarrollo para smartphones actuales: iOS, Symbian, Windows Phone 7 y Android, analizando sus ventajas e inconvenientes desde el punto de vista del desarrollo de software y relacionándolo con el del usuario. No se analiza la plataforma Blackberry ya que está más orientada al mundo profesional y no es el campo que nos interesa.

En la siguiente tabla se comparan distintos aspectos de los plataformas mencionadas (14)(15):

	Android	iOS	Symbian	Windows Phone 7
Buscador integrado	Si	Si	Si	Si
Compartir internet con otros dispositivos	Si - USB, hotspot Wifi o Bluetooth	No - Requiere modificar el dispositivo	Si – Wifi	N/D
Flash 10.1	Si	No	No – Versión Lite	No
Gestor de correo	Si	Si	Si	Si
Inicio personalizable	Si	Si - Limitado	Si	Si
Interacción con	Si	Si	Si	Si

redes sociales				
Juegos	Si – Market	Si – App Store	Si	Si
Libros (eBooks)	Si	Si	Si	No
Mapas con navegación GPS (gratuita)	Si – Google Maps	No	No	Si – Bing Maps
Multi-tarea	Si	Si – Limitado a ciertas aplicaciones y servicios	Si	Si – Limitado a ciertas aplicaciones y servicios
Multi táctil	Si	Si	Si	Si
Navegador Web	Si	Si	Si	Si
Servicios en "la nube"	Si	Si	Si	Si
Silverlight	No	No	No	Si
Suite Ofimática	Si	Si	Si	Si
Teclado	Físico y virtual	Sólo virtual	Físico y virtual	Físico y virtual
Temas personalizables para la IU	Si	Si- Limitado	No	No
Tienda de música	Si-Terceras partes	Si- iTunes	Si- Terceras Partes	Si- Zune
Tienda de aplicaciones	Si	Si	Si	Si
Lenguaje de programación	Java	Objective C	C/C++/Python	C/C++/.NET/Flash Lite
Necesidad de HW específico para programar	No	Si	No	No
Licencia	Liberada	Propietaria. Hay que pagar para desarrollar	Propietaria. Anunciada su liberación futura	Propietaria

Ilustración 2.10.- Tabla comparativa de sistemas operativos móviles

Analizando los datos de la tabla anterior, no se observa una gran diferencia entre las características principales de las cuatro plataformas, sin embargo, existen ciertos detalles que hacen a unas más atractivas que a otras desde el punto de vista de usuario y también desde el punto de vista de desarrollador.

Por un lado, desde la perspectiva del usuario, se da una mayor importancia a aspectos como la capacidad de personalización de la interfaz de usuario, si cuenta con un gran número de aplicaciones gratuitas, o si la plataforma cuenta con un amplio catálogo de juegos.

Teniendo en cuenta lo mencionados en el párrafo anterior, es fácil darse cuenta que las plataformas que cumplen en mayor grado estas condiciones son Android e iOS.

Cambiando el punto de vista al de un desarrollador de software, y relacionado con lo anterior, resulta evidente que en ambas plataformas existe una gran oportunidad de negocio, pues cada una de ellas cuenta con millones de usuarios.

A parte de todo lo mencionado anteriormente se deben tener en cuenta otras consideraciones para determinar la plataforma elegida y que están relacionadas con el propio proceso de desarrollo de aplicaciones para las mismas.

Ambas plataformas, iOS y Android, cuentan con bastantes herramientas de desarrollo, extensas APIs, multitud de librerías que extienden las funcionalidades de la plataforma y numeroso código fuente de ejemplo.

Sin embargo, Android cuenta con ciertas ventajas competitivas como poder desarrollar de forma independiente a la plataforma ya que en ningún caso requiere de hardware específico para el desarrollo, por el contrario iOS requiere contar con un equipo que tenga instalado Mac OS X; o el coste de la licencia de desarrollador siendo más baja para Android.

Además, Apple ha establecido una política de privacidad bastante restrictiva, en la que hasta hace poco no se permitía a los desarrolladores hablar sobre el SDK o compartir código, sin mencionar que se deben obtener varios certificados emitidos por la compañía para probar las aplicaciones desarrolladas. Por el contrario, Google con Android adopta una política más abierta facilitando el desarrollo de aplicaciones.

3. Materiales

En esta sección se van a comentar los materiales empleados para realizar el presente trabajo de fin de máster. Se dividen en dos grupos, por un lado los referentes al software y a la plataforma de desarrollo y por otro al hardware empleado.

3.1. Plataforma Software utilizado

3.1.1. Plataforma Android

Android es una plataforma de software (16), pensada especialmente para dispositivos móviles, y que en pocas palabras se compone de un sistema operativo, un conjunto de aplicaciones base y una capa middleware situada entre las aplicaciones y el propio sistema operativo.

Dichas aplicaciones base están desarrolladas utilizando el lenguaje de programación Java, mientras que por otro lado ciertos componentes de la arquitectura de Android como las librerías o el núcleo, basado en el kernel de Linux 2.6; están escritos en C/C++.

A continuación se describen los distintos aspectos de la plataforma como su arquitectura, los componentes de una aplicación, etc., haciendo hincapié en los aspectos a priori más importantes. Toda esta información ha sido obtenida de la página oficial de desarrollo de Android (17).

Las principales características de la plataforma Android son las siguientes:

- Cuenta con un framework de aplicaciones que permite reutilizar o sustituir las aplicaciones existentes.
- Cuenta con una máquina virtual especialmente optimizada para dispositivos móviles conocida como Dalvik VM.
- Navegador integrado basado en el motor de código abierto WebKit.
- Capaz de procesar gráficos en 2D (gracias a la librería SGL) y gráficos 3D basados en la especificación OpenGL ES 1.X/2.0 (dependiendo de la versión de Android).
- Soporte para el almacenamiento de datos estructurados mediante las librerías SQLite.
- Soporte de múltiples formatos multimedia tanto de audio como video o imagen (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF).
- Telefonía GSM.
- Comunicaciones siguiendo los protocolos estándar Bluetooth, EDGE, 3G, y Wifi.
- Soporte de múltiples dispositivos hardware como cámaras, GPS, brújula o acelerómetros.
- Potente entorno de desarrollo que incluye un emulador, herramientas de depuración o un complemento para el IDE Eclipse.

Arquitectura Android

En la siguiente figura podemos ver un diagrama que muestra los principales componentes de la arquitectura de Android:

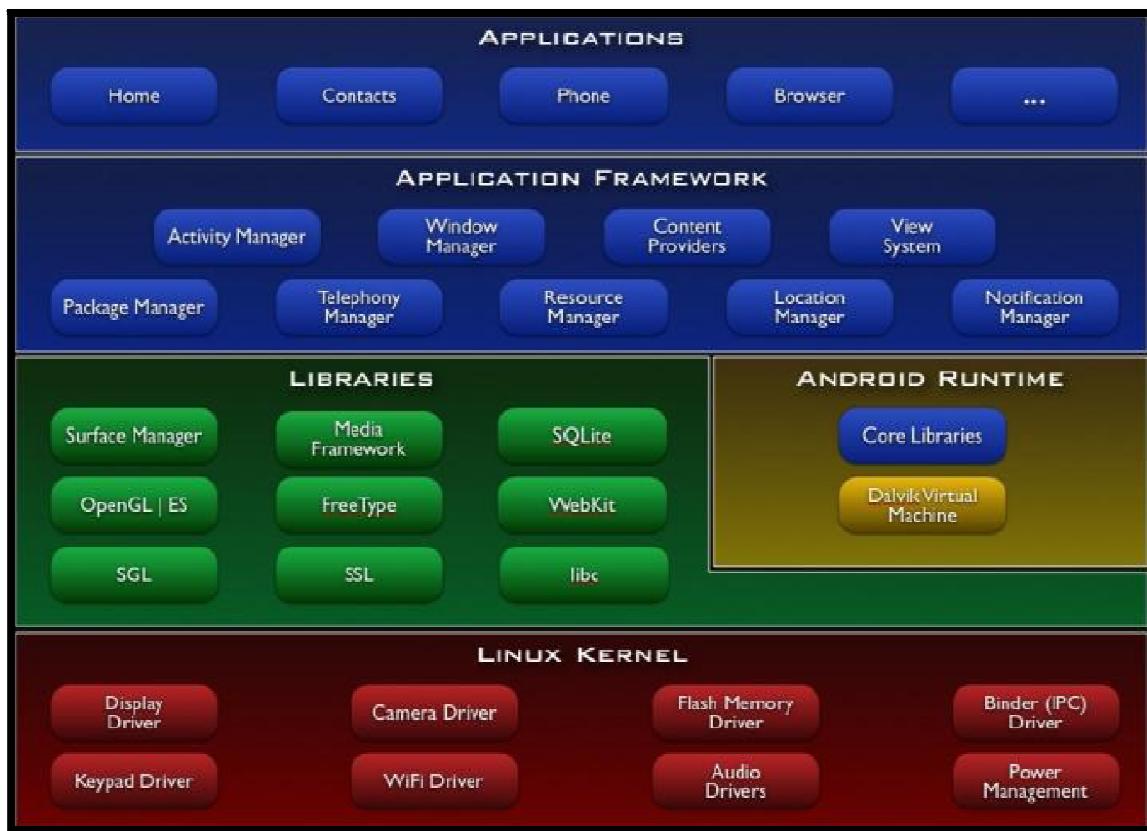


Ilustración 3.1.- Arquitectura de Android.

A continuación se detalla cada una de las partes de esta arquitectura:

Aplicaciones:

Android incluye una serie de aplicaciones básicas como un cliente de correo electrónico, un programa para mandar SMS, calendario, mapas, navegador web o contactos entre otros. Todas las aplicaciones están escritas en el lenguaje de programación Java y su desarrollo es posible gracias al Android SDK que provee de las herramientas e interfaces de programación necesarias.

Framework de aplicaciones:

En este nivel, situado entre las aplicaciones de usuario y las librerías del sistema, encontramos los siguientes servicios disponibles:

- *Gestor de Actividades*: se encarga de gestionar el ciclo de vida de las aplicaciones así como de establecer un sistema de navegación entre las mismas.
- *Gestor de Ventanas*: servicio que se encarga de ofrecer una interfaz de acceso al gestor de ventanas del propio sistema operativo.
- *Proveedores de Contenido*: que permiten a una aplicación acceder a los datos de otras aplicaciones, como por ejemplo los datos de los contactos, y a su vez compartir los datos de la propia aplicación con el resto.
- *Sistema de Vistas*: Ofrece un gran número de vistas que pueden ser usadas para desarrollar las aplicaciones. Las vistas disponibles van desde las listas o las galerías pasando por formularios para introducir texto, botones o un navegador web.

- *Gestor de Paquetes*: permite obtener información relativa a las aplicaciones que están instaladas actualmente en el dispositivo.
- *Gestor de Telefonía*: provee de acceso a la información relativa a los servicios de telefonía del dispositivo como por ejemplo el estado de los mismos (si se está realizando una llamada o no, etc.).
- *Gestor de Recursos*: da acceso a los recursos usados por las aplicaciones como por ejemplo las imágenes, las cadenas de texto mostradas o los archivos XML en los que se especifica el diseño de la interfaz.
- *Gestor de Localización*: este componente permite por ejemplo que las aplicaciones obtengan información sobre la localización geográfica del dispositivo y puedan lanzar algún evento en función de esta información.
- *Gestor de Notificaciones*: permite por ejemplo que una aplicación muestre una notificación en la barra de estrado.
- *Servicio XMPP*: API que sirve para poder acceder a este servicio de intercambio de mensajes en formato XML.

Librerías:

El conjunto de librerías de las que se compone Android es muy variado y extenso. Por un lado encontramos las librerías base de Android, escritas en Java y que ofrecen un conjunto de funcionalidades comunes a cualquier sistema operativo que pueden ser utilizadas en tiempo de ejecución por las aplicaciones. Estas librerías se encuentran empaquetadas en el archivo "android.jar" que se distribuye junto al SDK de Android. Por otro lado, Android incluye un conjunto de librerías escritas en C o C++ que son usadas por los distintos componentes que lo integran y que además pueden ser utilizadas por las aplicaciones a través del framework de aplicaciones.

Componentes de una aplicación Android

Las aplicaciones de Android pueden estar compuestas por uno o más de los siguientes componentes:

Actividades:

Las actividades se caracterizan por presentar una interfaz visual con la que los usuarios pueden interactuar para realizar una tarea. Una aplicación puede consistir de una sola actividad o de varias.

Cuando las aplicaciones cuentan con varias actividades, las cuales son totalmente independientes, se suele marcar una de ellas como la principal de forma que cuando se inicia la aplicación es esta actividad la que se presentan en primer lugar al usuario, y en el momento que se quiere cambiar a otra actividad sólo es necesario iniciarla desde la actual.

El sistema asigna una ventana para cada actividad aunque puede utilizar ventanas adicionales para mostrar por ejemplo una ventana emergente que requiere la interacción del usuario. El contenido visual de cada ventana está definido por una jerarquía de vistas, en la que cada una de ellas controla una zona de la interfaz y maneja las respuestas a las acciones del usuario.

Servicios:

Los servicios se caracterizan principalmente por dos cosas: la primera, que no presentan una interfaz visual de usuario, y segundo, que la tarea para la que están programados es ejecutada en segundo plano. Por ejemplo un reproductor de música puede hacer uso de un servicio que reproduzca la música mientras el usuario atiende otras actividades.

Cuando se inicia un servicio (o se registra para poder utilizarlo) se ofrece una interfaz de acceso al servicio que permite comunicarse con el mismo, en el caso de un reproductor de música esta interfaz ofrecerá métodos para controlar la reproducción como la pausa o el rebobinado.

Receptores de difusión

Los receptores de difusión son componentes con la función de recibir mensajes y reaccionar ante estos. Por ejemplo, cuando la batería del dispositivo está baja este componente informa al usuario.

Si bien no presentan una interfaz gráfica, si pueden invocar a una actividad en respuesta al mensaje recibido o incluso utilizar el gestor de notificaciones para alertar al usuario por ejemplo mediante una notificación en la barra de estado.

Proveedores de contenido

La principal función de estos componentes es poner a disposición de otras aplicaciones los datos que estamos utilizando en nuestras aplicaciones. Estos datos suelen estar almacenados en bases de datos SQLite.

Ciclo de vida de los componentes

En esta sección se comentarán los principales aspectos del ciclo de vida de los componentes de los que puede constar una aplicación.

Actividad

En una actividad existen fundamentalmente tres estados:

- *Activa o en ejecución*, cuando se encuentra en el primer plano de la pantalla, siendo la actividad el objetivo de las acciones del usuario.
- *En pausa*, si se ha perdido el foco de acción, pero sigue siendo visible para el usuario
- *Detenida*, si la actividad ha sido ocultada completamente por otra. Si una actividad está en pausa o detenida, el sistema puede terminar su ejecución, ya sea porque le pide finalizar o simplemente porque elimina su proceso.

La transición de un estado a otro se registra con las llamadas a los métodos *onCreate*, *onStart*, *onRestart*, *onResume*, *onPause*, *onStop* y *onDestroy*. Todos estos métodos se utilizan para realizar tareas específicas cuando cambia el estado de la actividad, como el caso de *onCreate* el cual deben implementar todas las actividades y que se utiliza para aplicar la configuración inicial de una nueva instancia de la actividad, o el método *onPause* que puede ser utilizado para guardar los cambios realizados sobre los datos.

En conjunto, estos métodos definen el ciclo de vida completo de una actividad, en el cual podemos diferenciar tres bucles de ejecución:

- Ciclo de vida completo de una actividad que sucede entre la primera llamada a `onCreate` y la llamada final a `onDestroy`. La actividad lleva a cabo su configuración inicial en la llamada a `onCreate`, y libera todos los recursos que quedan en `onDestroy`.
- Tiempo de vida visible de una actividad que ocurre entre una llamada a `onStart` hasta que se produce una llamada a `onStop`. Durante este tiempo, el usuario puede ver la actividad en pantalla, aunque puede que no sea en primer plano (interactuando con el usuario). Entre estos dos métodos, se pueden mantener los recursos que se necesitan para mostrar la actividad para el usuario.
- Tiempo de vida en primer plano de una actividad que ocurre entre una llamada a `onResume` hasta que se produce una llamada a `onPause`. Durante este tiempo, la actividad se encuentra en primer plano y el usuario está interactuando con la misma.

El siguiente diagrama ilustra estos bucles y los caminos que una actividad puede tomar entre los distintos estados. Los óvalos de color son los estados más importantes en los que se puede encontrar la actividad. Los rectángulos representan los métodos que se puede implementar para realizar operaciones cuando la actividad transita entre estados.

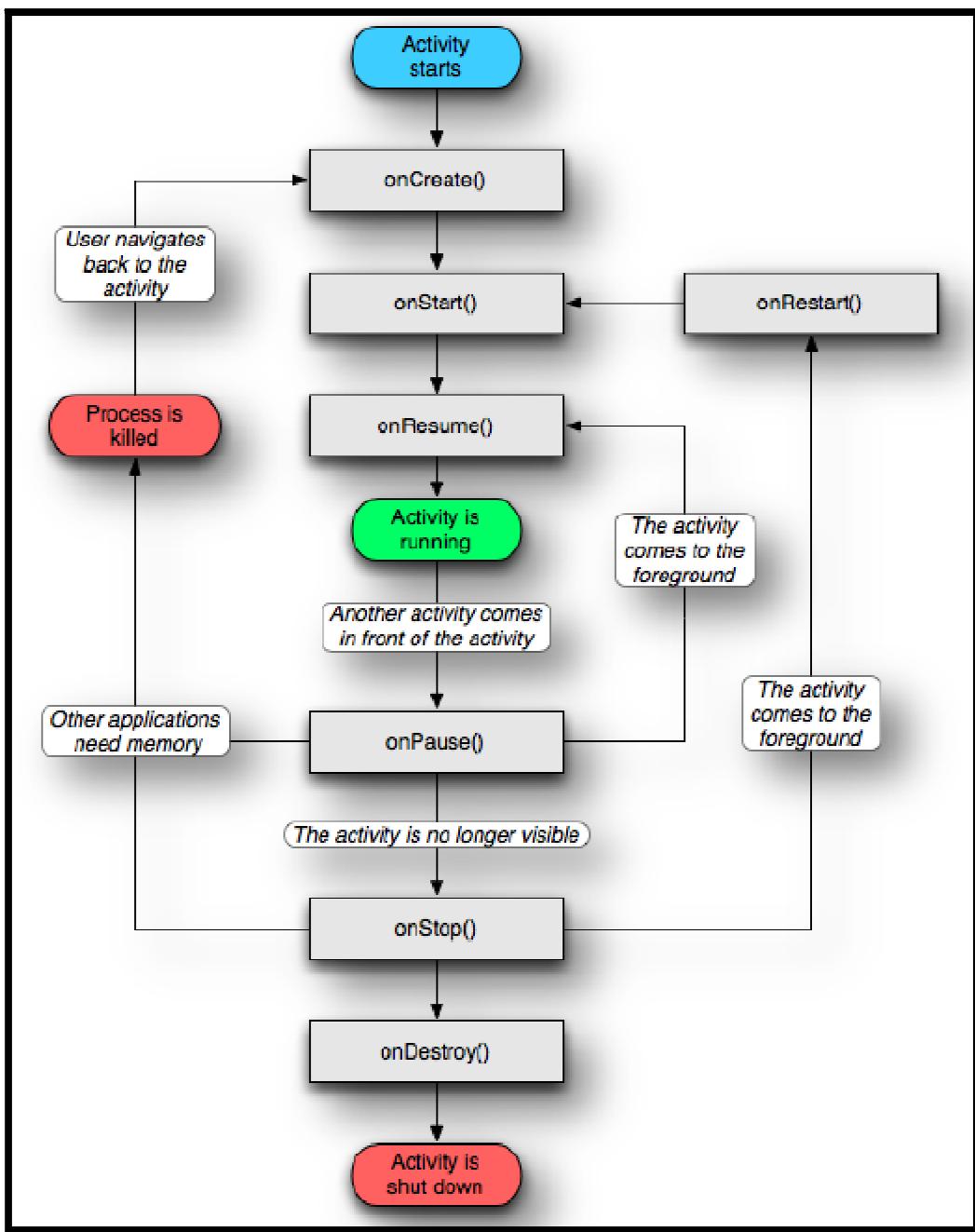


Ilustración 3.2.- Ciclo de vida de una actividad.

Cuando el sistema finaliza una actividad se debe tener en cuenta que cuando el usuario quiera volver a interactuar con dicha actividad esta debe presentarse tal y como se encontraba antes de ser expulsada de memoria por el sistema.

Para ello Android ofrece métodos dentro de su API que guardan el estado de la aplicación antes de que pueda ser eliminada (concretamente antes de la llamada a `onPause`) y recuperan el mismo cuando la actividad vuelve a tener el foco principal; aparte de objetos como `Bundle` con el que el estado de la aplicación es guardado como pares *nombre-valor*.

Servicio

Se puede acceder a un servicio de dos formas: iniciándolo y dejándolo ejecutar en segundo plano hasta que alguien lo pare o hasta que el mismo servicio finalice su ejecución con las funciones *startService* y *stopService*; y en segundo lugar, puede ser accedido usando una interfaz definida y exportada por el servicio, utilizando los métodos *bindService* y *unbindService*.

Al igual que las actividades, los servicios tienen unos métodos que definen su ciclo de vida, con los que se puede monitorizar las transiciones de un estado a otro, que son: *onCreate*, *onStart*, y *onDestroy*. De esta manera se definen dos bucles de ejecución para estos componentes:

- Ciclo de vida completo: queda definido entre la primera llamada que se hace a *onCreate* y el instante en que finaliza la llamada a *onDestroy*.
- Ciclo de vida activo: comienza con la llamada a *onStart*.

Mientras que *onCreate* y *onDestroy* son llamadas comunes a los servicios independientemente de cómo sean iniciados, *onStart* solo puede ser invocado por los servicios que fueron iniciados por *startService*. Así mismo si un servicio permite a otros componentes que accedan a él existen otros métodos que definen el estado de un servicio, y son: *onBind*, *onUnbind* y *onRebind*.

El siguiente diagrama muestra las transiciones entre estados de un servicio. Aunque en el diagrama se separan los servicios que son creados con una llamada a *startService* de los que son creados con una llamada a *bindService*, se debe tener en cuenta que un servicio puede permitir a otros componentes asociarse con él, por lo que cualquier componente de este tipo puede recibir llamadas *onBind* y *onUnbind*.

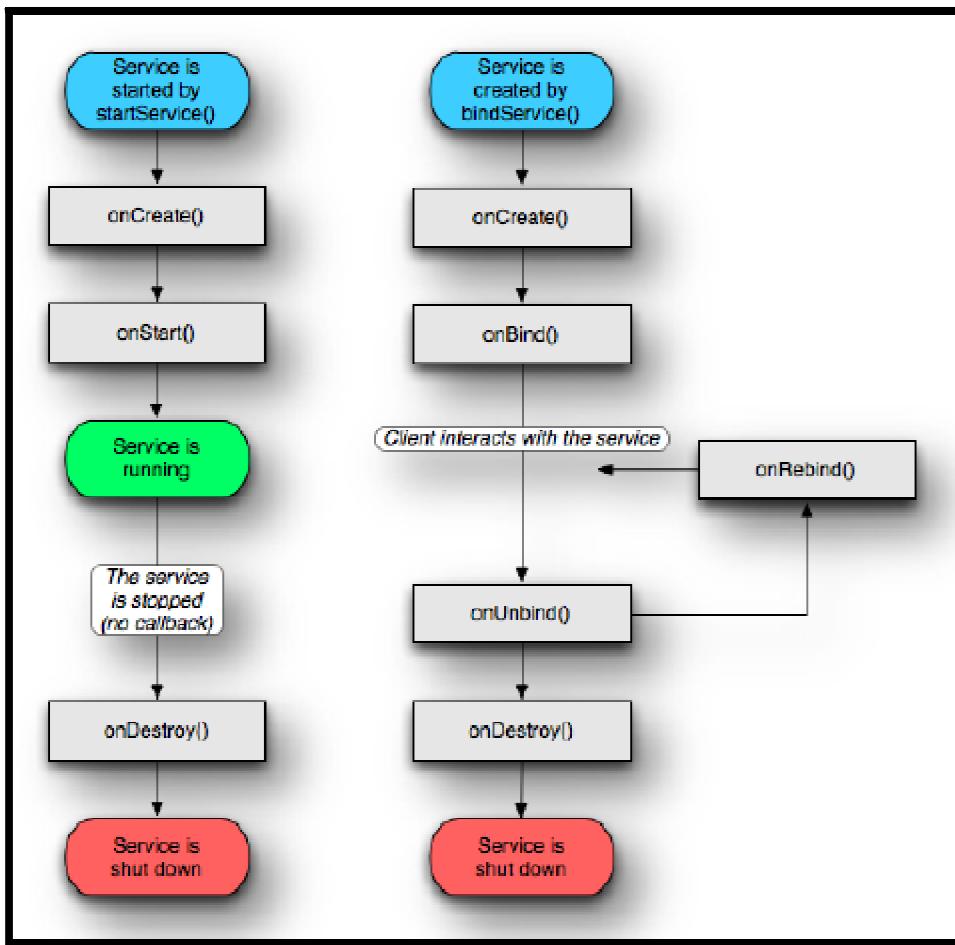


Ilustración 3.3.- Ciclo de vida de un servicio.

Receptores de difusión

Estos componentes tienen un solo método asociado con el ciclo de vida que es *onReceive*. Cuando llega un mensaje a un receptor de difusión se llama a este método pasándole dicho mensaje. En este momento, y sólo mientras ejecuta este método, el receptor se considera que está activo y por tanto un proceso que contenga a uno de estos componentes activos no podrá ser eliminado de la memoria del sistema.

Procesos

A la hora de determinar qué proceso debe ser eliminado para ceder sus recursos a otro, Android sitúa cada proceso dentro de una jerarquía de importancia basada en el número de componentes que están ejecutándose y el estado de los mismos, siendo eliminados primero los procesos con menor prioridad en la jerarquía para después ir suprimiendo los procesos de mayor prioridad si es necesario.

Esta jerarquía consiste en cinco niveles ordenados a continuación de mayor a menor importancia:

- *Primer plano*: aquellos procesos que son requeridos por la actividad del usuario. Están en este nivel de la jerarquía los procesos que cumplen una de las siguientes condiciones:
 - Están ejecutando una actividad con la que el usuario está interactuando.

- Contiene un servicio con el que el usuario está interactuando.
- Contiene un servicio que está ejecutando alguna de sus llamadas de ciclo de vida (*onCreate*, *onStart*, *onDestroy*).
- Contiene un receptor de difusión que está ejecutando el método *onReceive*.

En un mismo instante existen pocos procesos de este nivel de la jerarquía en memoria, los cuales son eliminados como último recurso cuando el sistema se ve obligado a realizar una re paginación de la memoria, por lo que la eliminación de estos procesos ayuda a que la interfaz responda a las acciones del usuario de forma apropiada.

- *Visibles*: son los procesos que no tienen componentes en primer plano pero que aún son visibles por el usuario. Cumplen alguna de estas condiciones:
 - Contiene una actividad que no está en primer plano pero aún es visible por el usuario (se ha invocado el método *onPause*).
 - Contiene un servicio que está enlazado a una actividad que es visible.

Estos procesos no son eliminados a no ser que sea necesario para que los procesos de primer plano sigan ejecutando.

- *Servicios*: son los procesos que se encuentran ejecutando un componente de este tipo pero no se ajustan a las prioridades superiores. No son visibles para el usuario aunque suelen llevar a cabo tareas importantes para este, por esto el sistema los mantiene en memoria hasta que es necesario que sean eliminados debido a que los procesos en primer plano o los visibles necesitan más memoria.
- *Segundo plano*: son los procesos que contienen actividades que no son visibles para el usuario (el método *onStop* ha sido invocado). Se pueden eliminar en cualquier momento de forma que los procesos de los niveles superiores de la jerarquía puedan obtener la memoria utilizada por los mismos. Normalmente hay varios procesos en segundo plano por lo que se mantiene una lista con los que contienen actividades que han sido usadas recientemente eliminándose por tanto en primer lugar los procesos más antiguos.
- *Vacíos*: son los procesos que no contienen ningún componente activo y que únicamente se mantienen en memoria como cache para futuras ejecuciones de forma que se mejore la rapidez con la que una aplicación inicia.

Android asigna siempre un proceso al nivel más elevado posible de prioridad, teniendo en cuenta ciertas consideraciones para ello, como por ejemplo la importancia de los componentes activos (en el caso de que un proceso contenga un servicio y una actividad visible, el proceso será asignado al nivel visible). Por otro lado la importancia de un proceso puede verse incrementada debido a que otros dependan del mismo, como por ejemplo en el caso de que un proceso esté sirviendo a otro, el servidor nunca debe ser asignado a un nivel de importancia menor que el cliente.

Por último, en función de lo comentado a cerca del ciclo de vida de los procesos, es recomendable pensar que tipo de componente debe ejecutar cierta acción dentro de una aplicación. Por ejemplo, se debería asignar un servicio a una operación de descarga, la cual se

supone va a consumir bastante tiempo, en lugar de una actividad para poder tener garantías de que la operación termina antes de que el proceso sea expulsado de la memoria.

Interfaces de usuario en Android

En Android la interfaz de usuario se construye básicamente con dos componentes: *View* y *ViewGroup*.

Cada elemento *View* contenido en la interfaz de usuario almacena información acerca de la disposición y el contenido de una región específica de la pantalla, gestionando distintos aspectos como el tamaño, los cambios en el foco de atención o las interacciones con esa zona específica de la pantalla por parte del usuario. Los objetos *View* sirven de base a otros elementos llamados *widgets*, los cuales ofrecen funcionalidades ya implementadas y que pueden ser utilizadas por la interfaz de usuario de una actividad, como por ejemplo un campo de texto o un botón. Por su parte los objetos *ViewGroup* sirven para definir la disposición o *layout* de los elementos que componen la interfaz.

Para definir la interfaz de usuario se debe establecer una jerarquía de objetos *View* y *ViewGroup* (ver Ilustración 3.4), utilizando para ello un conjunto de *widgets* y *layouts*, que pueden estar predefinidos o creados desde cero por el desarrollador.

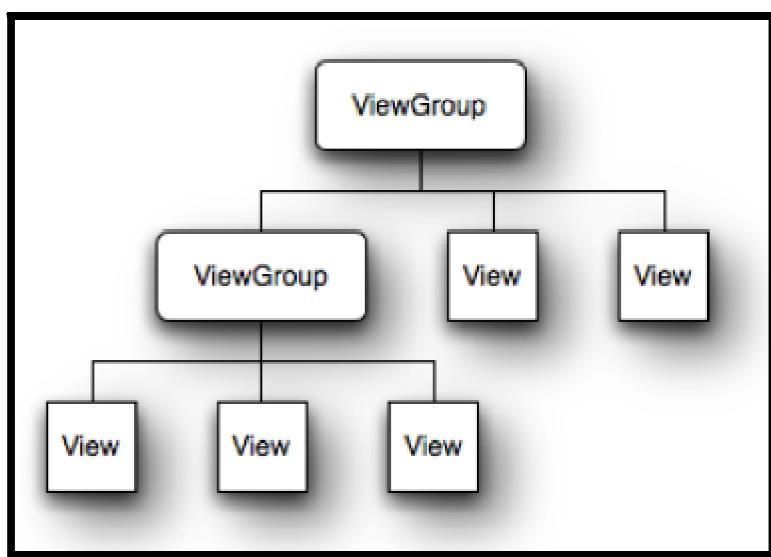


Ilustración 3.4.- Jerarquía de una interfaz de usuario.

Recursos

Una buena práctica de programación en Android consiste en referenciar los recursos (imágenes, textos, etc.) de una aplicación de forma externa a la misma, de esta manera el mantenimiento de estos recursos es totalmente independiente a la aplicación. Otra ventaja de realizar esta práctica es que se pueden definir recursos alternativos para distintas configuraciones de dispositivo como el lenguaje, tamaño de pantalla, etc.

Para ello, por un lado es necesario organizar los recursos empleados por la aplicación en distintos directorios bajo el directorio "res" que es el directorio raíz asignado a estos elementos. De esta forma, las imágenes se almacenan en un directorio distinto al de las cadenas de texto o de los ficheros que contienen el *layout* de una aplicación. Por otro lado,

cuando se desea definir recursos alternativos, para por ejemplo una imagen, se debe crear un directorio que identifique la configuración alternativa con la que el recurso va a ser utilizado.

El archivo AndroidManifest.xml

Se trata de un archivo obligatorio en toda aplicación Android. Contiene información esencial que el sistema debe conocer antes de poder ejecutar dicha aplicación. Este archivo, se utiliza, entre otras cosas, para lo siguiente:

- Define el nombre del paquete APK que sirve como identificador único para la aplicación.
- Describe los componentes de la aplicación –actividades, servicios, etc. Define además qué clases son las que implementan cada uno de estos componentes.
- Define qué procesos albergarán los componentes de la aplicación.
- Define los permisos que la aplicación debe poseer para acceder a determinados servicios o funciones del sistema o de otras aplicaciones.
- Define los permisos que otras aplicaciones deben poseer para utilizar los componentes de la aplicación.
- Define la versión mínima del API de Android que la aplicación necesita para funcionar
- Lista las librerías que la aplicación tiene enlazadas.

3.1.2. Lenguajes de programación

Java

El lenguaje de programación empleado en el desarrollo de la aplicación es JAVA. Se trata de un lenguaje de programación orientado a objetos desarrollado por Sun MicroSystems a mediados de los años 90.

La programación orientada a objetos se hizo popular por ser capaz de dividir programas largos en unidades semi-autónomas (18). El lema de la programación orientada a objetos es "divide y vencerás". En otras palabras, un programa se puede dividir en partes fácilmente identificables.

Los cinco principios fundamentales en los que se basa la programación en Java son:

- Utilización de la metodología de la programación orientada a objetos.
- Ejecución de un mismo programa en multitud de plataformas o sistemas operativos.
- Por defecto se debería poder dar soporte para trabajar en red.
- Uso de lo mejor de otros lenguajes como C++.

XML

XML (19), siglas en inglés de eXtensible Markup Language('lenguaje de marcas extensible'), es un metalenguaje extensible de etiquetas desarrollado por el World Wide Web Consortium (W3C).

XML no es realmente un lenguaje en particular, sino una manera de definir lenguajes para diferentes necesidades. Algunos de estos lenguajes que usan XML para su definición son XHTML, SVG, MathML.

XML es una tecnología sencilla que tiene a su alrededor otras que la complementan y la hacen mucho más grande y con unas posibilidades mucho mayores. Tiene un papel muy importante en la actualidad ya que permite la compatibilidad entre sistemas para compartir la información de una manera segura, fiable y fácil.

La tecnología XML busca dar solución al problema de expresar información estructurada de la manera más abstracta y reutilizable posible. Que la información sea estructurada quiere decir que se compone de partes bien definidas, y que esas partes se componen a su vez de otras partes. Entonces se tiene un árbol de trozos de información. Estas partes se llaman elementos, y se las señala mediante etiquetas.

Una etiqueta consiste en una marca hecha en el documento, que señala una porción de éste como un elemento. Un pedazo de información con un sentido claro y definido. Las etiquetas tienen la forma `<nombre>`, donde *nombre* es el nombre del elemento que se está señalando.

SQL

SQL(Structured Query Language), es un lenguaje de programación diseñado para manipular bases de datos relacionales (20). Como la mayor parte de los sistemas actuales son de este tipo y como el lenguaje SQL es el más ampliamente usado en éstos, se puede decir sin ningún género de dudas que hoy en día no tiene rival alguno (21).

Las principales ventajas que aporta SQL son dos:

- Su enorme difusión, pues es empleado en la mayoría de los sistemas actuales.
- Su elevada expresividad. Por ejemplo, operaciones que costarían semanas de duro esfuerzo en ser desarrolladas en un lenguaje de programación tradicional, pueden ser realizadas con SQL en tan sólo unos minutos.

Este lenguaje es un lenguaje de cuarta generación. Es decir, en este lenguaje se indica qué información se desea obtener o procesar, pero no cómo se debe hacer. Es labor interna del sistema elegir la forma más eficiente de llevar a cabo la operación ordenada por el usuario.

3.1.3. Eclipse

Se trata de un entorno de desarrollo integrado multiplataforma (22)(23) y de código abierto que permite desarrollar aplicaciones de escritorio, es decir, aplicaciones que se usan fuera del navegador. Originalmente Eclipse fue desarrollado por IBM aunque actualmente forma parte de una organización independiente llamada fundación Eclipse siendo un programa libre.

Eclipse consta de un editor de texto con resaltado de sintaxis y realiza la compilación del código en tiempo real pudiendo así mostrar los errores en el código instantáneamente. Dispone además de un apartado de plugins para añadir diferentes controles y forma de realizar los proyectos.

3.1.4. SQLite

Se trata de una librería que permite el acceso a la base de datos relacional que incluye Android. SQLite no presenta todas las características de los grandes productos de bases de datos cliente/servidor, pero incluye todo lo necesario para el almacenamiento local. Además, trabajar con ello es relativamente rápido y fácil (24) (25).

3.1.5. AndroidPlot

AndroidPlot es una API de Java para crear gráficos en aplicaciones Android. A diferencia de otras librerías multiplataforma que deben atender a un mínimo de requisitos en común, AndroidPlot está diseñada exclusivamente para Android (26).

A continuación puede verse una comparación de las librerías para diseñar gráficos actualmente disponibles de acuerdo con las características que cubre AndroidPlot:

Nombre	Soporta Datos Dinámicos	Documentación	Foro	Licencia	Precio
AndroidPlot	Sí	Wiki, tutoriales, Javadoc	Sí	BSD	Gratis
Android Chart	Desconocido	Ninguna	No	Comercial	\$299-\$999
Chartdroid	No	Wiki, tutoriales, código	Sí	Apache 2.0	Gratis
GraphView	No	No	Sí	Custom	Donación
RChart 2	Desconocido	Limitada	No	Custom	\$138.6
AChartEngine	Desconocido	Código fuente	Sí	Apache 2.0	Gratis
aiCharts	Sí	Ejemplos, Javadoc	Sí	Comercial	\$299 en adelante

Ilustración 3.5.- Tabla comparativa de las librerías de gráficos actuales.

En la tabla anterior, se observa cómo AndroidPlot es el que mejor se ajusta a nuestras necesidades por ser gratuito y presentar distintas fuentes de documentación.

3.1.6. SPSS

El programa SPSS (Statistical Package for Social Science) (27) constituye una potente aplicación estadística de la que, desde su versión inicial hace más de 35 años, se han desarrollado diferentes versiones (28). Inicialmente fue planteado para grandes sistemas informáticos, si bien, en la actualidad, se halla operativo en una gran cantidad de entornos (Unix, MsDos, Mac, Windows, etc.). De hecho las normas generales de su sintaxis y programación son las mismas, por lo que facilita que datos y programas sean fácilmente transportables de un entorno a otro.

3.2. Hardware utilizado

A pesar de la gran utilidad de los emuladores para ayudar en el desarrollo de las aplicaciones Android, todavía presentan algunas limitaciones como la simulación del acelerómetro. Gracias a que se disponía de un terminal Android, concretamente el HTC Desire HD, la aplicación ha podido desarrollarse y probarse.

Aunque se haya probado en un dispositivo concreto, al haberse desarrollado la aplicación para el sistema operativo móvil Android, cualquier dispositivo que tenga instalado Android podrá hacer uso de la misma.

Las características más relevantes para la aplicación del terminal empleado pueden verse a continuación. Para una visión más extensa de dichas características, puede consultarse la bibliografía (29):

- Tamaño:



Ilustración 3.6.- Tamaño del dispositivo HTC Desire HD.

- Pantalla:
 - Tipo: Pantalla táctil con zoom.
 - Tamaño: 10,92 cm.
 - Resolución: WVGA (480 x 800)

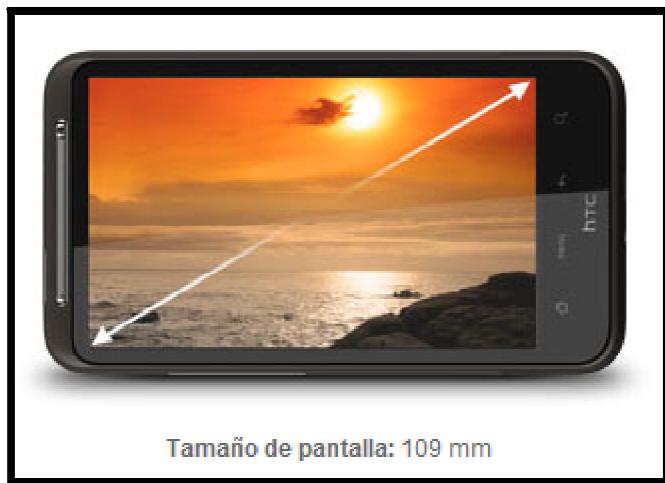


Ilustración 3.7.- Tamaño de la pantalla del HTC Desire HD.

- Velocidad de procesamiento de la CPU: 1GHz

- Memoria:
 - Memoria interna del teléfono: 1,5 GB
 - RAM: 768 MB.
 - Ranura de ampliación: Tarjeta de memoria microSD (compatible con SD 2.0).
- Plataforma: Android 2.2 (Froyo) con HTC Sense
- Conectores
 - Conector de audio estéreo de 3,5 mm
 - Micro USB estándar (micro USB 2.0 de 5 patillas)
- Sensores:
 - Acelerómetro
 - Brújula digital
 - Sensor de proximidad
 - Sensor de luz ambiental

De todas estas características, la más relevante es el sensor acelerómetro, ya que va a ser el que proporcione los datos para poder hacer el seguimiento de la actividad física en que se basa la aplicación.

En este dispositivo, la aceleración se mide a lo largo de tres ejes: lateral longitudinal y vertical (ver Ilustración 3.8). Dicho valor de la aceleración, se ve afectado por el valor de la gravedad que también se mide y que se suma a la medición.

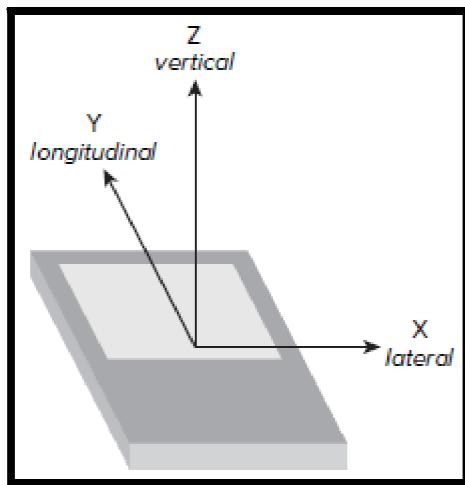


Ilustración 3.8.- Ejes de medida del acelerómetro.

4. Metodología de trabajo

En este capítulo se explica de forma detallada los pasos seguidos en el desarrollo del presente trabajo de fin de máster, empezando por un nivel más abstracto (diagramas de casos de uso y de secuencia) para finalizar en un nivel más concreto (implementación).

4.1. Especificación de requisitos

Teniendo en cuenta la problemática que se quiere paliar, el contexto donde se pretende dar cabida a la aplicación planteada y varios antecedentes en la temática, se han considerado diferentes posibilidades e ideas para alcanzar los objetivos propuestos. A continuación se describen brevemente las características principales que ha de cumplir la aplicación a desarrollar.

- Requisitos funcionales. En este subapartado se incluyen las funciones que debe cumplir el sistema. Estas son:
 - Configuración de los datos iniciales del usuario.
 - Puesta en marcha del acelerómetro.
 - Lectura e interpretación de los datos del acelerómetro.
 - Realimentación de información al usuario para que sea consciente de su evolución.
 - Envío de notificaciones para animar a seguir un estilo de vida saludable.
 - Mostrar consejos al usuario para que éste sepa la importancia de mejorar el estilo de vida.
 - El usuario deberá poder modificar su peso a medida que pase el tiempo. En función de ello se fijaran los objetivos a conseguir durante la semana.
- Requisitos no funcionales. Para poder instalar la aplicación y que funcione, el usuario debe disponer de un dispositivo móvil con sistema operativo Android de versión 2.2 o superior que incluya como parte de su hardware interno un sensor de acelerometría.

4.2. Fase de análisis

4.2.1. Diagrama de Casos de Uso

Un diagrama de casos de uso representa la vista funcional del sistema que se debe desarrollar (30). Cada caso de uso representa un requisito del sistema. El diagrama muestra las relaciones entre los actores y los casos de uso.

Los actores representan aquellos elementos que se comunican con el sistema pero que no forman parte de él. En nuestro caso (ver Ilustración 4.1), tenemos dos actores; los usuarios del sistema que actúan con él desencadenando la ejecución de operaciones y el sistema de almacenamiento. Existen varios sistemas de almacenamiento en la aplicación que entrará a comentarse en la parte de implementación.

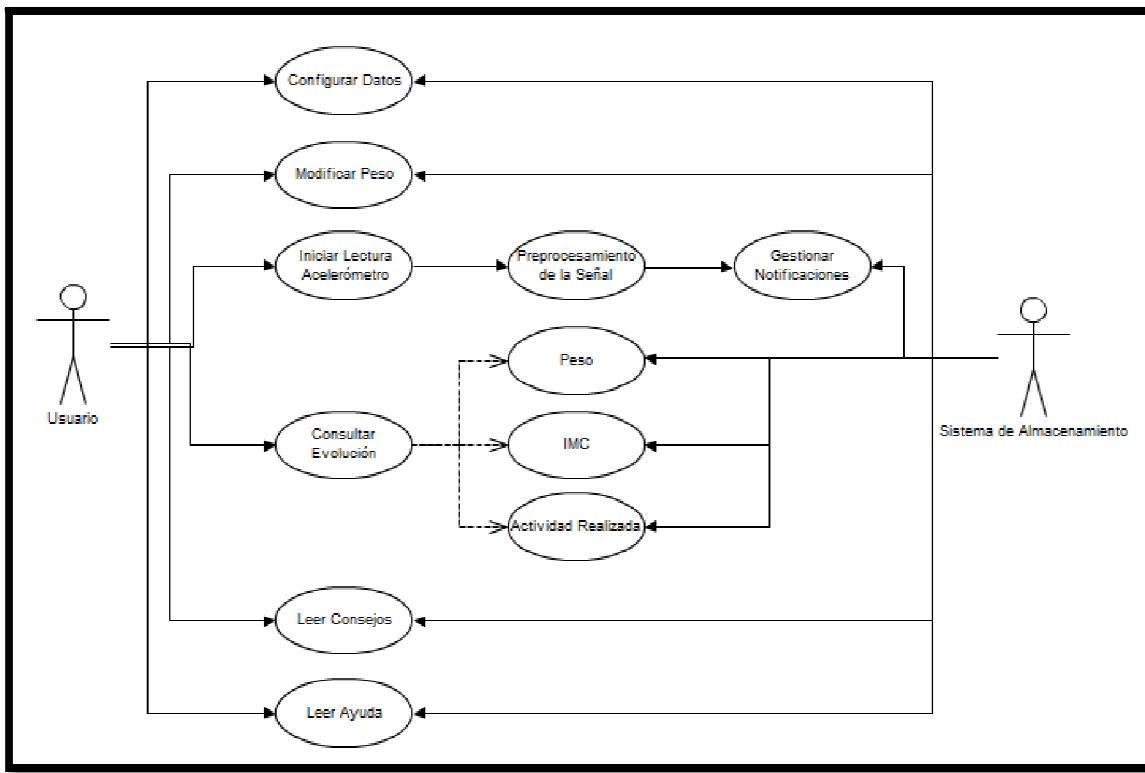


Ilustración 4.1.- Diagrama de casos de uso de la aplicación.

4.2.2. Descripción de Casos de Uso

En este apartado se describen más a fondo los distintos casos de uso de los que se compone la aplicación:

- *Configurar Datos*: De acuerdo al diagrama de casos de uso, *Configurar Datos* se relaciona con los dos actores existentes. El proceso llevado a cabo sería el siguiente: El usuario selecciona la opción de configuración en el menú. Como consecuencia, se abre la interfaz de configuración, donde el usuario introducirá los datos. Una vez completado el formulario, el usuario debe pulsar el botón *Aceptar* para que los datos queden registrados en el sistema de archivos.
- *Modificar Peso*: Esta funcionalidad permite al usuario introducir su peso en un determinado momento. Todos los datos introducidos serán almacenados por el sistema.
- *Iniciar Lectura Acelerómetro*: El usuario debe indicar al sistema que comience a hacer la lectura de los datos del acelerómetro. Este caso de uso hace que se inicie el de *preprocesamiento de la señal*.
- *Preprocesamiento de la señal*: En este caso de uso, y como su propio nombre indica, se lleva a cabo el preprocesamiento de la señal, de la que se sacan las características más relevantes para poder hacer una clasificación para determinar la actividad física realizada.
- *Gestionar Notificaciones*: Con la información obtenida del preprocesamiento de la señal y con unas pautas fijadas sobre el tipo de ejercicio a seguir, se generan una serie de notificaciones al usuario para que sea consciente de su evolución.
- *Consultar Evolución*: El usuario podrá consultar su evolución según tres criterios:

- *Peso*: Leyendo los datos del sistema de almacenamiento se genera un gráfico donde el usuario podrá ver la evolución de peso que ha ido teniendo.
 - *IMC*: Igual que el anterior pero con el IMC.
 - *Actividad Realizada*: Al igual que en los dos casos anteriores, se muestra al usuario la actividad física realizada cuyos datos se encuentran almacenados en el sistema.
- *Leer Consejos*: El usuario, al seleccionar esta opción en el menú, podrá leer algunos consejos sobre cómo reforzar un estilo de vida saludable. Es el sistema el que carga el fichero con los consejos una vez que el usuario selecciona la opción.
- *Leer Ayuda*: El objetivo de esta funcionalidad es mostrar al usuario las pautas básicas para el manejo de la aplicación. Al igual que en el caso anterior, una vez seleccionada la opción, el sistema carga el fichero con la ayuda de la aplicación.

4.3. Fase de diseño

4.3.1. Diagramas de Secuencia

Mediante los diagramas de secuencia se describen los flujos de eventos en términos de objetos y de mensajes pasados entre ellos. Se suele construir un diagrama de secuencia por cada flujo de eventos.

Configurar Datos

Como se muestra en la Ilustración 4.2, para configurar los datos iniciales, el usuario debe seleccionar la opción correspondiente en el menú. Inmediatamente, se abre la interfaz del formulario donde se deben llenar los datos. Una vez que el usuario termina de introducir los datos, debe pulsar el botón aceptar para que estos queden registrados. En función del peso y de la estatura, se calcula el IMC. Todos los datos, los introducidos por el usuario y los calculados, son almacenados por la aplicación.

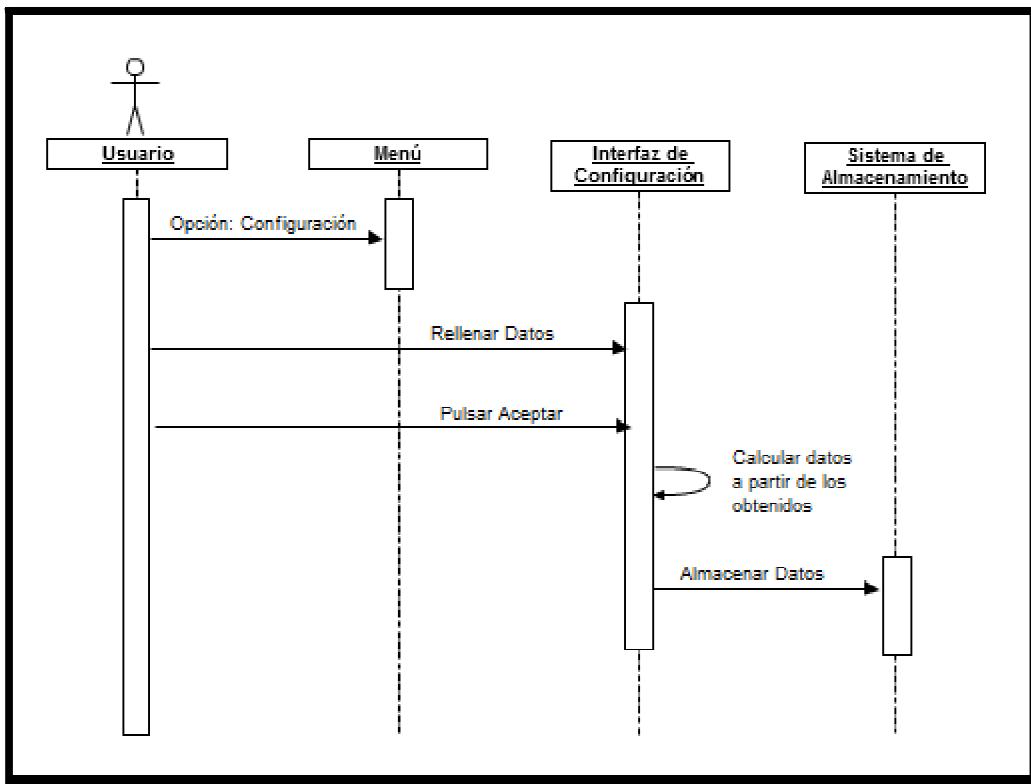


Ilustración 4.2.- Diagrama de secuencia "Configurar Datos".

Modificar Peso

En este diagrama de secuencia se observan los pasos necesarios para modificar el último peso almacenado en el sistema. Para ello, el usuario debe seleccionar la opción correspondiente tanto en el menú como en la interfaz de evolución. Una vez que introduce el nuevo peso, la aplicación calcula el nuevo IMC y almacena los datos en la base de datos.

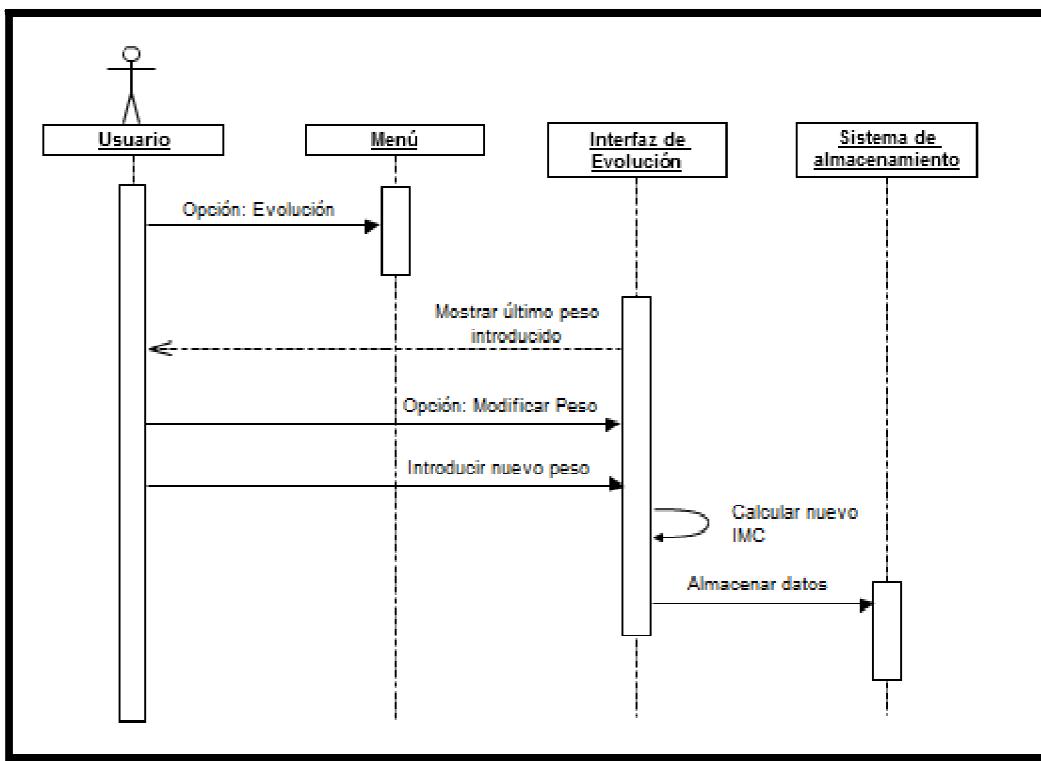


Ilustración 4.3.- Diagrama de secuencia "Modificar Peso".

Lectura e Interpretación de la Señal de Acelerometría

Para poner en marcha la lectura de los datos del acelerómetro, el usuario debe seleccionar la opción de "play" en el menú principal. Inmediatamente se pone en marcha la recogida de datos del sensor así como las dos alarmas necesarias en el proceso. Por un lado se tiene una alarma por cada minuto y por otro una cada día.

Cada minuto, se recogen todos los datos obtenidos del sensor y se llevan a cabo unas operaciones con ellos para determinar qué tipo de actividad se ha realizado en ese periodo de tiempo.

Con la alarma diaria, se pone en marcha un proceso que consiste en mirar los datos obtenidos cada minuto y determinar la cantidad de tiempo en el día, que el usuario ha pasado en reposo, realizando una actividad moderada o realizando una actividad intensa.

En función de los datos diarios obtenidos, comparándolos con los objetivos diarios, se envía una notificación al usuario para felicitarle si ha cumplido con los objetivos o para animarle si no los ha alcanzado.

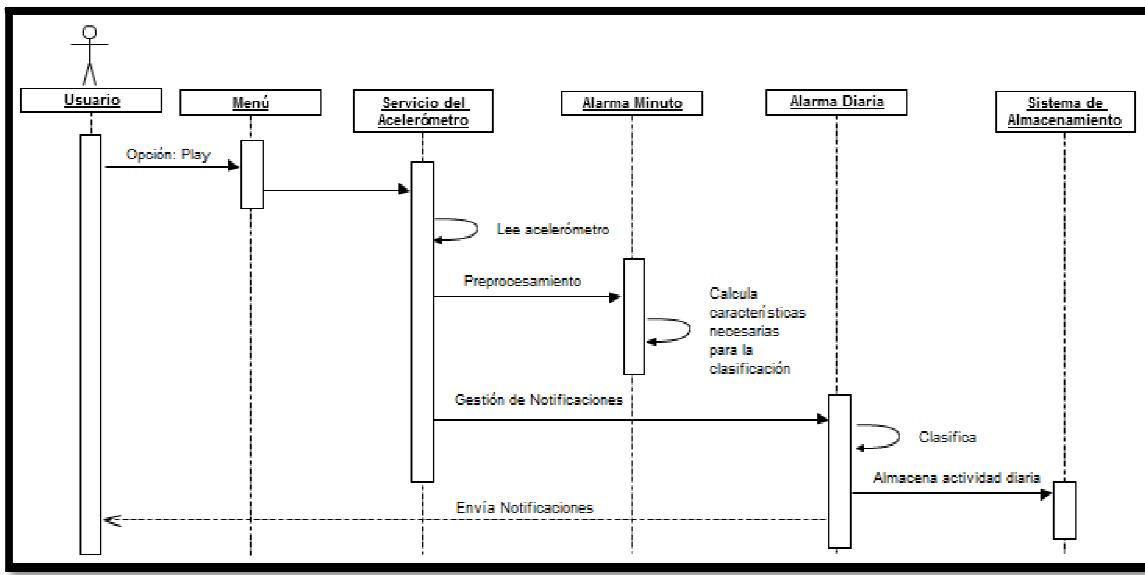


Ilustración 4.4.- Diagrama de secuencia "Lectura e Interpretación de la Señal de Acelerometría".

Consultar Evolución

En este diagrama de secuencia se ve el flujo de eventos seguido cuando un usuario desea ver su evolución. Lo primero es seleccionar la opción "evolución" en el menú principal y dentro de la interfaz que aparece, seleccionar el tipo de evolución que se desea consultar. En la Ilustración 4.5, la X representa cada uno de los tipos de evolución existentes (por peso, por IMC o por actividad), ya que los flujos seguidos en los tres casos son los mismos.

Las gráficas de evolución obtendrán los datos del sistema de almacenamiento.

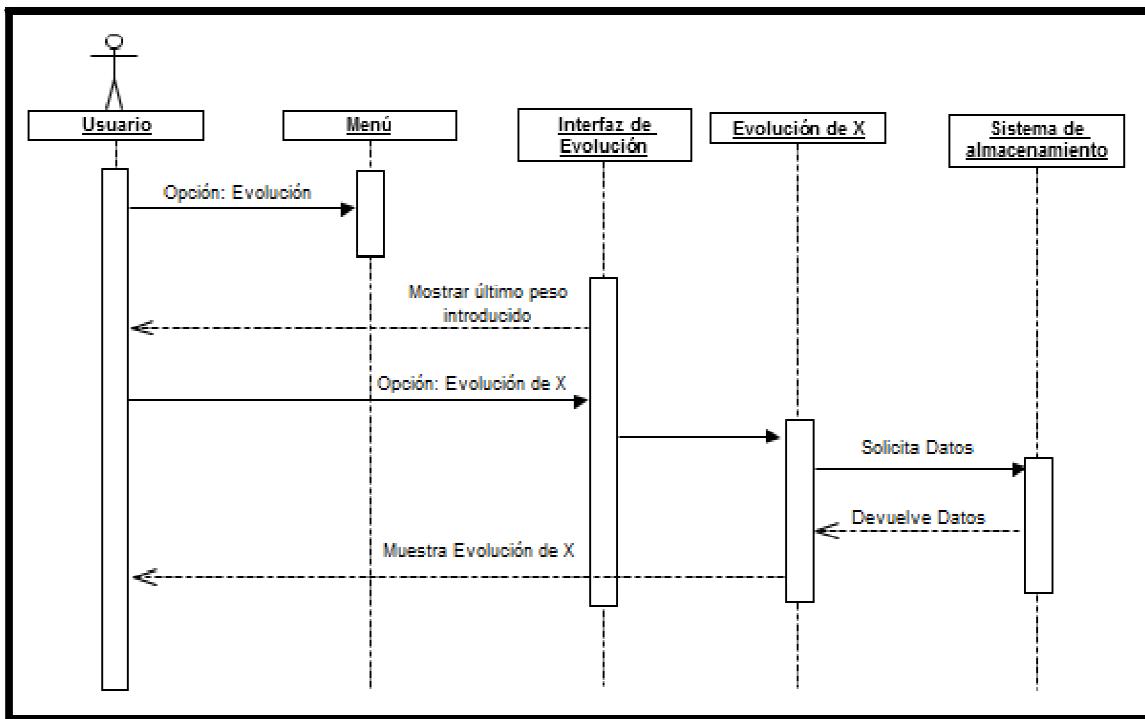


Ilustración 4.5.- Diagrama de secuencia "Consultar Evolución".

Leer Consejos/Ayuda

Tanto si el usuario quiere leer los consejos como la ayuda, deberá únicamente seleccionar la opción en el menú principal. Inmediatamente, el archivo requerido se lee y se muestra la información al usuario.

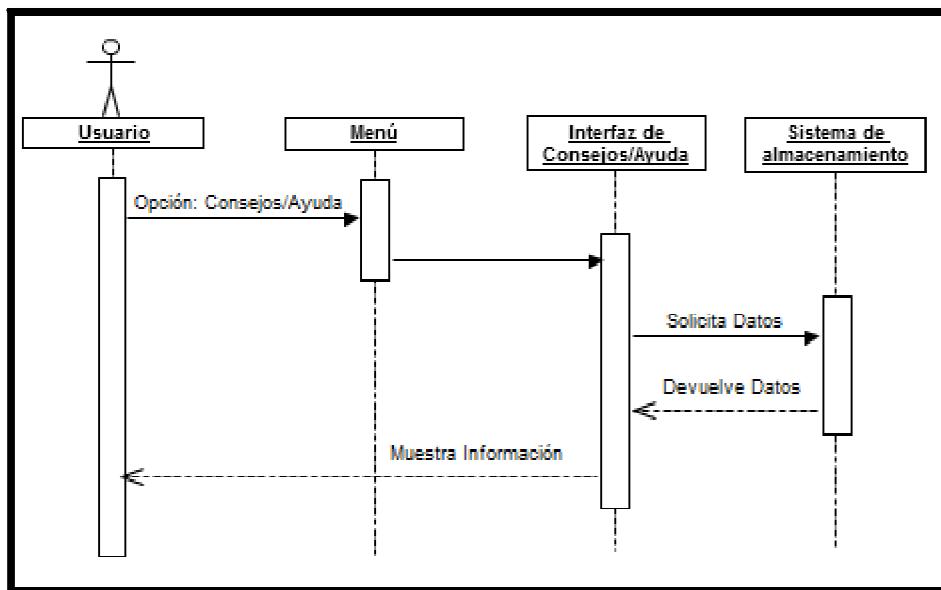


Ilustración 4.6.- Diagrama de secuencia "Leer Consejos/Ayuda".

4.3.2. Estructura de la aplicación

Las aplicaciones Android, siguen su propia arquitectura, por lo que se hace necesario comentarla para comprender el desarrollo de la aplicación.

Una aplicación Android se divide en una serie de subdirectorios (31):

- *Sources (src)*: Contiene una estructura que contiene los paquetes de la aplicación. En este caso, sólo se tiene un paquete, *com.android.tfm*, que contiene las siguientes clases:
 - *AccelerometerService.java*: Esta clase es la encargada de leer los datos procedentes del acelerómetro y de gestionar las dos alarmas necesarias para el funcionamiento de la aplicación. Por un lado una alarma que se repite cada minuto para hacer el preprocesamiento de los datos y por otro lado una alarma diaria para ver la evolución de la actividad a lo largo del día.
 - *ActivityEvolutionActivity.java*: Contiene los métodos para la inicialización de los componentes de la interfaz de la evolución de la actividad así como los necesarios para la recuperación de los datos para poder mostrar dicha evolución.
 - *AlarmReceiverDay.java*: Contiene los métodos necesarios para que cada día se almacene la actividad realizada, se compare con los objetivos, y se envíe una notificación al usuario para felicitarle si ha conseguido los objetivos o para animarle si este no ha sido el caso. Se trata de una clase del tipo *BroadcastReceiver*.

- *AlarmReceiverMinute.java*: Contiene los métodos de preprocesamiento de la señal a partir de los datos obtenidos por el acelerómetro. Al igual que la anterior, se trata de una clase del tipo *BroadcastReceiver*.
 - *BMIEvolutionActivity.java*: Contiene los métodos para recuperar los datos del IMC y mostrarlos en una gráfica.
 - *ConfigurationActivity.java*: Contiene los métodos para inicializar los componentes del formulario de configuración, así como los necesarios para almacenar los datos.
 - *GeneralActivity.java*: Se trata de una clase cuya única función es compartir variables con otras clases.
 - *HelpActivity.java*: Contiene los métodos para poder mostrar la ayuda al usuario.
 - *MeasureObject.java*: Se trata de un objeto que almacenará los datos de cada una de las medidas del acelerómetro.
 - *MenuActivity.java*: Contiene los métodos necesarios para inicializar el menú y gestionar qué ocurre cuando se selecciona cada una de las opciones.
 - *MessagesActivity.java*: Contiene los métodos para poder mostrar los consejos al usuario.
 - *MonitoringActivity.java*: Contiene los métodos necesarios para poder crear el menú para seleccionar el tipo de evolución que se desea consultar, así como los necesarios para modificar el peso y actualizar la base de datos.
 - *PreprocessingObject.java*: Se trata de un objeto que contiene los datos de preprocesado de cada minuto.
 - *ReadRawFileActivity.java*: Clase empleada para leer los ficheros del directorio */res/raw*.
 - *SplashActivity.java*: Contiene los métodos necesarios para inicializar la pantalla de inicio de la aplicación.
 - *TfmDbAdapter.java*: Clase encargada de la gestión de la base de datos.
 - *WeightEvolutionActivity.java*: Contiene los métodos para recuperar los datos del peso y mostrarlos en una gráfica.
- *Generated (gen)*: Contiene clases generadas automáticamente por el programa, concretamente la clase *R.java*, que contiene la versión en java de todos los recursos empleados en la aplicación.
- *Referenced Libraries*: Aquí se incluyen las librerías externas empleadas en el proyecto. En este caso se tiene la librería de gráficos *AndroidPlot*.
- *Assets*: Este subdirectorio contiene ficheros auxiliares necesarios para la aplicación como pueden ser ficheros de configuración, de datos, etc. En esta aplicación no se tiene ninguno de estos ficheros, por lo que dicha carpeta permanece vacía.
- *Libraries (lib)*: Contiene librerías. Para que una librería esté incluida en el proyecto, debe estar referenciada y por tanto incluida en el directorio *Referenced Libraries*, no es suficiente con que se incluya en esta carpeta.
- *Resources (res)*: Contiene los recursos empleados en la aplicación. Presenta varios subniveles. En concreto, para esta aplicación se cuenta con los siguientes:
 - *Anim*: Contiene dos ficheros .xml (*custom_anim.xml*, *fade_in.xml*) encargados de gestionar la animación de la pantalla principal.

- *Drawable*: Contiene las imágenes necesarias para la aplicación. Aunque para el presente trabajo no se ha tenido en cuenta, Se puede dividir en /drawable-ldpi, /drawable-mdpi y /drawable-hdpi para utilizar diferentes recursos dependiendo de la resolución del dispositivo.
- *Layout*: Contiene los ficheros de definición de la interfaz gráfica. El formato de estos ficheros es XML y a continuación se indican los empleados en la aplicación:
 - *activity_evolution.xml*: Interfaz que muestra la evolución de la actividad realizada.
 - *advices.xml*: Interfaz para mostrar los consejos al usuario.
 - *bmi_evolution.xml*: Interfaz que muestra la gráfica de la evolución del IMC.
 - *configuration.xml*: Interfaz del formulario de configuración de los datos.
 - *help.xml*: Interfaz que muestra la ayuda al usuario.
 - *main.xml*: Pantalla principal de la aplicación.
 - *menu.xml*: Interfaz que muestra el menú de la aplicación.
 - *monitoring.xml*: Interfaz donde se muestran las opciones para seleccionar las distintas formas de ver la evolución, así como la opción para modificar el peso del usuario.
 - *weight_evolution.xml*: Interfaz que muestra la gráfica de la evolución del peso.
- *Raw*: Contiene recursos adicionales, normalmente en formato distinto a XML, que no se incluyen en el resto de carpetas de recursos. La diferencia entre este directorio y el de *assets*, es que para éste, se genera un ID en la clase R. Sin embargo, para los recursos incluidos en la carpeta *assets*, no se genera ningún ID, por lo que se accede a ellos por su ruta, como cualquier otro fichero del sistema.
- *Values*: En este caso, contiene los siguientes recursos para la aplicación:
 - *arrays.xml*: Se corresponde con el array de cadenas de texto necesarias en el uso del *Spinner* (Género sin especificar, Masculino, Femenino).
 - *colors.xml*: Contiene las definiciones de los colores empleados en el diseño de la aplicación.
 - *dimens.xml*: Contiene las definiciones de las distintas dimensiones empleadas, como tamaños de letras, espacios, etc.
 - *strings.xml*: Contiene las cadenas de texto empleadas en la aplicación.
- *AndroidManifest.xml*: Se trata del archivo de configuración de la aplicación. Aquí deben declararse las actividades, servicios, permisos, etc. necesarios para que la aplicación funcione correctamente.

4.4. Detalles de implementación

A continuación se explican en detalle los distintos pasos llevados a cabo en la implementación.

4.4.1. Visión General

Para facilitar la comprensión de la aplicación, a continuación se muestra el flujo seguido por las clases más relevantes que componen la misma.

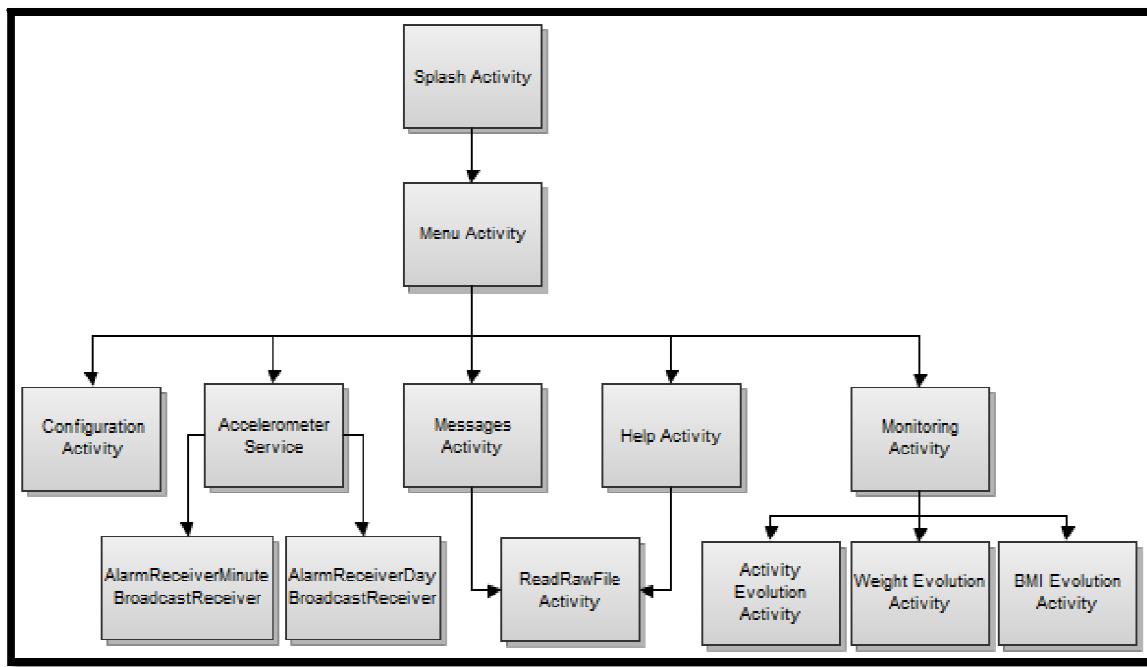


Ilustración 4.7.- Esquema general de la aplicación.

Como puede verse en la ilustración anterior, la aplicación comienza con la *Splash Activity*, es decir, con una pantalla inicial animada cuyo detalle de implementación puede consultarse en la sección 4.4.3. Cuando el usuario pulsa sobre cualquier parte de la pantalla, accede al menú de la aplicación (sección 4.4.5) desde donde podrá navegar por las distintas opciones de la aplicación.

Existen cinco opciones disponibles en el menú: configuración (*Configuration Activity*), play (*Accelerometer Service*), consejos (*Messages Activity*), ayuda (*Help Activity*) y evolución (*Monitoring Activity*).

En la configuración, el usuario completa el formulario y una vez que pulsa aceptar, los datos se almacenan terminándose el flujo de eventos.

El caso de la puesta en marcha de la aplicación es más complicado, ya que por un lado hay que tener un preprocesado cada minuto, y por otro una consulta de los datos diaria. Para conseguir que las distintas funciones se ejecuten en el tiempo determinado, es necesario hacer uso de dos nuevas clases: *AlarmReceiverMinute* y *AlarmReceiverDay*. Los detalles de esta implementación se comentan en apartados sucesivos.

Para que el usuario pueda leer los consejos y la ayuda, se hace necesario emplear la clase *ReadRawFile Activity*.

Por último, para el caso de la evolución del usuario, éste podrá seleccionar qué es lo que desea consultar. Puede elegir entre: evolución del peso (*WeightEvolution Activity*), del IMC (*BMIEvolution Activity*) o de la actividad (*ActivityEvolution Activity*).

4.4.2. Persistencia de los datos

Android ofrece distintos medios para almacenar datos (24), incluyendo acceso al sistema de archivos, una base de datos relacional a través de *SQLite* y un sistema de preferencias que permite almacenar datos de una forma sencilla de la forma clave/valor. En el desarrollo de la aplicación para el presente trabajo fin de máster, se ha hecho necesario el empleo de los dos de los métodos mencionados además del almacenamiento externo en la tarjeta SD. A continuación se pasa a comentar cada uno de ellos:

- *SharedPreferences*: Como se ha mencionado, se trata de un sistema de almacenamiento de datos de la forma clave/valor. Este método de almacenamiento se ha escogido para guardar los valores iniciales de configuración del usuario por el sencillo hecho de que los valores guardados de esta forma, permanecen ahí aunque se cierre la aplicación o se reinicie el móvil.
- *Almacenamiento externo (Tarjeta SD)*: Una de las ventajas de la plataforma Android es que ofrece acceso libre a la tarjeta de memoria SD. Se elige este método de almacenamiento para guardar los datos del preprocesamiento de la señal usados en el proceso de clasificación. Se hacía necesario disponer de un método que permitiese acceder a los datos de forma sencilla, por lo que se utilizaron ficheros de texto.
- *Base de datos SQLite*: Se decidió utilizar la base de datos que ofrece Android para almacenar los datos relacionados con las distintas modificaciones del peso y la actividad diaria realizada. Para ello, se ha creado una base de datos con dos tablas:
 - *Tabla weight_bmi*: Esta tabla contiene la información del peso del usuario, su IMC y el *timestamp* en el que se recogieron los datos.
 - *Tabla daily_activity*: Que recoge el día y la cantidad de actividad realizada de cada uno de los tipos de actividades posibles (reposo, actividad moderada o actividad intensa). Además presenta un campo que determina si la actividad realizada ha cumplido con los objetivos diarios y otro que contiene una referencia a la tabla de la actividad semanal para indicar en qué semana se encuentra.
 - *Tabla week_activity*: En esta tabla se incluyen los objetivos, tanto semanal como diario, el acumulativo de ejercicio realizado a medida que va transcurriendo la semana, un campo que indica si se ha cumplido el objetivo semanal y otro campo con la fecha del primer día de la semana.

4.4.3. Clasificación

A parte de diseñar e implementar la aplicación, se lleva a cabo un proceso de clasificación para determinar qué valores de qué características corresponden a cada actividad.

Para ello, se ha probado la aplicación realizando los distintos tipos de actividades y los datos obtenidos se han guardado en ficheros de texto para una posterior lectura empleando el siguiente código que almacena los ficheros generados en la tarjeta SD de la memoria externa.

```
/**  
 * Comprobar la disponibilidad del almacenamiento externo para leer  
 * o escribir.  
 */  
boolean mExternalStorageAvailable = false;  
boolean mExternalStorageWriteable = false;
```

```

String state = Environment.getExternalStorageState();
String string = title;
String fileName = "clasificacion_" + System.currentTimeMillis() +
".txt";

if (Environment.MEDIA_MOUNTED.equals(state)) {
    // We can read and write the media
    mExternalStorageAvailable = mExternalStorageWriteable = true;
} else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
    // We can only read the media
    mExternalStorageAvailable = true;
    mExternalStorageWriteable = false;
} else {
    // Something else is wrong. It may be one of many other states,
    but all we need
    // to know is we can neither read nor write
    mExternalStorageAvailable = mExternalStorageWriteable = false;
}

if(mExternalStorageWriteable ){
    // Se determina la ruta de almacenamiento
    File sdDir = Environment.getExternalStorageDirectory();

    if(sdDir.exists() && sdDir.canWrite()){
        //Creamos un directorio para este ejemplo.
        File uadDir = new File(sdDir.getAbsolutePath() + "/tfm");

        //Crea el directorio si no existe.
        uadDir.mkdir();

        if(uadDir.exists() && uadDir.canWrite()){
            //Determinar el nombre del archivo que deseamos
            crear.
            File file = new File(uadDir.getAbsolutePath() + "/" +
+ fileName);
            try{
                file.createNewFile();
            }catch(IOException e){

            }
            if(file.exists() && file.canWrite()){
                FileOutputStream fos = null;
                try{
                    fos = new FileOutputStream(file);
                    // Se recorre el vector y se añaden sus
valores al String
                    Iterator<PreprocessingObject> iterador =
AlarmReceiverMinute.preprocessing_list.listIterator();

                    //Mientras que el iterador tenga un
proximo elemento
                    while( iterador.hasNext() ) {
                        PreprocessingObject mObj =
(PreprocessingObject) iterador.next();

                        string += mObj.fft + "      " +
mObj.mean + "      " + mObj.std_dev + "\r\n";
                    }
                    fos.write(string.getBytes());
                } catch (FileNotFoundException e){

```

```

        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        } finally{
            if(fos != null){
                try{
                    fos.flush();
                    fos.close();

                    AlarmReceiverMinute.preprocessing_list.clear();
                }catch (IOException e) {
                    // TODO Auto-generated
                    e.printStackTrace();
                }
            }
        }
    }
}

```

Durante hora y media se alternaron las tres actividades cada cinco minutos, momento en el que se recogían y analizaban los datos. Se obtuvieron por tanto tres ficheros, uno por cada actividad que presentaban seis mediciones correspondientes a cada uno de esos cinco minutos. Los datos recogidos para cada una de las actividades (reposo, andando o intensidad moderada y corriendo o actividad intensa) se muestran a continuación:

En la imagen siguiente (Ilustración 4.8), se ven los datos recogidos en reposo. Llama la atención que cada primera medida de cada uno de los cinco minutos, presenta una diferencia significativa con el resto de medidas, no solo de su grupo, sino con las de todo el fichero. Esto se debe a que las actividades se realizaron alternando unas con otras, por lo que las primeras medidas podrían verse afectadas por la actividad anterior como ocurre en este caso.

MEDIA	DESV ESTANDAR
0.9248979	1.5890523
0.10569473	0.027223019
0.08228983	0.016981917
0.04532844	0.012828722
0.041460134	0.014009631
MEDIA	DESV ESTANDAR
4.800104	3.841687
0.1169645	0.014670374
0.0661492	0.030273683
0.047032285	0.019365165
0.06941489	0.047324594
MEDIA	DESV ESTANDAR
4.8249636	2.7364783
1.5938989	0.41399696
0.66060334	0.12883343
0.3738974	0.06539298
0.15035701	0.033423763
MEDIA	DESV ESTANDAR
4.0860415	3.2504296
0.7375397	0.16925856
0.274348	0.04855586
0.16262472	0.020412961
0.056177907	0.019510256
MEDIA	DESV ESTANDAR
4.982903	3.4238853
0.3035807	0.09475875
0.14543621	0.067540556
0.075634815	0.0032029008
0.060616225	0.046132136
MEDIA	DESV ESTANDAR
4.837439	3.7812433
0.23513085	0.06932291
0.16246676	0.0061673447
0.09237712	0.03131597
0.050878037	0.011237173

Ilustración 4.8.- Datos recogidos en reposo.

A continuación, Ilustración 4.9 e Ilustración 4.10, se muestran los datos recogidos andando y corriendo respectivamente. En estos casos, las medidas no se ven tan afectadas por la actividad anterior como ocurría con el reposo.

MEDIA	DESV_ESTANDAR
2.7488139	2.2055173
2.9712253	0.8739555
2.508085	1.4554341
1.6047604	0.38290682
2.4414623	0.0
MEDIA	DESV_ESTANDAR
2.2908726	2.2246716
3.1741812	1.2720947
2.6056166	1.224166
2.4775274	2.062576
1.7506784	0.61359715
MEDIA	DESV_ESTANDAR
2.6548002	2.3415158
1.7264351	0.49647948
2.4869611	0.6811983
3.2960668	2.6955793
2.534628	0.82925236
MEDIA	DESV_ESTANDAR
2.181421	2.315912
2.3264122	0.9417737
2.4069097	1.137165
1.3738073	0.597321
3.162572	0.993867
MEDIA	DESV_ESTANDAR
2.2425046	2.1085207
2.618914	0.8663008
2.538122	0.9977069
3.4820888	0.8041868
1.9212477	0.952509
MEDIA	DESV_ESTANDAR
2.474635	2.0589643
2.0050077	0.9849853
2.5229378	1.6499615
2.0889103	1.0240088
2.4213836	1.1911223

Ilustración 4.9.- Datos recogidos andando.

MEDIA	DESV_ESTANDAR
5.8514757	3.357333
5.284623	2.9221985
4.017442	0.40279758
4.9064856	1.5723249
6.145098	1.9260426
MEDIA	DESV_ESTANDAR
5.853578	3.4333577
7.563674	3.279047
7.478212	4.356214
5.495072	2.4036202
5.3862653	1.3722879
MEDIA	DESV_ESTANDAR
4.8727136	2.623879
5.0047297	2.789728
6.926898	2.3439538
6.78003	2.896386
7.392401	3.3230917
MEDIA	DESV_ESTANDAR
5.4841084	3.3200884
6.8264084	3.162035
5.9456053	2.1320322
7.187441	0.2600243
7.214802	3.5891674
MEDIA	DESV_ESTANDAR
5.0771484	3.0979471
6.963624	3.717399
7.425811	2.6762736
6.692784	1.5051835
5.7632284	2.084843
MEDIA	DESV_ESTANDAR
5.6833487	2.800207
8.283475	2.932547
7.440379	3.076233
8.413986	3.1818454
8.911522	3.8656623

Ilustración 4.10.- Datos recogidos corriendo.

Una vez con los datos obtenidos, se realizó un análisis estadístico de los mismos para determinar los umbrales que determinan qué actividad realiza el usuario en cada momento. Para ello se empleó el programa de análisis estadístico SPSS.

En el análisis de los datos, se cuenta con todos los valores de las actividades andando y corriendo, sin embargo, para las de reposo, como se comentó anteriormente, existen mediciones que se ven afectadas, por lo que deciden eliminarse del estudio para que no se contaminen los resultados. A continuación se muestra un resumen de los datos empleados:

Actividad	Casos						
	Válidos		Perdidos		Total		
	N	Porcentaje	N	Porcentaje	N	Porcentaje	
Media_min	Reposo	23	100,0%	0	,0%	23	100,0%
	Andar	30	100,0%	0	,0%	30	100,0%
	Correr	30	100,0%	0	,0%	30	100,0%
DesvT_min	Reposo	23	100,0%	0	,0%	23	100,0%
	Andar	30	100,0%	0	,0%	30	100,0%
	Correr	30	100,0%	0	,0%	30	100,0%

Ilustración 4.11.- Resumen del procesamiento de los casos.

El primer paso para llevar a cabo un análisis estadístico adecuado con variables cuantitativas, es comprobar su normalidad, es decir, determinar si las variables de estudio

siguen una distribución normal. Estas pruebas de normalidad, se realizan por grupos (reposo, andando y corriendo) empleando dos técnicas distintas: Kolmogorov-Smirnov y Shapiro-Wilk. Los resultados pueden verse en la Ilustración 4.12:

Actividad	Kolmogorov-Smirnov ^a			Shapiro-Wilk		
	Estadístico	gl	Sig.	Estadístico	gl	Sig.
Media_min	Reposo	,274	23	,000	,709	23
	Andar	,126	30	,200(*)	,970	30
	Correr	,117	30	,200(*)	,972	30
DesvT_min	Reposo	,204	23	,014	,806	23
	Andar	,171	30	,025	,930	30
	Correr	,146	30	,102	,933	30

* Este es un límite inferior de la significación verdadera.
a Corrección de la significación de Lilliefors

Ilustración 4.12.- Pruebas de normalidad.

Observando la actividad "reposo" para cada una de las dos variables, se llega a las siguientes conclusiones:

- En el caso de *media_min*, se determina que la actividad "reposo" no sigue una distribución normal ya que su valor de sigma para las dos pruebas es 0,000, que es inferior al 0,05 habitual. Se dice por tanto que los datos en reposo, no siguen una distribución normal a cualquier nivel de confianza (95%, 99%, etc.).
- Para *desvT_min*, se llega a la misma conclusión que con *media_min*. Con Kolmogorov-Smirnov, se obtiene de nuevo un valor de sigma igual a 0,000, lo que determina que a cualquier nivel de confianza, la actividad de reposo no sigue una distribución normal. En el caso de Shapiro-Wilk, sigma vale 0,014. Aunque sea un valor mayor que los anteriores, sigue siendo menor de 0,05, por lo que se concluye que los datos en reposo no siguen una distribución normal.

Para que se pueda hacer un análisis paramétrico de los datos, los datos deben cumplir los dos enunciados siguientes:

- Criterio de normalidad. La variable cuantitativa debe distribuirse según la Ley Normal en cada uno de los grupos que se comparan.
- Criterio de homocedasticidad. Las varianzas de las distribución de la variable cuantitativa en las poblaciones de las que provienen los grupos que se comparan, deben ser homogéneas.

Como se ha demostrado, la actividad "reposo", no sigue una distribución normal, por lo que no se puede realizar un análisis paramétrico (t de Student o ANOVA). En su lugar, debe realizarse un análisis NO PARAMÉTRICO de K muestras independientes.

Lo primero será determinar si existen diferencias significativas entre las variables de estudio para los distintos grupos. Para ello, se realiza un contraste de la hipótesis de la igualdad de medias:

Estadísticos de contraste(a,b)		
	Media_min	DesvT_min
Chi-cuadrado	72,522	60,394
Grados libertad	2	2
Sig. asintót.	,000	,000

a Prueba de Kruskal-Wallis
b Variable de agrupación: Actividad

Ilustración 4.13.- Contraste de hipótesis de igualdad de medias.

A la vista del valor de sigma (0,000), se puede afirmar que las dos variables de estudio rechazan la hipótesis nula de que los grupos presenten igualdad de medias.

$$H_0 = \mu_{repose} = \mu_{caminar} = \mu_{correr}$$

Por lo tanto, ambas variables son útiles para clasificar la pertenencia a una actividad.

Se ha determinado que las medias de los distintos grupos, son diferentes entre sí. A continuación, se estudia si esto también ocurre cuando se analizan los grupos por parejas:

Variable dependiente	(I) Actividad	(J) Actividad	Diferencia de medias (I-J)	Error típico	Sig.	Intervalo de confianza al 95%	
						Límite superior	Límite inferior
Media_min	HSD de Tukey	Reposo	Andar	2,2556763(*)	,2173579 ,000	2,774749	1,736603
			Correr	6,2301223(*)	,2173579 ,000	6,749195	5,711049
		Andar	Reposo	2,2556763(*)	,2173579 ,000	1,736603	2,774749
			Correr	3,9744461(*)	,2024959 ,000	4,458027	3,490865
		Correr	Reposo	6,2301223(*)	,2173579 ,000	5,711049	6,749195
			Andar	3,9744461(*)	,2024959 ,000	3,490865	4,458027
	Games-Howell	Reposo	Andar	2,2556763(*)	,0974950 ,000	2,493138	2,018214
			Correr	6,2301223(*)	,2218573 ,000	6,776290	5,683955
		Andar	Reposo	2,2556763(*)	,0974950 ,000	2,018214	2,493138
			Correr	3,9744461(*)	,2359431 ,000	4,549607	3,399285
		Correr	Reposo	6,2301223(*)	,2218573 ,000	5,683955	6,776290
			Andar	3,9744461(*)	,2359431 ,000	3,399285	4,549607
DesvT_min	HSD de Tukey	Reposo	Andar	1,2227282	,1955551 ,000	1,689734	-,755722
			Correr	2,6367449(*)	,1955551 ,000	3,103751	2,169739
		Andar	Reposo	1,2227282(*)	,1955551 ,000	,755722	1,689734
			Correr	1,4140167(*)	,1821839 ,000	1,849091	-,978943
		Correr	Reposo	2,6367449(*)	,1955551 ,000	2,169739	3,103751
			Andar	1,4140167(*)	,1821839 ,000	,978943	1,849091
	Games-Howell	Reposo	Andar	1,2227282(*)	,1256845 ,000	1,532971	-,912485
			Correr	2,6367449(*)	,1734606 ,000	3,065020	2,208470
		Andar	Reposo	1,2227282(*)	,1256845 ,000	,912485	1,532971
			Correr	1,4140167(*)	,2138640 ,000	1,929744	-,898289
		Correr	Reposo	2,6367449(*)	,1734606 ,000	2,208470	3,065020
			Andar	1,4140167(*)	,2138640 ,000	,898289	1,929744

• La diferencia de medias es significativa al nivel .05.

Ilustración 4.14.- Contraste de igualdad de medias por parejas.

Como se observa en la tabla anterior (Ilustración 4.14), cuando se comparan los grupos por parejas, también existen diferencias de medias a cualquier nivel, ya que sigma es 0,000 en todos los casos para los dos procedimientos empleados:

- HSD Tukey que asume varianzas iguales.
- Games-Howell que no asume varianzas iguales.

Por tanto, tras realizar el análisis completo de las variables cuantitativas, se puede afirmar que ambas son válidas para diferenciar grupos de actividades.

Una vez se ha llegado a esta conclusión, el siguiente paso es determinar los umbrales que separan las tres actividades. Para ello, se obtiene la siguiente tabla (Ilustración 4.15), donde puede verse el valor medio y los límites de los intervalos correspondientes a cada una de las actividades para las dos variables en estudio:

Actividad				Estadístico	Error típ.
Media_min	Reposo	Media		,178957	,0390947
		Intervalo de confianza para la media al 95%	Límite inferior	,097879	
			Límite superior	,260034	
	Andar	Media		2,434633	,0893134
		Intervalo de confianza para la media al 95%	Límite inferior	2,251966	
			Límite superior	2,617299	
DesvT_min	Correr	Media		6,409079	,2183856
		Intervalo de confianza para la media al 95%	Límite inferior	5,962430	
			Límite superior	6,855728	
	Reposo	Media		,043380	,0085834
		Intervalo de confianza para la media al 95%	Límite inferior	,025579	
			Límite superior	,061181	
DesvT_min	Andar	Media		1,266108	,1253910
		Intervalo de confianza para la media al 95%	Límite inferior	1,009655	
			Límite superior	1,522562	
	Correr	Media		2,680125	,1732481
		Intervalo de confianza para la media al 95%	Límite inferior	2,325793	
			Límite superior	3,034457	

Ilustración 4.15.- Valores medios y límites de los intervalos para cada una de las actividades.

A la vista de las medidas e intervalos de confianza de las variables, se sabe que es fácil establecer umbrales entre los grupos puesto que no existe solapamiento entre los mismos, es decir, podemos encontrar los umbrales simplemente calculando los dos puntos medios existentes entre los tres intervalos de confianza (uno por cada grupo).

- Para la variable *media_min*:
 - Umbral que separa reposo y andar:

$$\text{Umbral}_{\text{reposto-andar}} = \text{Reposto}_{\text{lim. superior}} + \frac{\text{Andar}_{\text{lim.inferior}} - \text{Reposto}_{\text{lim.superior}}}{2}$$

$$= \text{Reposto}_{\text{lim. superior}} + 0.995966 = 1.256$$

- Umbral que separa andar y correr:

$$\text{Umbral}_{\text{andar-correr}} = \text{Andar}_{\text{lim. superior}} + \frac{\text{Correr}_{\text{lim.inferior}} - \text{Andar}_{\text{lim.superior}}}{2}$$

$$= \text{Andar}_{\text{lim. superior}} + 1.6725 = 4.2898$$

- Para la variable *desvT_min*:
 - Umbral que separa reposo y andar:

$$\text{Umbral}_{\text{reposto-andar}} = \text{Reposto}_{\text{lim. superior}} + \frac{\text{Andar}_{\text{lim.inferior}} - \text{Reposto}_{\text{lim.superior}}}{2}$$

$$= \text{Reposto}_{\text{lim. superior}} + 0.474237 = 0.5354$$

- Umbral que separa andar y correr:

$$Umbral_{andar-correr} = Andar_{lim. superior} + \frac{Correr_{lim.inferior}-Andar_{lim.superior}}{2} = \\ Andar_{lim. superior} + 0.4016 = 1.9241$$

En vista de los resultados obtenidos, se determina que cualquiera de las variables podría ser empleada para determinar el umbral de las actividades, sin embargo, observando las gráficas que muestran las medias de cada variable respecto a las actividades, se decide decantarse por el empleo de la desviación típica como medida de umbral ya que la diferenciación de clases es ligeramente mejor:

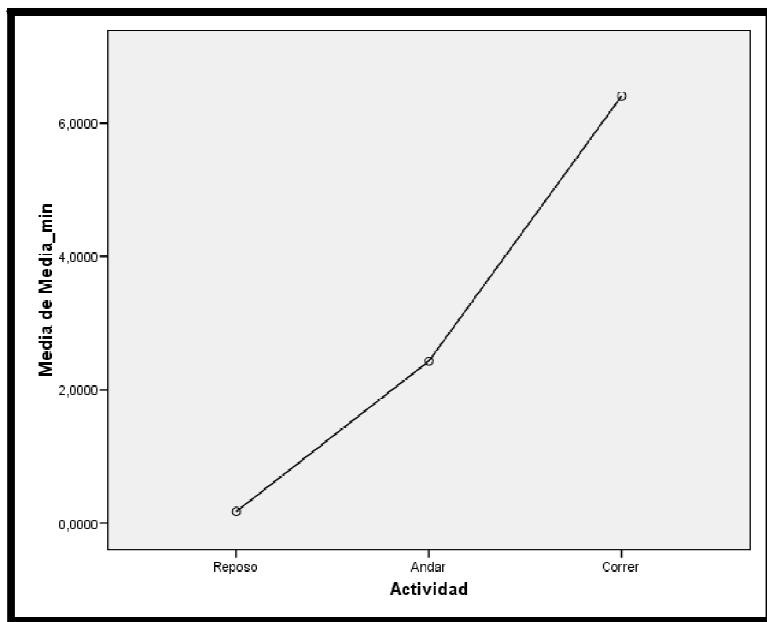


Ilustración 4.16.- Representación de la media de la variable media_min respecto a los tres grupos de actividad.

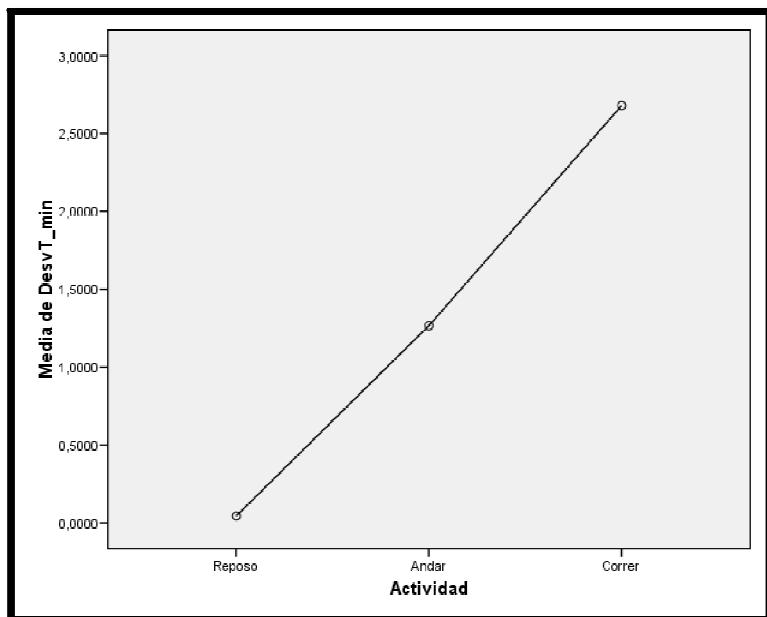


Ilustración 4.17.- Representación de la media de la variable desvT_min respecto a los tres grupos de actividad.

4.4.4. Página inicial

En cada pantalla existen dos partes claramente diferenciadas. Por un lado se tiene la interfaz gráfica que se crea con XML y por otro el propio código de la pantalla implementado en java.

Para diseñar la parte gráfica, se han hecho dos bocetos, primero uno en bruto (Ilustración 4.18) y después uno detallando cada uno de los componentes que formarán parte de la interfaz (Ilustración 4.19).

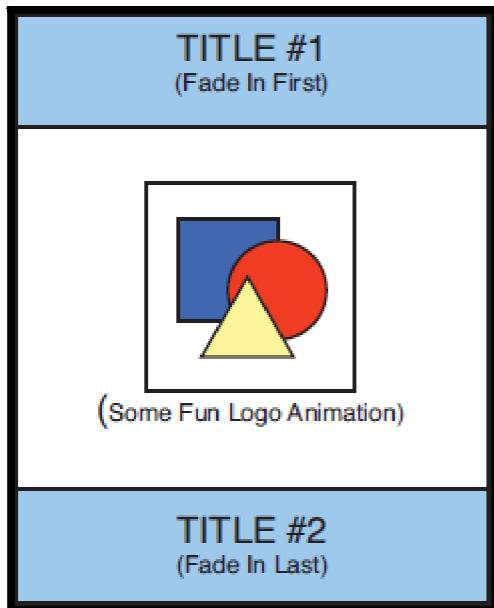


Ilustración 4.18.- Pantalla inicial en bruto.

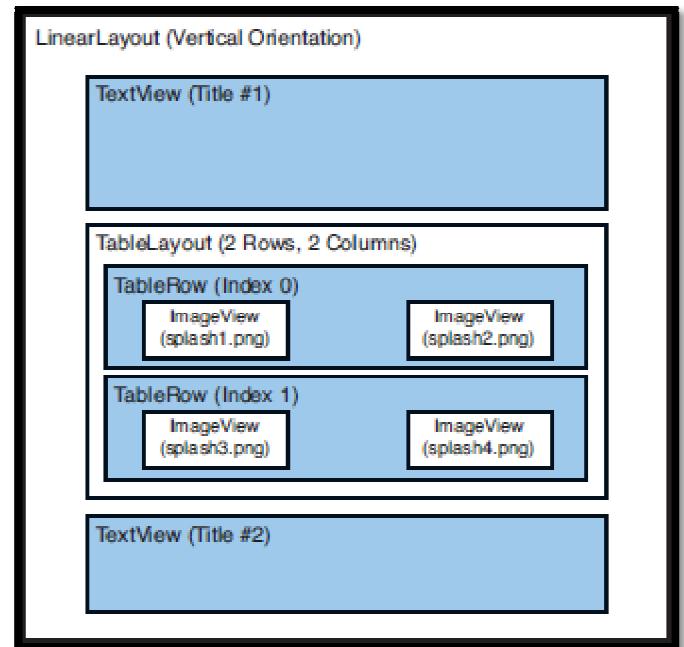


Ilustración 4.19.- Pantalla inicial en detalle.

Cuando el usuario pulsa sobre cualquier parte de la pantalla, se abre el menú de la aplicación. Esto se lleva a cabo gestionando el *OnTouchListener* de la siguiente forma:

```
LinearLayout ll_splash = (LinearLayout)
findViewById(R.id.linearlayout1);

ll_splash.setOnTouchListener(new AdapterView.OnTouchListener() {
    public boolean onTouch(View v, MotionEvent event) {
        if(event.getAction() == MotionEvent.ACTION_DOWN) {
            startActivity(new Intent(SplashActivity.this,
MenuActivity.class));
            SplashActivity.this.finish();
        }
        return true;
    }
});
```

También mencionar, que la pantalla inicial contiene animación, por lo que a continuación se muestra el código empleado para dicho efecto:

```

private void startAnimating() {
    // Fade in top title
    TextView logo1 = (TextView) findViewById(R.id.txt_top_title);
    Animation fadel = AnimationUtils.loadAnimation(this,
R.anim.fade_in);
    logo1.startAnimation(fadel);

    // Load animations for all views within the TableLayout
    Animation spinin = AnimationUtils.loadAnimation(this,
R.anim.custom_anim);
    LayoutAnimationController controller = new
LayoutAnimationController(spinin);
    TableLayout table = (TableLayout)
findViewById(R.id.TableLayout01);
    for (int i = 0; i < table.getChildCount(); i++) {
        TableRow row = (TableRow) table.getChildAt(i);
        row.setLayoutAnimation(controller);
    }
}

```

4.4.5. Menú

Esta pantalla es la que presenta el menú de la aplicación. Al igual que se hizo en el caso anterior, se diseñaron dos bocetos de interfaz, uno en bruto (Ilustración 4.20) y otro en detalle (Ilustración 4.21).

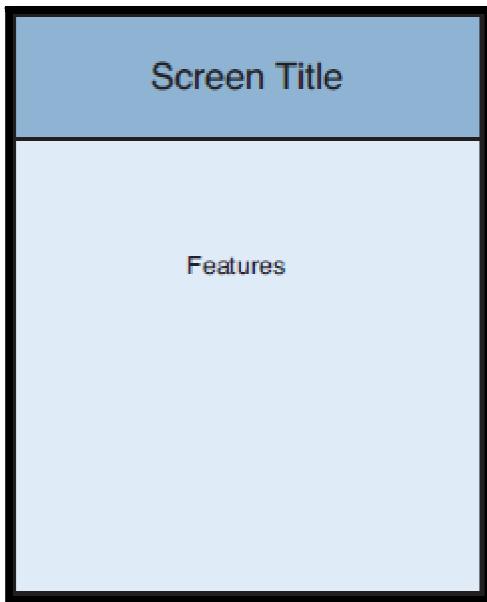


Ilustración 4.20.- Diseño en bruto de la pantalla del menú.

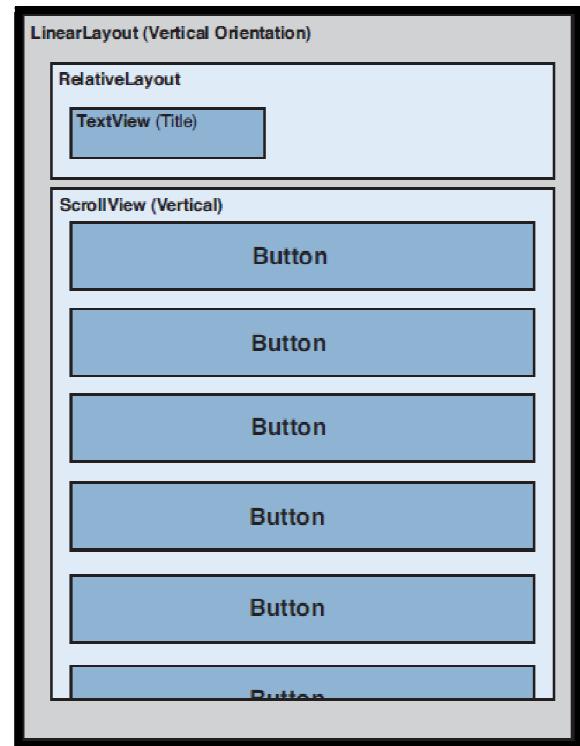


Ilustración 4.21.- Diseño detallado de la pantalla del menú.

Desde los distintos botones, se accede a las diferentes funcionalidades de la aplicación. Para ello, se gestionan los eventos de *OnClickListener*. Para todos los casos menos para el de poner en marcha el acelerómetro, se debe comenzar una *activity*, por lo que se ha creado un método general para evitar la redundancia de código. Dicho código se muestra a continuación:

```
public void manage_click_btn(Button btn, final Class<?> cls){  
    btn.setOnClickListener(new AdapterView.OnClickListener() {  
        @Override  
        public void onClick(View v) {  
            // Launch Activity  
            startActivity(new Intent(MenuActivity.this, cls));  
        }  
    };
```

Para el caso de la puesta en marcha del acelerómetro, al haberse implementado con un *Service*, debe cambiarse la línea de `startActivity`, por esta otra:

```
startService(new Intent(MenuActivity.this, cls));
```

4.4.6. Configuración

Una vez más, se tienen los diseños para la interfaz gráfica, el de en bruto (Ilustración 4.22) y el detallado (Ilustración 4.23).

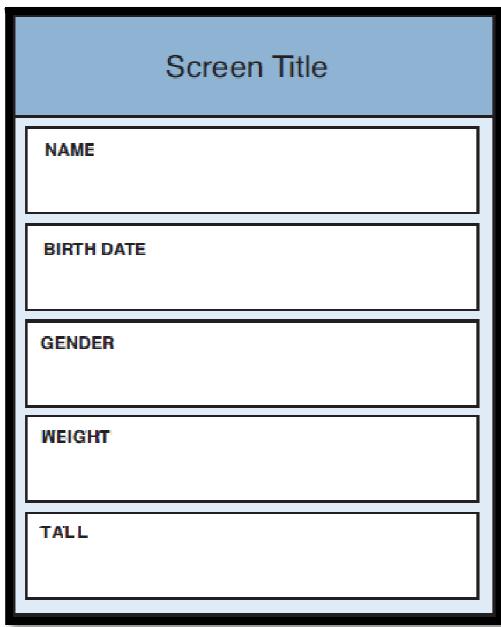


Ilustración 4.22.- Diseño en bruto de la pantalla de configuración.

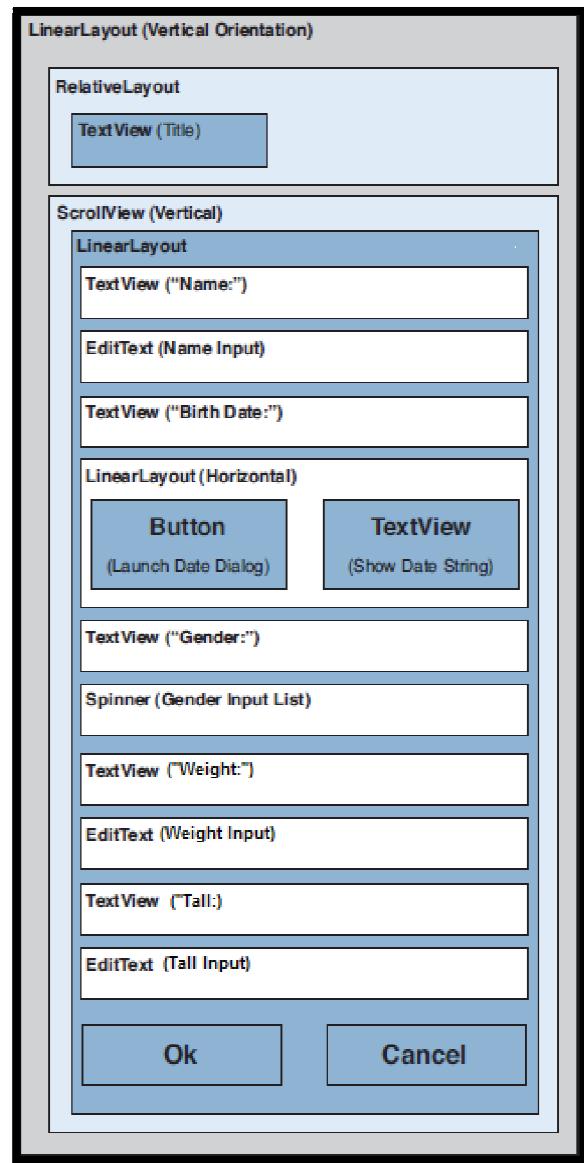


Ilustración 4.23.- Diseño detallado de la pantalla de configuración.

Lo primero que el usuario debe hacer al abrir la aplicación por primera vez, es llenar el formulario de configuración. Una vez llenado, debe pulsar el botón de aceptar. En ese momento, se almacenan los datos de dos formas distintas. Por un lado todos los datos se guardan como `SharedPreferences` y por otro, con el peso y la altura se inserta una fila en la tabla `weight_bmi` de la base de datos. El código empleado para el almacenamiento como `SharedPreferences` es el siguiente:

```
// Se asigna al SharedPreferences APP_PREFERENCES la propiedad de usar
// datos compartidos únicamente por esta aplicación (MODE_PRIVATE)
SharedPreferences settings = getSharedPreferences (APP_PREFERENCES,
        MODE_PRIVATE);

// Se indica que se va a editar su valor
SharedPreferences.Editor editor = settings.edit();
```

```

// Se guardan los valores que se desee
editor.putString(APP_PREFERENCES_NAME, name);
editor.putString(APP_PREFERENCES_DOB, date_of_birth);
editor.putString(APP_PREFERENCES_GENDER, gender);
editor.putFloat(APP_PREFERENCES_WEIGHT, weight);
editor.putFloat(APP_PREFERENCES_TALL, tall);

// Actualizar las SharedPreferences
editor.commit();

```

Mientras que el empleado para guardar los datos en la base de datos es este otro:

```

Calendar cal = Calendar.getInstance();
long now = cal.getTimeInMillis();
long ms = cal.MILLISECOND;
long sec_ms = cal.SECOND * 1000;
long min_ms = cal.MINUTE * 60000;
long h_ms = cal.HOUR * 3600000;
long timestamp = now - h_ms - min_ms - sec_ms - ms;

WeightBMIOBJECT w_bmi_obj = new WeightBMIOBJECT(weight, tall,
timestamp);

mDbHelper.insertWeightBMI(w_bmi_obj);

```

Es importante restar al *timestamp* los valores de la hora, ya que ese dato se va a leer posteriormente por la librería de gráficas *AndroidPlot* que sólo maneja los *timestamp* por días, por lo que podrían mostrarse valores incoherentes aunque estos se hayan recogido correctamente.

4.4.7. Play

Esta funcionalidad no presenta interfaz gráfica ya que se implementa con un *Service*. La razón de implementarlo de esta forma es porque se necesita interacción con el usuario pero sí que se sigan recogiendo datos del acelerómetro aunque se salga de la aplicación.

Cuando el usuario selecciona la opción "play" en el menú, inmediatamente se inicia el servicio definido por la clase *AccelerometerService*. En esta clase se llevan a cabo dos operaciones importantes. Por un lado se inicializa el sensor para que comience a recoger datos y por otro, se inicializan dos alarmas, una para cada minuto y otra para una vez al día. Es importante mencionar que se ha decidido registrar los datos cada minuto ya que la frecuencia de lectura del sensor no es fija, lo que podría dificultar el análisis de los datos.

Para inicializar y registrar el sensor, se ejecuta el siguiente código:

```

// Obtener la referencia al Sensor
SensorManager s_manager = (SensorManager) getSystemService(
Context.SENSOR_SERVICE);

// Se va a usar el acelerómetro
Sensor accelerometer_sensor = s_manager.getDefaultSensor(
Sensor.TYPE_ACCELEROMETER);

// Registrar sensor
s_manager.registerListener(mySensorEventListener, accelerometer_sensor,
SensorManagerSENSOR_DELAY_NORMAL);

```

Cada vez que se genere un dato por el sensor, se recogen esos datos y se tratan para eliminarles la componente de la gravedad como muestra el siguiente código:

```
// Eliminar la influencia de la gravedad en la medición de la
// aceleración.
// alpha is calculated as t / (t + dT) with t, the low-pass filter's
// time-constant and dT, the event delivery rate
final float alpha = (float) 0.8;

gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

linear_acceleration[0] = event.values[0] - gravity[0];
linear_acceleration[1] = event.values[1] - gravity[1];
linear_acceleration[2] = event.values[2] - gravity[2];

// Se crea el objeto con los datos y se guarda en el ArrayList
// correspondiente
MeasureObject mObj = new MeasureObject(linear_acceleration[0],
linear_acceleration[1], linear_acceleration[2]);

measure_list.add(mObj);
```

Por otro lado se tenía la inicialización de las alarmas. Esto se lleva a cabo de la siguiente forma:

```
// Se quiere que se ejecute cada día a las 23:55
// 23:59 is the last minute of the day and 00:00 is the first minute
// of the next day
Calendar cal = Calendar.getInstance();
    cal.set(Calendar.HOUR_OF_DAY, 23);
    cal.set(Calendar.MINUTE, 55);
    cal.set(Calendar.SECOND, 0);

AlarmManager alarmManager = (AlarmManager) getSystemService(
ALARM_SERVICE);

Intent intent_day      = new Intent(this, AlarmReceiverDay.class);
Intent intent_minute = new Intent(this, AlarmReceiverMinute.class);

PendingIntent pending_intent_day      = PendingIntent.getBroadcast(this,
ALARM_REQUEST_CODE_DAY, intent_day, 1);
PendingIntent pending_intent_minute = PendingIntent.getBroadcast(this,
ALARM_REQUEST_CODE_MINUTE, intent_minute, 1);

// Alarma cada 60000 segundos => 1 minuto
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(),
),60000,pending_intent_minute);

// Alarma cada 86400000 segundos => 24h
alarmManager.setRepeating(AlarmManager.RTC_WAKEUP, cal.getTimeInMillis(),
),86400000,pending_intent_day);
```

A continuación se va a explicar qué ocurre cuando se ejecutan cada una de las alarmas anteriormente mencionadas. Antes de ello, mencionar que ambas clases se implementan como *BroadcastReceivers* ya que responden al evento de la alarma.

AlarmReceiverMinute

Cada minuto, se ven los datos recogidos por el acelerómetro y a partir de ellos se calculan las características necesarias para clasificar la actividad realizada en ese minuto. Se calculan la media y la desviación estándar de la siguiente manera:

```
/**  
 * Función que calcula la media de la aceleración absoluta.  
 * @return  
 */  
public float calculate_mean(){  
    float measure_mean = 0;  
  
    // Se devuelven todos los valores del ArrayList  
    Iterator<MeasureObject> iterador =  
AccelerometerService.measure_list.listIterator();  
  
    //Mientras que el iterador tenga un proximo elemento  
    while( iterador.hasNext() ) {  
        MeasureObject mObj = (MeasureObject) iterador.next();  
  
        measure_mean += mObj.function_xyz;  
    }  
    return measure_mean/AccelerometerService.measure_list.size();  
}  
  
/**  
 * Función que calcula la varianza de la aceleración.  
 * @return  
 */  
public float calculate_acc_variance(){  
    float measure_acc_var = 0;  
  
    // Se devuelven todos los valores del ArrayList  
    Iterator<MeasureObject> iterador =  
AccelerometerService.measure_list.listIterator();  
  
    //Mientras que el iterador tenga un proximo elemento  
    while( iterador.hasNext() ) {  
        MeasureObject mObj = (MeasureObject) iterador.next();  
  
        measure_acc_var += Math.pow(mObj.function_xyz-mean, 2);  
    }  
    return measure_acc_var/AccelerometerService.measure_list.size();  
}
```

En función de la desviación estándar, como se ha visto en la sección 4.4.3, se determina el tipo de actividad realizada:

```
if(std_dev < threshold_rest_moderate)  
    return "reposo";  
else  
    if(std_dev > threshold_moderate_intensive)  
        return "intensivo";  
    else return "moderado";
```

Al finalizar cada minuto, se crea un objeto *PreprocessingObject*, se añade al *arraylist* correspondiente y se borra la información del acelerómetro para liberar memoria:

```

PreprocessingObject pObj = new PreprocessingObject(mean, std_dev,
activity_type);
preprocessing_list.add(pObj);

AccelerometerService.measure_list.clear();

```

AlarmReceiverDay

Al final de cada día, se llevan a cabo varias operaciones. En primer lugar, se recogen los datos de cada minuto generados a lo largo del día para determinar el tiempo dedicado a cada una de las actividades:

```

for (int i=0; i<preprocessing_list.size(); i++){
    if (preprocessing_list.get(i).activity_type == REST)
        num_min_rest++;
    if (preprocessing_list.get(i).activity_type == MODERATE)
        num_min_moderate++;
    if (preprocessing_list.get(i).activity_type == INTENSIVE)
        num_min_intensive++;
}

```

Una vez obtenidos estos datos, se inserta una nueva fila en la tabla *daily_activity* de la base de datos con la información de la actividad realizada durante el día:

```

//Se coge la fecha
Calendar cal = Calendar.getInstance();
nYear = cal.get(Calendar.YEAR);
nMonth = cal.get(Calendar.MONTH);
nDay = cal.get(Calendar.DAY_OF_MONTH);

// Los meses empiezan a contar en 0, por eso se suma 1
date_day_activity = nDay + "/" + (nMonth +1) + "/" + nYear;

// Ver si el objetivo se ha cumplido
mission_day_accomplished = is_day_mission_accomplished();

// Coger el id de la ultima fila de la tabla week_activity
Cursor c = mDbHelper.fetchAllWeekActivities();
fk_week = c.getCount();

c.close();

// Se inserta en la bbdd
mDbHelper.insertDailyActivity(num_min_rest,
                               num_min_moderate,
                               num_min_intensive,
                               mission_day_accomplished,
                               fk_week,
                               date_day_activity);

```

Para saber si el objetivo diario se ha cumplido o no, se miran los minutos de actividad realizados y se comparan con el objetivo diario fijado en la tabla semanal de actividad:

```

// La semana en la que estamos es la última de la bbdd.
Cursor c = mDbHelper.fetchAllWeekActivities();
int num_rows = c.getCount();
c.moveToPosition(num_rows-1);
int daily_aim_moderate = (int) c.getLong(4);
int daily_aim_intensive = (int) c.getLong(5);

```

```

c.close();

if ( num_min_moderate  >= daily_aim_moderate || 
    num_min_intensive  >= daily_aim_intensive)
    return 1;
else return 0;

```

El siguiente paso será actualizar la base de datos semanal para llevar el acumulativo de minutos de cada actividad en la semana en curso:

```

// La semana en la que estamos es la última de la bbdd.
Cursor c = mDbHelper.fetchAllWeekActivities();
int num_rows = c.getCount();
c.moveToPosition(num_rows-1);

tt_moderate  =  num_min_moderate  + c.getInt(2);
tt_intensive =  num_min_intensive + c.getInt(3);

mission_week_accomplished = is_mission_week_accomplished();

mDbHelper.updateWeekActivity(num_rows,
                             tt_moderate,
                             tt_intensive,
                             c.getInt(4) ,
                             c.getInt(5) ,
                             c.getInt(6) ,
                             c.getInt(7) ,
                             mission_week_accomplished,
                             c.getString(1));

c.close();

```

Para saber si se ha cumplido el objetivo semanal, se contrasta el acumulativo semanal con el objetivo final de la semana:

```

// La semana en la que estamos es la última de la bbdd.
Cursor c = mDbHelper.fetchAllWeekActivities();
int num_rows = c.getCount();
c.moveToPosition(num_rows-1);
int week_aim_moderate = (int) c.getLong(6);
int week_aim_intensive = (int) c.getLong(7);
c.close();

if (tt_moderate  >= week_aim_moderate ||
    tt_intensive >= week_aim_intensive)
    return 1;
else return 0;

```

Por último, y en función de los valores obtenidos anteriormente que determinan si se han cumplido o no los objetivos, se envían notificaciones con unos mensajes u otros al usuario.

En primer lugar se mira si es domingo (día elegido como fin de semana). Si no es domingo y el acumulativo de minutos de actividad supera el 50% del objetivo, se envía una notificación con el siguiente mensaje:

¡BUEN TRABAJO! ¡Falta menos para conseguir el reto, sigue así!
--

Por el contrario, si es domingo se mira si se ha cumplido el objetivo semanal. Si es así se envía la siguiente notificación:

```
¡ENHORABUENA! ¡Has superado el objetivo de esta semana!
```

Si no se cumple el objetivo, se envía este otro mensaje:

```
Esta vez no pudo ser...;No te desanimes! Invierte más tiempo esta semana y lo conseguirás ;)
```

Estos mensajes se envían a través de notificaciones. El código de creación de una notificación es el siguiente:

```
// Get a reference to the NotificationManager:  
String ns = Context.NOTIFICATION_SERVICE;  
NotificationManager mNotificationManager = (NotificationManager)  
context.getSystemService(ns);  
  
// Instantiate the Notification:  
int icon = R.drawable.icon;  
CharSequence tickerText = "Notificación";  
long when = System.currentTimeMillis();  
  
Notification notification = new Notification(icon, tickerText, when);  
  
notification.defaults |= Notification.DEFAULT_SOUND;  
  
notification.flags |= Notification.FLAG_AUTO_CANCEL;  
  
// Define the Notification's expanded message and Intent:  
  
CharSequence contentTitle = msg1_title;  
CharSequence contentText = msg1_txt;  
Intent notificationIntent = new Intent();  
PendingIntent contentIntent = PendingIntent.getActivity(context, 0,  
notificationIntent, 0);  
  
notification.setLatestEventInfo(context, contentTitle, contentText,  
contentIntent);  
  
// Pass the Notification to the NotificationManager:  
mNotificationManager.notify(HELLO_ID, notification);
```

Además de esto, al llegar el domingo significa que se ha terminado la semana y comienza una nueva, por lo que se crea una nueva entrada en la tabla de actividad semanal de la base de datos. Para ello, lo primero será calcular el nuevo objetivo en función del último IMC del que se tenga constancia:

```
// Nos interesa el ultimo peso de la bbdd  
Cursor c = mDbHelper.fetchAllWeightBMI();  
int num_rows = c.getCount();  
c.moveToPosition(num_rows-1);  
  
float new_bmi = c.getFloat(2);  
  
c.close();  
  
return new_bmi;
```

Conociendo el último IMC, se calcula el nuevo objetivo y se crea la entrada a la tabla de la base de datos anteriormente mencionada:

```
// Según el IMC se fijan unos objetivos
// 18.5=< IMC < 25 => NORMAL
// 25  =< IMC < 30 => PREOBESO
// 30  =< IMC      => OBESO
String next_day = (nDay+1) + " / " + (nMonth +1) + " / " + nYear;
if(new_bmi < 18.5)
    mDbHelper.insertWeekActivity(0, 0,
                                aim_day_moderate_prenormal,
                                aim_day_intensive_prenormal,
                                aim_week_moderate_prenormal,
                                aim_week_intensive_prenormal,
                                0, next_day);
if(new_bmi >= 18.5 && new_bmi < 25)
    mDbHelper.insertWeekActivity(0, 0,
                                aim_day_moderate_normal,
                                aim_day_intensive_normal,
                                aim_week_moderate_normal,
                                aim_week_intensive_normal,
                                0, next_day);
if(new_bmi >= 25 && new_bmi < 30)
    mDbHelper.insertWeekActivity(0, 0,
                                aim_day_moderate_preobesity,
                                aim_day_intensive_preobesity,
                                aim_week_moderate_preobesity,
                                aim_week_intensive_preobesity,
                                0, next_day);
if(new_bmi >= 30)
    mDbHelper.insertWeekActivity(0, 0,
                                aim_day_moderate_obesity,
                                aim_day_intensive_obesity,
                                aim_week_moderate_obesity,
                                aim_week_intensive_obesity,
                                0, next_day);
```

Los objetivos semanales y diarios se han definido en minutos de la siguiente forma:

```
int aim_week_moderate_obesity      = 210;
int aim_week_intensive_obesity     = 0;
int aim_day_moderate_obesity       = 30;
int aim_day_intensive_obesity     = 0;

int aim_week_moderate_preobesity   = 210;
int aim_week_intensive_preobesity  = 70;
int aim_day_moderate_preobesity    = 30;
int aim_day_intensive_preobesity   = 10;

int aim_week_moderate_normal       = 210;
int aim_week_intensive_normal      = 70;
int aim_day_moderate_normal        = 30;
int aim_day_intensive_normal       = 10;

int aim_week_moderate_prenormal    = 105;
int aim_week_intensive_prenormal   = 35;
int aim_day_moderate_prenormal     = 15;
int aim_day_intensive_prenormal    = 5;
```

4.4.8. Evolución

Una vez más, como el resto de pantallas con interfaz gráfica, se realizaron dos bocetos, uno en bruto (Ilustración 4.24) y una en detalle (Ilustración 4.25).

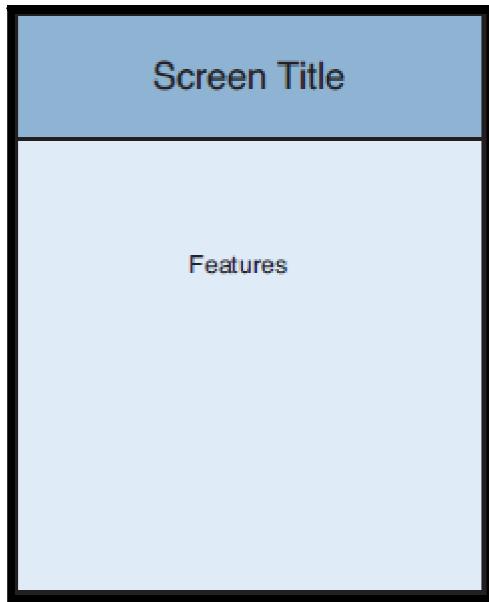


Ilustración 4.24.- Pantalla en bruto de la interfaz general de evolución.

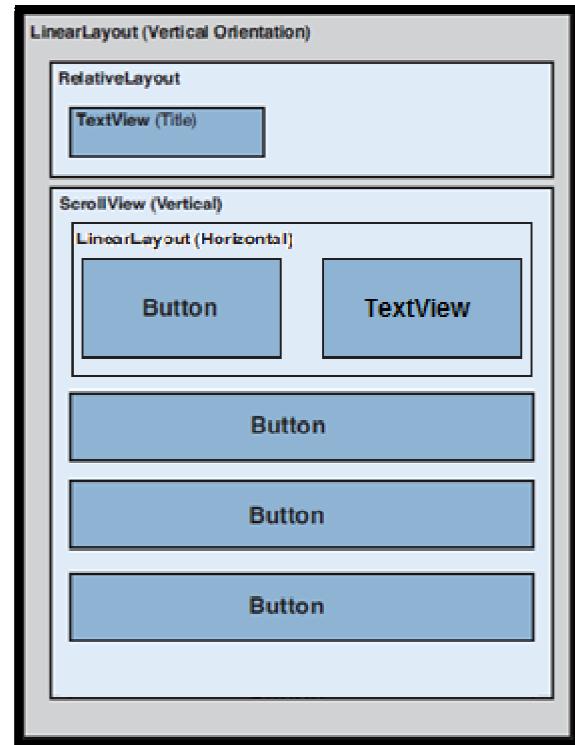


Ilustración 4.25.- Pantalla en detalle de la interfaz general de evolución.

Navegando por los distintos botones, se accede a las distintas funcionalidades dentro de esta pantalla. Dichas funcionalidades se comentan a continuación:

Modificar peso

Si se selecciona esta opción se abre un diálogo que permite introducir el peso del usuario. Al pulsar sobre aceptar, se recogen los datos y se inserta el nuevo registro en la base de datos como se muestra a continuación:

```
LayoutInflater inflater = (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE);
final View layout = inflater.inflate(R.layout.weight_dialog,
(ViewGroup) findViewById(R.id.root));
final EditText ewd = (EditText)
layout.findViewById(R.id.edit_weight_dialog);

return new AlertDialog.Builder(MonitoringActivity.this)
.setView(layout)
.setTitle(R.string.new_weight)
.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
```

```

public void onClick(DialogInterface dialog, int whichButton) {

    // Coger datos y guardarlos en la bbdd.
    Calendar cal = Calendar.getInstance();
    long now = cal.getTimeInMillis();
    long ms = cal.MILLISECOND;
    long sec_ms = cal.SECOND * 1000;
    long min_ms = cal.MINUTE * 60000;
    long h_ms = cal.HOUR * 3600000;
    long timestamp = now - h_ms - min_ms - sec_ms - ms;

    weight = Float.parseFloat(ewd.getText().toString());

    WeightBMIOBJECT w_bmi_obj = new WeightBMIOBJECT(weight,
    sp_tall, timestamp);

    mDbHelper.insertWeightBMIS(w_bmi_obj);

    // We forcefully dismiss and remove the Dialog, so it
    // cannot be used again
    MonitoringActivity.this.removeDialog(WEIGHT_DIALOG_ID);
}

.setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int whichButton) {
        MonitoringActivity.this.removeDialog(WEIGHT_DIALOG_ID);
    }
})
.create();

```

Evolución del peso

La evolución del peso se muestra en una gráfica diseñada gracias a la librería AndroidPlot. Los datos a dibujar se recogen de la base de datos de la siguiente forma:

```

ArrayList<WeightBMIOBJECT> weight_bmi_list = new
ArrayList<WeightBMIOBJECT>();

Cursor c = mDbHelper.fetchAllWeightBMIS();

// Iteramos a través de los registros del cursor
if (c != null) {
    c.moveToFirst();
    while (c.isAfterLast() == false) {
        WeightBMIOBJECT weight_bmi = new WeightBMIOBJECT();
        weight_bmi.set_weight(c.getFloat(1));
        weight_bmi.set_bmi(c.getFloat(2));
        weight_bmi.set_timestamp(c.getLong(3));
        weight_bmi_list.add(weight_bmi);
        c.moveToNext();
    }
    c.close();
} else;

int num_rows = weight_bmi_list.size();
Number [] weight_array = new Number [num_rows];
Number [] timestamps = new Number [num_rows];

for (int i=0 ; i<num_rows; i++){
    weight_array[i] = (Number)weight_bmi_list.get(i).get_weight();
}

```

```
    timestamps[i] = (Number)weight_bmi_list.get(i).get_timestamp();
}
```

Las características de la gráfica se definen con el siguiente código:

```
XYSeries series2 = new SimpleXYSeries(
    Arrays.asList(timestamps),
    Arrays.asList(weight_array),
    "Peso según Fecha");

mySimpleXYPlot.getGraphWidget().getGridBackgroundPaint().setColor(Color.WHITE);

mySimpleXYPlot.getGraphWidget().getGridLinePaint().setColor(Color.BLACK);

mySimpleXYPlot.getGraphWidget().getGridLinePaint().setPathEffect(new
DashPathEffect(new float[]{1,1}, 1));

mySimpleXYPlot.getGraphWidget().getDomainOriginLinePaint().setColor(Color.BLACK);

mySimpleXYPlot.getGraphWidget().getRangeOriginLinePaint().setColor(Color.BLACK);

mySimpleXYPlot.setBorderStyle(Plot.BorderStyle.SQUARE, null, null);
mySimpleXYPlot.getBorderPaint().setStrokeWidth(1);
    mySimpleXYPlot.getBorderPaint().setAntiAlias(false);
mySimpleXYPlot.getBorderPaint().setColor(Color.WHITE);

// setup our line fill paint to be a slightly transparent gradient:
Paint lineFill = new Paint();
lineFill.setAlpha(200);
lineFill.setShader(new LinearGradient(0, 0, 0, 250, Color.WHITE,
Color.GREEN, Shader.TileMode.MIRROR));

LineAndPointFormatter formatter = new
LineAndPointFormatter(Color.rgb(0, 0, 0), Color.BLUE, Color.RED);
formatter.setFillPaint(lineFill);
mySimpleXYPlot.getGraphWidget().setPaddingRight(2);
mySimpleXYPlot.addSeries(series2, formatter);

// draw a domain tick for each year:
mySimpleXYPlot.setDomainStep(XYStepMode.SUBDIVIDE, timestamps.length);

// customize our domain/range labels
mySimpleXYPlot.setDomainLabel("Fecha");
mySimpleXYPlot.setRangeLabel("Peso en Kg");

// get rid of decimal points in our range labels:
mySimpleXYPlot.setRangeValueFormat(new DecimalFormat("###.##"));

mySimpleXYPlot.setDomainValueFormat(new MyDateFormat());

// by default, AndroidPlot displays developer guides to aid in laying
out your plot.
// To get rid of them call disableAllMarkup():
mySimpleXYPlot.disableAllMarkup();
```

Evolución del IMC

Para la evolución del IMC, se siguen los mismos pasos que para la evolución del peso, pero cambiando el valor extraído de la base de datos:

```
ArrayList<WeightBMIOBJECT> weight_bmi_list = new  
ArrayList<WeightBMIOBJECT>();  
  
Cursor c = mDbHelper.fetchAllWeightBMI();  
  
//Iteramos a traves de los registros del cursor  
if (c != null) {  
    c.moveToFirst();  
    while (c.isAfterLast() == false) {  
        WeightBMIOBJECT weight_bmi = new WeightBMIOBJECT();  
        weight_bmi.set_weight(c.getFloat(1));  
        weight_bmi.set_bmi(c.getFloat(2));  
        weight_bmi.set_timestamp(c.getLong(3));  
        weight_bmi_list.add(weight_bmi);  
        c.moveToNext();  
    }  
    c.close();  
} else;  
  
int num_rows = weight_bmi_list.size();  
  
Number [] bmi_array = new Number [num_rows];  
Number [] timestamps = new Number [num_rows];  
  
for (int i=0 ; i<num_rows; i++){  
    bmi_array[i] = (Number)weight_bmi_list.get(i).get_bmi();  
    timestamps[i] = (Number)weight_bmi_list.get(i).get_timestamp();  
}
```

Este gráfico es más complicado de representar que el del peso ya que se divide en distintas franjas de acuerdo al IMC, para que el usuario sea consciente de su estado (poco peso, normal, preobeso u obeso). El código que permite esto es el siguiente:

```
Paint line_fill_obesity = new Paint();  
line_fill_obesity.setColor(Color.RED);  
  
lp_formatter_obesity = new LineAndPointFormatter(  
    Color.RED,  
    Color.RED,  
    Color.RED);  
lp_formatter_obesity.setFillPaint(line_fill_obesity);  
lp_formatter_obesity.setVertexPaint(null);  
lp_formatter_obesity.getLinePaint().setShadowLayer(0, 0, 0, 0);  
  
mySimpleXYPlot.addSeries(new SimpleXYSeries(Arrays.asList(timestamps),  
    Arrays.asList(obesity),  
    "Obeso"), lp_formatter_obesity);  
  
// PREOBESITY SERIE  
Paint line_fill_preobesity = new Paint();  
line_fill_preobesity.setColor(Color.YELLOW);  
  
lp_formatter_preobesity = new LineAndPointFormatter(  
    Color.YELLOW,  
    Color.YELLOW,
```

```

        Color.YELLOW);
lp_formatter_preobesity.setFillPaint(line_fill_preobesity);

//Quitar los circulos de los puntos
lp_formatter_preobesity.setVertexPaint(null);
lp_formatter_preobesity.getLinePaint().setShadowLayer(0, 0, 0, 0);

mySimpleXYPlot.addSeries(new SimpleXYSeries(Arrays.asList(timestamps),
    Arrays.asList(preobesity),
    "Pre-Obeso"), lp_formatter_preobesity);

// NORMAL SERIE
Paint line_fill_normal = new Paint();
line_fill_normal.setColor(Color.GREEN);

lp_formatter_normal = new LineAndPointFormatter(
    Color.GREEN,
    Color.GREEN,
    Color.GREEN);
lp_formatter_normal.setFillPaint(line_fill_normal);
lp_formatter_normal.setVertexPaint(null);
lp_formatter_normal.getLinePaint().setShadowLayer(0, 0, 0, 0);

mySimpleXYPlot.addSeries(new SimpleXYSeries(Arrays.asList(timestamps),
    Arrays.asList(normal),
    "Normal"), lp_formatter_normal);

// UNDERWEIGHT SERIE
Paint line_fill_underweight = new Paint();
line_fill_underweight.setColor(Color.GRAY);

lp_formatter_underweight = new LineAndPointFormatter(
    Color.GRAY,
    Color.GRAY,
    Color.GRAY);
lp_formatter_underweight.setFillPaint(line_fill_underweight);
lp_formatter_underweight.setVertexPaint(null);
lp_formatter_underweight.getLinePaint().setShadowLayer(0, 0, 0, 0);

mySimpleXYPlot.addSeries(new SimpleXYSeries(Arrays.asList(timestamps),
    Arrays.asList(underweight),
    "Poco Peso"), lp_formatter_underweight);

// BMI SERIE:
lp_formatter_bmi = new LineAndPointFormatter(
    Color.BLACK,
    Color.BLACK,
    Color.BLACK);
lp_formatter_bmi.setFillPaint(null);

lp_formatter_bmi.getLinePaint().setShadowLayer(0, 0, 0, 0);

mySimpleXYPlot.addSeries(new SimpleXYSeries(Arrays.asList(timestamps),
    Arrays.asList(bmi_array),
    "IMC"), lp_formatter_bmi);

// draw a domain tick for each year:
mySimpleXYPlot.setDomainStep(XYStepMode.SUBDIVIDE, timestamps.length);

// customize our domain/range labels

```

```

mySimpleXYPlot.setDomainLabel("Fecha");
mySimpleXYPlot.setRangeLabel("IMC");

// get rid of decimal points in our range labels:
mySimpleXYPlot.setRangeValueFormat(new DecimalFormat("##.##"));

mySimpleXYPlot.setDomainValueFormat(new MyDateFormat());

// by default, AndroidPlot displays developer guides to aid in laying
// out your plot.
// To get rid of them call disableAllMarkup():
mySimpleXYPlot.disableAllMarkup();

```

Evolución de la actividad

En este caso no se muestra una tabla para informar al usuario, sino que se emplea una tabla que indica la fecha, el objetivo que se debería haber cumplido, y si se ha cumplido o no. Al igual que en los casos anteriores, la información se recoge de la base de datos. Se puede ver el código a continuación:

```

// Sólo se van a mostrar los resultados de la semana en curso.
int id_last_week = 0;

Cursor c_week = mDbHelper.fetchAllWeekActivities();
if (c_week != null) {
    id_last_week = c_week.getCount();
    c_week.moveToPosition(id_last_week-1);

    Cursor c_day = mDbHelper.fetchAllDailyActivities();

    // Iteramos a traves de los registros del cursor
    if (c_day != null) {
        c_day.moveToFirst();
        while (c_day.getLong(6) == id_last_week) {
            String activity_date = c_day.getString(1);
            String activity_aim = c_week.getInt(4) + "min mod o "
                + c_week.getInt(5) + "min int";
            int day_mission_accomplished = c_day.getInt(5);
            String activity_success;
            if(day_mission_accomplished == 1)
                activity_success = "SI";
            else activity_success = "NO";
            insertScoreRow(activities_table, activity_date,
activity_aim, activity_success);
            c_day.moveToNext();
        }
        c_day.close();
    }
    String activity_date = "SEMANA";
    String activity_aim = c_week.getInt(6) + "min mod o "
        + c_week.getInt(7) + "min int";
    int week_mission_accomplished = c_week.getInt(8);
    String activity_success;
    if(week_mission_accomplished == 1)
        activity_success= "SI";
    else activity_success = "NO";
    insertScoreRow(activities_table, activity_date, activity_aim,
activity_success);
} else {
    final TableRow newRow = new TableRow(this);

```

```

        TextView noResults = new TextView(this);
        noResults.setText(getResources().getString(R.string.no_activities))
    );
    newRow.addView(noResults);
    activities_table.addView(newRow);
}

```

4.4.9. Consejos

La pantalla de consejos debe mostrar una gran cantidad de texto, por lo que tiene que tener la capacidad de *Scroll*. Por tanto, el diseño en bruto de esta pantalla sería el mostrado en la Ilustración 4.26.

En la Ilustración 4.27, se ve en mejor detalle cómo quedaría el archivo XML del *layout* de esta pantalla

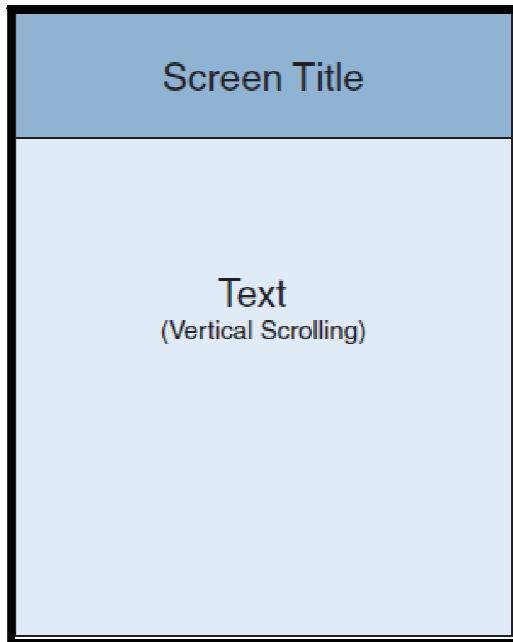


Ilustración 4.26.- Pantalla en bruto de consejos.

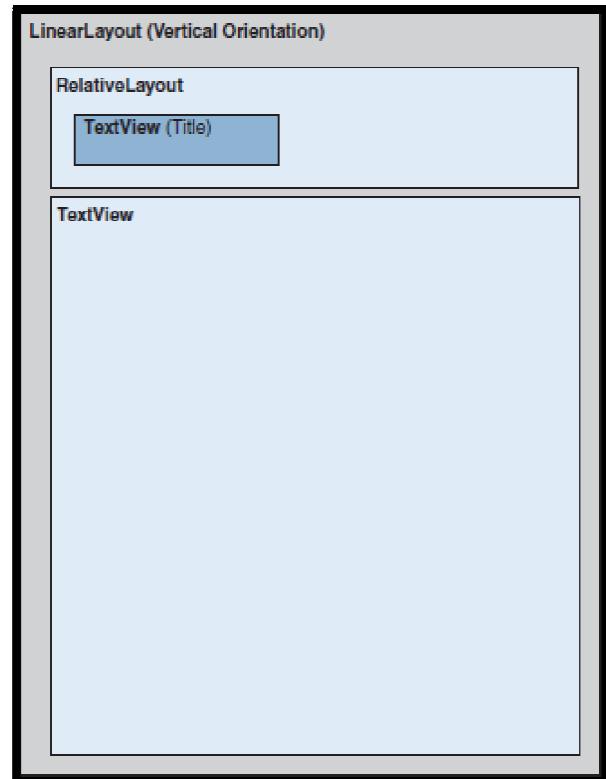


Ilustración 4.27.- Pantalla en detalle de consejos.

El código para leer el fichero con el texto y para mostrarlo en la pantalla, es el siguiente:

```

// Read raw file into string and populate TextView
InputStream iFile = getResources().openRawResource(R.raw.advices);
try {
    TextView advicesText = (TextView)
    findViewById(R.id.txt_advices_text);
    String strFile = inputStreamToString(iFile);
}

```

```
    advicesText.setText(strFile);
} catch (Exception e) {
    Log.e(DEBUG_TAG, "InputStreamToString failure", e);
}
```

4.4.10. Ayuda

La pantalla de ayuda sigue la misma estructura que la de consejos, por lo que no se ve necesaria su explicación.

5. Manual de usuario

En este capítulo se muestra el manual de usuario para que éste pueda hacer un buen uso de la aplicación.

Al abrir la aplicación, aparece la siguiente pantalla de bienvenida:



Ilustración 5.1.- Pantalla inicial de la aplicación.

Pulsando sobre cualquier punto de esta pantalla se accede al menú principal de la aplicación (ver Ilustración 5.2).

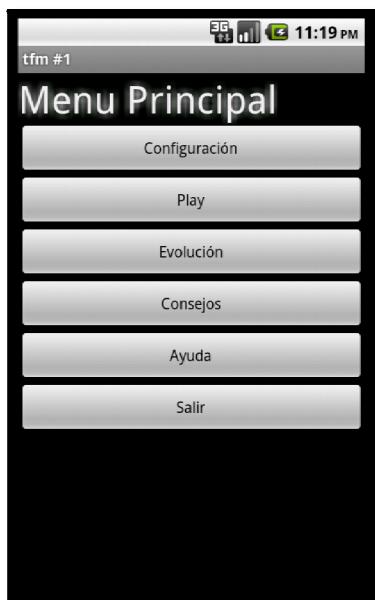


Ilustración 5.2.- Pantalla del menú de la aplicación.

Es importante saber, que el primer paso a realizar si es la primera vez que se abre la aplicación, es el de configuración. Para ello, se selecciona la opción correspondiente en el menú. Inmediatamente, se abre el formulario de configuración (ver Ilustración 5.3).

En esta pantalla, se introducen los siguientes datos: el nombre, la fecha de nacimiento (ver Ilustración 5.4), el género (ver Ilustración 5.5), el peso y la estatura (ver Ilustración 5.6).



Ilustración 5.3.- Pantalla inicial de configuración.



Ilustración 5.4.- Selección de la fecha de nacimiento.



Ilustración 5.5.- Selección del género.

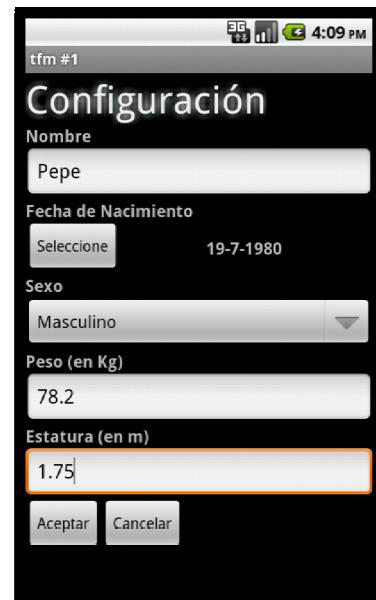


Ilustración 5.6.- Pantalla con todos los datos rellenos.

Una vez introducidos todos los datos, se pulsa el botón de aceptar y estos se almacenan. Se muestra un mensaje al usuario para informar de que los datos han sido guardados (ver Ilustración 5.7).

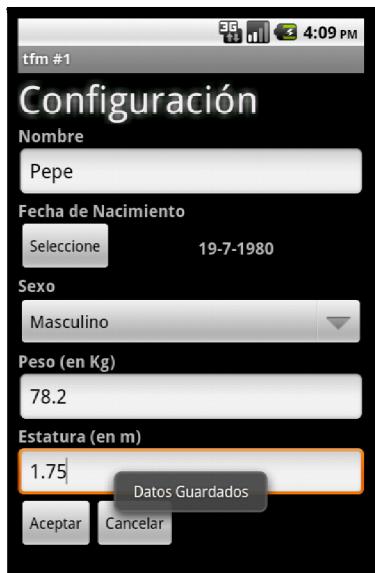


Ilustración 5.7.- Mensaje de datos guardados.

Si se selecciona en el menú principal la opción de evolución, se accede a la siguiente pantalla:

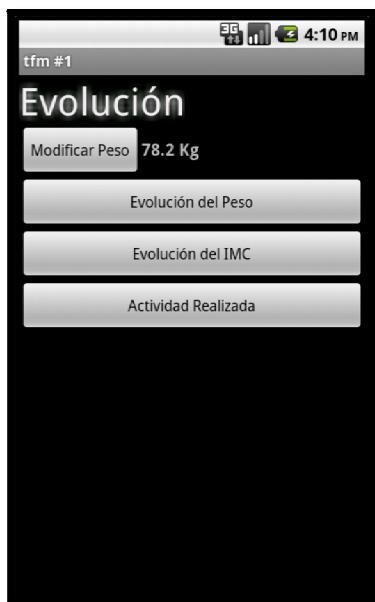


Ilustración 5.8.- Pantalla de la interfaz general de evolución.

En ella puede verse el último peso introducido. Si éste desea cambiarse, se debe seleccionar la opción de modificar peso. En este caso, aparece un cuadro de diálogo para que el usuario pueda introducir el nuevo peso (ver Ilustración 5.9).

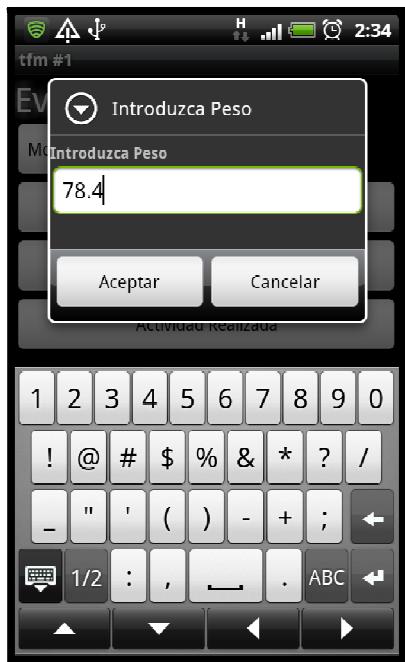


Ilustración 5.9.- Diálogo para la modificación del peso.

Si la opción elegida es la de la evolución del peso, se muestra una gráfica como la siguiente con los últimos datos introducidos:



Ilustración 5.10.- Pantalla de la evolución del peso.

A parte de por el peso, también puede verse la evolución del IMC seleccionando la opción "Evolución del IMC" dentro del menú de evolución. La gráfica que se muestra es la siguiente:



Ilustración 5.11.- Pantalla de la evolución del IMC.

En esta gráfica se muestran mediante franjas diferenciadas con distintos colores, el estado de salud al que corresponde cada uno de los IMC. El usuario tiene así una mayor facilidad para seguir y comprender su evolución.

También puede consultarse la evolución de la actividad física de la última semana. Los datos se recogen de la forma que se ve en la Ilustración 5.12. Puede verse que por cada día se indica si el objetivo se ha cumplido o no. Al final de la semana se hace un resumen para ver si se ha alcanzado el objetivo semanal.

Fecha	Objetivo	Cumplido?
07/07/2011	0,5h	SI
08/07/2011	0,5h	NO
09/07/2011	0,5h	NO
10/07/2011	0,5h	SI
11/07/2011	0,5h	SI
12/07/2011	0,5h	SI
13/07/2011	0,5h	NO
SEMANA	3,5h	SI

Ilustración 5.12.- Pantalla real de la evolución de la actividad.

Si se selecciona la opción de consejos en el menú principal, se muestra la siguiente pantalla con consejos que pueden ser útiles al usuario para conseguir un estilo de vida saludable.

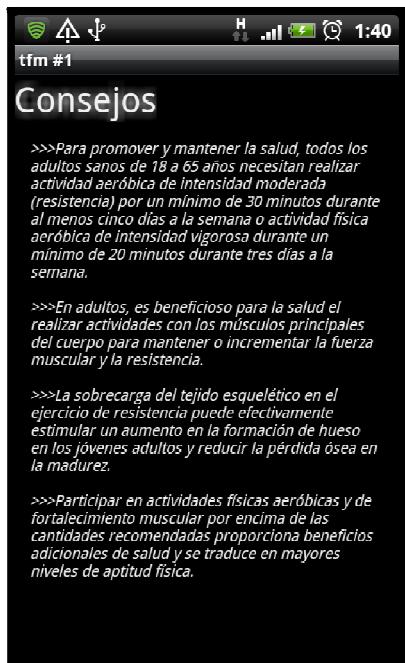


Ilustración 5.13.- Pantalla real de consejos.

Si la opción seleccionada es la ayuda, se muestra un breve resumen del significado de cada una de las funcionalidades de la aplicación:

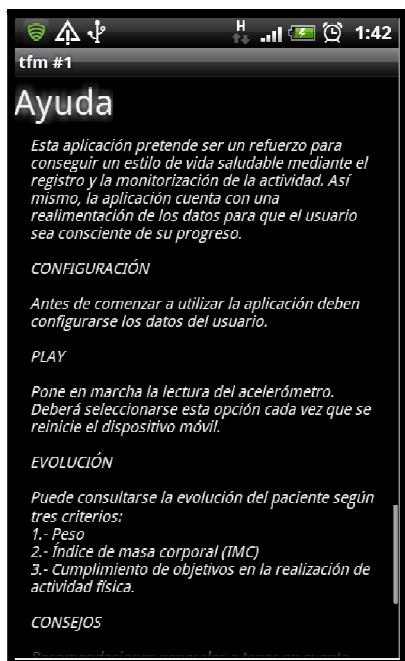


Ilustración 5.14.- Pantalla real de ayuda.

De las opciones del menú inicial, la más importante es la de "play", ya que es la que pone en marcha la lectura del acelerómetro y desencadena el resto de funciones no

mencionadas anteriormente que son las encargadas de interpretar y almacenar los datos. Según los datos recogidos por el acelerómetro, se analiza el tipo de actividad que el usuario ha realizado y si éste ha cumplido el objetivo semanal. La aplicación avisará al usuario de sus logros o fracasos mediante notificaciones en la barra superior del dispositivo (ver Ilustración 5.15).

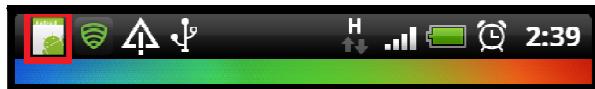


Ilustración 5.15.- Notificación en la barra del dispositivo móvil.

Cuando el usuario despliegue la barra, podrá ver con claridad el mensaje notificado, como se muestra a continuación:



Ilustración 5.16.- Mensaje como notificación.

6. Conclusiones y Propuestas de Mejora

En este capítulo se explican las conclusiones obtenidas al finalizar el presente trabajo fin de máster, así como las posibles mejoras futuras que pueden llevarse a cabo en la aplicación desarrollada.

6.1. Conclusiones

Como se ha comentado en el capítulo anterior, se han cubierto y satisfecho los objetivos planteados al inicio del proyecto, pero llegar a este punto no ha sido sencillo.

Para comenzar se hizo un estudio de mercado de los sistemas operativos móviles para ver con cuál se podría tener una mejor salida para la aplicación desarrollada. Con este estudio se determinó que la mejor plataforma era Android, tanto por las perspectivas de evolución del sistema como por las facilidades que presenta para el desarrollador.

Una vez que se eligió el sistema operativo, tocaba abordar el segundo problema. Hasta el momento de iniciar el proyecto, no se tenía ningún conocimiento sobre programación para dispositivos móviles con Android, por lo que fue necesario familiarizarse con ello.

Uno de los objetivos del proyecto era la realimentación al usuario de su evolución mediante gráficas, por lo que también se hizo un estudio de las posibles librerías que ayudasen a cumplir este objetivo, decantándose finalmente por AndroidPlot. Al igual que ocurría con el sistema operativo Android, esta librería se desconocía, por lo que también tuvo que estudiarse cómo utilizarla.

Otro de los objetivos clave para la realización de la aplicación, era leer los datos del sensor acelerómetro del dispositivo móvil. Aquí se presentó un nuevo problema debido a que la frecuencia de lectura de datos no era fija, sino que podía, y de hecho sufría modificaciones a lo largo del tiempo. Esto se solucionó analizando los datos de cada minuto independientemente del número de muestras que contuviera.

La lectura de los datos del acelerómetro, nos llevaba al siguiente objetivo, el de preprocesar los datos para interpretar la señal. Nuevamente se tenía un ámbito desconocido, por lo que se realizó un estudio sobre las diferentes técnicas que podían emplearse en el preprocesamiento de una señal de acelerometría. Pero además de ello, debía tenerse en cuenta si esas técnicas podrían ser empleadas o no en un dispositivo móvil, ya que éstos cuentan con unos recursos limitados.

Una vez se determinaron las características más viables para interpretar la señal, se hizo necesario estudiar los datos obtenidos para poder establecer los umbrales que determinasen qué tipo de actividad realizaba el usuario en cada momento. Para el análisis de los datos se empleó el software de estadística SPSS que como ocurría en casos anteriores, tuvo que aprenderse a utilizar para realizar este trabajo.

Una vez determinados los umbrales, ya se podía hacer una clasificación de las actividades de acuerdo a sus características, por lo que pudo terminarse de implementar la aplicación.

Aunque en un principio no se pensó en ello, finalmente se decidió emplear la base de datos SQLite incluida en Android para almacenar los datos diarios y semanales de la actividad, así como la evolución del peso y del IMC, de forma que si la aplicación se reiniciase por cualquier motivo, esos datos no se perdiessen.

Finalmente, analizando los datos de la base de datos, se determina si el usuario ha cumplido o no con el objetivo fijado y se le envían notificaciones para que sea consciente de sus progresos.

Para terminar, decir que se ha conseguido implementar una aplicación de registro, procesamiento, monitorización y realimentación de los datos de acelerometría recogidos por un dispositivo móvil con plataforma Android capaz de ayudar al usuario a conseguir un estilo de vida saludable.

6.2. Propuestas de Mejora

Aunque la aplicación cumple con los objetivos propuestos, existen algunas mejoras que podrían llevarse a cabo, tanto para mejorar su eficiencia como para ampliar su ámbito de uso. Dichas mejoras son las siguientes:

- El estudio de los datos para la clasificación se realizó con un único individuo durante 30 minutos de cada actividad. Se propone, para una posible mejor clasificación, realizar un estudio con más gente y durante un periodo de tiempo mayor.
- Además, podría realizarse dicho estudio de clasificación teniendo en cuenta alguna otra variable además de la media y la desviación estándar, como podría ser la transformada rápida de Fourier (FFT) o la entropía de la señal. Aunque como se vio en el apartado 2.3, esta última variable no es aconsejable para implementar en un dispositivo móvil debido a su consumo de recursos.
- Puede realizarse también un estudio de usabilidad para mejorar la interfaz gráfica y el usuario pueda así tener una mejor experiencia con la aplicación.
- Para una futura versión podría pensarse en almacenar los datos recogidos en un servidor para liberar así la memoria del teléfono.
- Podría pensarse también en añadir otro tipo de funcionalidades empleando todos los recursos que ofrece Android como los mapas para trazar rutas o el GPS para localizar a posibles amigos que se encuentren realizando actividad en ese mismo momento.
- También podría pensarse en ampliar el rango de actividades añadiendo por ejemplo: montando en bicicleta, subiendo escaleras, etc.
- Se podría también llevar un cálculo de las calorías quemadas en función de la actividad realizada.

7. Bibliografía

1. **Días, Cinco.** www.cincodias.com. [En línea]
http://www.cincodias.com/articulo/empresas/telefonia-movil-supera-espana-numero-habitantes/20060605cdscdiemp_5/.
2. *Preprocessing techniques for context recognition from accelerometer data.* **Figo, Davide, y otros.** 2010, Pers Ubiquit Comput, Vol. 14, págs. 645-662.
3. **Serrano Rubio, Álvaro José.** Plataforma PREDIRCAM: Diseño e implementación de un sistema remoto para el registro, transferencia, y monitorización de la actividad física. *Proyecto de fin de carrera.* s.l. : UPM, 2010.
4. **Organization, World Health.** <http://www.euro.who.int>. [En línea]
<http://www.euro.who.int/en/what-we-do/health-topics/disease-prevention/nutrition/a-healthy-lifestyle/body-mass-index-bmi>.
5. *Fitness Tour: A Mobile Application for Combating Obesity.* **Chuah, M. y Sample, Steve.** 2011.
6. **Nike.** <http://nikerunning.nike.com>. [En línea]
http://nikerunning.nike.com/nikeos/p/nikeplus/en_US/.
7. **Adidas.** www.adidas.com. [En línea] <http://www.adidas.com/us/micoach>.
8. **UOC.** Universidad Oberta de Cataluya. [En línea]
http://www.uoc.edu/in3/emath/docs/Distrib_Normal.pdf.
9. **Devore, Jay L.** *Probabilidad y estadística para ingeniería y ciencias.* s.l. : Thomson, 2005.
10. *An analysis of variance test for normality (complete samples).* **Shapiro, S. S. y Wilk, M. B.** 1965, Biometrika, págs. 591–611.
11. **Spiegel, Murray R., Schiller, John y Srinivasan, R. Alu.** *Probabilidad y estadística.* s.l. : Mc Graw Hill, 2003.
12. *Aplicación de métodos de comparaciones múltiples en Biotecnología Vegetal.* **Casas, Cardoso Gladys y Veitia, Novisel.** 2005, Biotecnlogía Vegetal.
13. **Company, Nielsen.** [En línea] <http://blog.nielsen.com>.
14. Comparativa plataformas iOS, Android y Windows Phone. [En línea]
<http://blog.neuronatraining.net/?p=15221>.
15. Symbian. [En línea] <http://www.symbian.org/>.
16. **Android.** Página de desarrollo de Android. [En línea]
<http://developer.android.com/guide/basics/what-is-android.html>.
17. —. Android SDK. [En línea] <http://developer.android.com/sdk/index.html>.
18. **Holzner, Steven.** *La Biblia de Java 2.* s.l. : Anaya, 2005.

19. **Wikipedia.** XML. [En línea] http://es.wikipedia.org/wiki/Extensible_Markup_Language.
20. **Harrington, Jan L.** *SQL Clearly Explained*. s.l. : Elsevier, 2010.
21. **Quintana, G., y otros.** *Aprende SQL*. s.l. : Colecció Treballs D'Informàtica I Tecnologia, 2008.
22. **Eclipse.** <http://www.eclipse.org/>. [En línea] <http://www.eclipse.org/>.
23. **Wikipedia.** Eclipse (Software). [En línea] [http://es.wikipedia.org/wiki/Eclipse_\(software\)](http://es.wikipedia.org/wiki/Eclipse_(software)).
24. **Ableson, Frank W., Sen, Robi y King, Chris.** *Android in Action*. s.l. : Manning Publications, 2011.
25. **SQLite.** SQLite. [En línea] <http://www.sqlite.org/lang.html>.
26. **AndroidPlot.** AndroidPlot. [En línea] <http://androidplot.com/wiki/Home>.
27. **IBM.** IBM SPSS Statistics. [En línea] <http://www-01.ibm.com/software/es/analytics/spss/>.
28. **Salafranca Cosials, Lluís, y otros.** *Análisis estadístico mediante aplicaciones informáticas: SPSS, STATGRAPHICS, MINITAB Y EXCEL*. s.l. : Universitat de Barcelona, 2005.
29. **HTC.** www.htc.com. [En línea]
http://www.htc.com/es/product/desirehd/overview.html?utm_source=google&utm_medium=cpc&utm_term=HTC%20desire%20HD&utm_campaign=Model+-+HTC+Desire+HD&utm_content=sGfN342d1!pcrid!13485446505.
30. **Polo, M.** *Apuntes de Ingeniería del Software II*. s.l. : UCLM, Facultad de Informática, 2008.
31. **Rogers, Rick, y otros.** *Android Application Development*. s.l. : O'Reilly, 2009.