Parte 1 – Descripción del proyecto

**Lista de cambios**

|  |  |  |  |
| --- | --- | --- | --- |
| **Versión** | **Fecha** | **Descripción** | **Autor** |
| 1.1 | 11/10/2011 | Esqueleto inicial | Romina Liuzzi |
| 1.2 | 21/04/2012 | Ampliación de apartado de introducción | Romina Liuzzi |
| 1.3 | 22/04/2012 | Ampliación de apartado de objetivos | Romina Liuzzi |
| 1.4 | 22/04/2012 | Reestructuración de apartado conceptos teóricos | Romina Liuzzi |
| 1.5 | 22/04/2012 | Incluir esqueleto de técnicas y herramientas | Romina Liuzzi |
| 1.6 | 28/04/2012 | Completar apartado de conceptos teóricos | Romina Liuzzi |
| 1.7 | 16-06-2012 | Revisión final | Romina Liuzzi |

Tabla 1 Lista de cambios

Tabla de contenidos

[Introducción 7](#_Toc327829159)

[Objetivos del proyecto 10](#_Toc327829160)

[Objetivos marcados por los requisitos funcionales 10](#_Toc327829161)

[Objetivos de carácter técnico 11](#_Toc327829162)

[Conceptos teóricos 13](#_Toc327829163)

[Paradigma de la programación por capas 14](#_Toc327829164)

[Capas y niveles 15](#_Toc327829165)

[Beneficios de esta solución 17](#_Toc327829166)

[Modelo Vista Controlador 18](#_Toc327829167)

[El modelo y sus componentes 18](#_Toc327829168)

[Comunicación entre componentes 19](#_Toc327829169)

[Beneficios de la aplicación del Patrón MVC 20](#_Toc327829170)

[BMI (Índice de Masa Corporal) 22](#_Toc327829171)

[Técnicas y herramientas 24](#_Toc327829172)

[Arquitectura Android 24](#_Toc327829173)

[Aplicación 25](#_Toc327829174)

[Actividad 26](#_Toc327829175)

[Vista 27](#_Toc327829176)

[Ciclo de vida de una Actividad 29](#_Toc327829177)

[Sensores integrados en Smartphones 31](#_Toc327829178)

[Desarrollo 33](#_Toc327829179)

[Eclipse 33](#_Toc327829180)

[Plugin: eGit 33](#_Toc327829181)

[Plugin: ADT 34](#_Toc327829182)

[Persistencia de datos en Android 39](#_Toc327829183)

[SQLite Manager 41](#_Toc327829184)

[Versionado 42](#_Toc327829185)

[gitHub 42](#_Toc327829186)

[Pruebas unitarias y funcionales 44](#_Toc327829187)

[JUnit 3.0 44](#_Toc327829188)

[Google Earth 45](#_Toc327829189)

[Librerías externas 46](#_Toc327829190)

[AChartEngine 46](#_Toc327829191)

[Joda-Time 47](#_Toc327829192)

[Google MapView 47](#_Toc327829193)

[Servidores 48](#_Toc327829194)

[Apache Tomcat 48](#_Toc327829195)

[Publicación de Aplicación 49](#_Toc327829196)

[Android Market (Google Play) 49](#_Toc327829197)

[BB Flahsback screen recorder 50](#_Toc327829198)

[InkScape 50](#_Toc327829199)

[Documentación 51](#_Toc327829200)

[Aspectos relevantes del desarrollo del proyecto 52](#_Toc327829201)

[Trabajos relacionados 54](#_Toc327829202)

[Conclusiones y líneas de trabajo futuras 56](#_Toc327829203)

## Tabla de Figuras

[Figura 1 Arquitectura de 3 capas 14](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829071)

[Figura 2 Arquitectura de software por capas 17](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829072)

[Figura 3 Patrón MVC 19](#_Toc327829073)

[Figura 4 Arquitectura SO Android 24](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829074)

[Figura 5 Vistas 27](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829075)

[Figura 6. Una vista en nuestra aplicación 28](#_Toc327829076)

[Figura 7 Ciclo de vida de una actividad 29](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829077)

[Figura 8. Eclipse ADT. Android SDK Manager 34](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829078)

[Figura 9. Eclipse ADT. Android AVD Manager 35](#_Toc327829079)

[Figura 10. Características técnicas definidas a través del wizard 35](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829080)

[Figura 11 Esquema de resoluciones relativas estandarizadas 36](#_Toc327829081)

[Figura 12 Vista de un emulador estándar 37](#_Toc327829082)

[Figura 13. GitHub herramienta integrada de gestión de incidencias 43](#_Toc327829083)

[Figura 14 GitHub estadística de frecuencia de código 44](#_Toc327829084)

[Figura 15. Google Earth capturar ruta de longitud determinada 45](#_Toc327829085)

[Figura 16. AChartEngine biblioteca de gráficos 46](#_Toc327829086)

[Figura 17. Google Maps con overlays 47](file:///C:\Users\Romina\eCoach\Parte1_Descripcion_del_Proyecto.docx#_Toc327829087)

[Figura 18. Google Developer Console. Proceso de publicación 49](#_Toc327829088)

[Figura 19 Ventas por SO. Datos de Gartner. Abril 2011 53](#_Toc327829089)

# Introducción

El presente proyecto comprende dos partes fundamentales. En primer lugar un estudio previo de la plataforma, entorno de desarrollo y técnicas y herramientas necesarias para desarrollar y publicar una aplicación móvil en Android. Y en segundo lugar el diseño e implementación de una aplicación que nos permita reflejar de una manera práctica e intuitiva los resultados obtenidos en la primera fase. Asimismo la aplicación desarrollada servirá al usuario final para medir los progresos obtenidos a partir de un programa de ejercicios.

En el presente documento se reflejarán ambos apartados, siguiendo la estructura que se define a continuación.

**Primera parte. Descripción del proyecto:**

Esta sección del documento contendrá las introducciones teóricas necesarias para facilitar la comprensión del proyecto. Se compone de los siguientes apartados:

* **Introducción:** Breve descripción del proyecto realizado y resumen del contenido de la memoria y documentación técnica del mismo.
* **Objetivos del proyecto:** Se abordarán los objetivos que se pretender conseguir con la realización de este proyecto de fin de carrera.
* **Conceptos teóricos:** Definición de conceptos teóricos básicos requeridos para la puesta en marcha del proyecto.
* **Técnicas y Herramientas:** Se introducirán todas las técnicas y herramientas empleadas en la elaboración del proyecto, desde la parte de investigación hasta el diseño e implementación de la aplicación, así como herramientas auxiliares empleadas en esta misma documentación.
* **Aspectos relevantes del desarrollo del proyecto:** En este apartado se pondrán de relieve aquellos aspectos destacables en la elaboración y desarrollo de este proyecto.
* **Trabajos relacionados:** En este apartado se presentarán aplicaciones comerciales con objetivos similares a la solución adoptada contrastando sus diferencias.
* **Conclusiones y líneas de trabajo futuras:** En este último apartado se comentarán las posibles mejoras que se pueden implementar, tanto para ampliar el proyecto a nivel funcional como a nivel de infraestructura. Así mismo se extraerán aquí las conclusiones derivadas de la consecución del proyecto.

**Segunda parte. Documentación técnica:**

Esta segunda sección del documento contendrá los apartados específicos del análisis, diseño e implementación de la aplicación, así como los distintos manuales. Su estructura será la siguiente:

* **Plan del proyecto:** Se comentará el calendario previsto así como los detalles de planificación.
* **Especificación de requisitos:** En este anexo, se detallarán los requisitos correspondientes a la aplicación desarrollada.
* **Especificación de diseño:** En este anexo se justifican las decisiones tomadas para implementar la solución al problema descrito en la fase anterior.
* **Documentación técnica de programación:** Este anexo estará dedicado a comentar aspectos relevantes en la implementación de la aplicación.
* **Manuales:** Este anexo estará compuesto por un completo manual de la aplicación orientado al usuario final de la misma. Se explicará el proceso de instalación y se tratarán todos los menús y contenidos presentes en la aplicación para facilitar su uso. También se incluye un apartado para el programador, donde se describe el proceso de desarrollo y depuración de la aplicación.

# Objetivos del proyecto

Se distinguen dos tipos de objetivos; los marcados por los requisitos del software o requisitos funcionales y los objetivos de carácter técnico.

### Objetivos marcados por los requisitos funcionales

Se busca diseñar e implementar una aplicación móvil que permita medir el progreso conseguido en distintas rutinas de ejercicios a lo largo del tiempo. Se pretende distribuir la aplicación a través del Android Market con la idea de ser utilizado con fines recreativos una vez completado su desarrollo.

Un usuario de la aplicación podrá realizar distintos tipos de ejercicios que involucren desplazamientos físicos reales. De tal forma que, mediante los cambios en las coordenadas GPS, recogidos a través de los sensores integrados en el terminal, sea posible calcular: la distancia recorrida, el tiempo, la velocidad y las calorías quemadas al finalizar la actividad.

La aplicación deberá ser capaz de explotar los datos obtenidos a partir de los sensores, persistirlos y utilizarlos para presentar al usuario con gráficas y estadísticas inherentes a su progreso. Será imprescindible que el terminal cuente con un sensor GPS integrado y activo para su correcto funcionamiento. Sería posible habilitar y deshabilitar el sensor desde un menú de la propia aplicación.

Al ser una aplicación con fines puramente lúdicos, es aceptable que los algoritmos utilizados en los cálculos presenten valores aproximados.

En cambio es importante que las gráficas presentadas en la aplicación representar de forma atractiva los factores claves para motivar al usuario a seguir haciendo ejercicio. Será posible conocer las velocidades máximas alcanzadas, los mínimos y los promedios. De forma parecida se tratarán las calorías quemadas y peso perdido.

Será posible acceder a un resumen histórico de ejercicios. Se pretende también que sea la propia aplicación la que evalúe el consumo de calorías en función de los datos de usuario. Será importante que estos datos sean recientes por lo que se debe asegurar que son válidos antes de comenzar el ejercicio. En caso que los datos no sean válidos, éstos podrán ser actualizados de forma ágil.

Debe garantizarse el correcto funcionamiento a nivel de seguridad, de rendimiento y a nivel funcional de la aplicación. Para ello se debe definir un proceso de pruebas adecuado.

### Objetivos de carácter técnico

Al desarrollo de este proyecto subyace la necesidad de explotar y ampliar los conocimientos adquiridos durante la carrera. De este modo ha de ser posible estructurar la aplicación en base al modelo de tres capas: datos, lógica y presentación. Las aplicaciones de Android utilizan una aproximación al modelo MVC y es también objetivo del proyecto comprender su funcionamiento.

Para la parte del desarrollo se utilizarán lenguajes Orientados a Objetos, concretamente: Java. En Android el proceso se ejecuta dentro de una máquina virtual, sobre un Kernel de Linux casi completo. Es necesario interiorizar también estas tecnologías para el correcto despliegue de la aplicación.

Además, es también objetivo del proyecto perfeccionar conocimientos sobre:

* El uso de entornos de desarrollo, cómo Eclipse en sus distintas distribuciones (STS, IDE for Java Developers, etc), que mejora notablemente el desarrollo con la integración de los servidores y facilita la utilización de librerías.
* La integración de plugins del IDE para asistir el desarrollo, por ejemplo, para Android: Android ADT que incluye las herramientas del SDK y el AVD, entre muchas otras funcionalidades.
* Las librerías externas que permiten completar las funcionalidades ofrecidas por la aplicación. Ej. AChartEngine, Joda Time, Google Maps API.
* Profundizar sobre opciones y buenas prácticas a la hora de persistir datos en Android.
* Adquirir conocimiento sobre versionado y aprender a utilizar un gestor de versiones distribuido (gitHub).
* Comprender las diferencias, respecto a un entorno web, a la hora de crear una Suite de pruebas unitarias dentro de un proyecto Android.
* Comprender el proceso de firma y publicación de una aplicación Android incluyendo su comercialización.
* Realizar una buena definición de las pruebas de aceptación y cumplir con lo pautado.

Finalmente, aunque no menos importante, otro de los objetivos marcados a la hora de abordar el proyecto es intentar utilizar el mayor número de herramientas de libre distribución para demostrar que es posible realizar un proyecto semejante con un desembolso económico mínimo.

# Conceptos teóricos

En este apartado, se describen los conceptos teóricos más relevantes aplicados en el proyecto. La investigación previa sobre estos conceptos fue de vital importancia a la hora de tomar decisiones a nivel de diseño e implementación.

Se ha optado por omitir en este apartado algunos conceptos, que se entiende, son conocidos dentro de este contexto.

Entre estos ellos se encuentran:

* Programación Orientada a Objetos
* Bases de Datos Relacionales
* Lenguaje de consulta de BBDD

## Paradigma de la programación por capas

El patrón de arquitecta de la programación por capas en el desarrollo de software tiene como principal objetivo la separación del software en función de su misión lógica. Así, cada capa tiene una única responsabilidad de forma que permite el diseño de arquitecturas escalables fácilmente ampliables.

Cada capa se comunica con otra capa a través de una API de forma que basta con conocer dicha API, abstrayéndose de su implementación. Por otro lado, el impacto en un cambio en una de las capas es mucho menor ya que sólo se pueden ver afectados elementos de la propia capa.

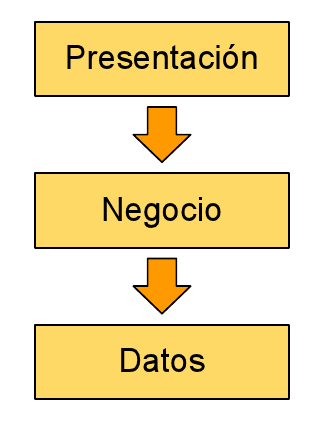


Figura 1 Arquitectura de 3 capas

Se pueden diseñar tantas capas como sean necesarias. Por ejemplo, un nivel en dos capas podría consistir en la separación entre la capa de datos y la capa de presentación. Sin embargo, el diseño más utilizado actualmente es el de tres capas.

### Capas y niveles

Las tres capas en las que se separa la aplicación son:

**1-.Capa de presentación:**

Es la capa del nivel más superior. Su misión principal es la de proveer la interfaz gráfica al usuario para que pueda interactuar con el sistema y para mostrarle los resultados. Esta capa únicamente se comunica con la capa de negocio (su capa inferior directa).

**2-.Capa de negocio:**

Esta capa coordina la aplicación, procesa comandos, toma decisiones y realiza evaluaciones lógicas. Es aquí donde se definen las reglas de negocio de la aplicación. Se comunica con la capa de presentación para recibir las peticiones de los usuarios y para mostrar los resultados. Se comunica con la capa de datos para solicitar los datos que necesite o para almacenar o modificar datos.

**3-.Capa de datos:**

Esta es la capa del nivel más inferior. Se utiliza para guardar y consultar la información necesaria para la aplicación. Esta información se pasa a la capa de negocio que será la encargada de dársela a la capa de presentación para que ésta la muestre al usuario. Esta capa es la que decide cómo se guardan estos datos. Pueden guardarse en una base de datos, en un fichero o en cualquier otro formato que esta capa decida.

En función de la complejidad de cada la capa se pueden distribuir en uno o más ordenadores. Podría llegar a darse el caso que la capa de negocio resida en un solo ordenador y que hay más de uno para la capa de datos. O por el contrario, puede haber un único ordenador para proveer el servicio de la capa de datos y varios para la capa de negocio.

### Beneficios de esta solución

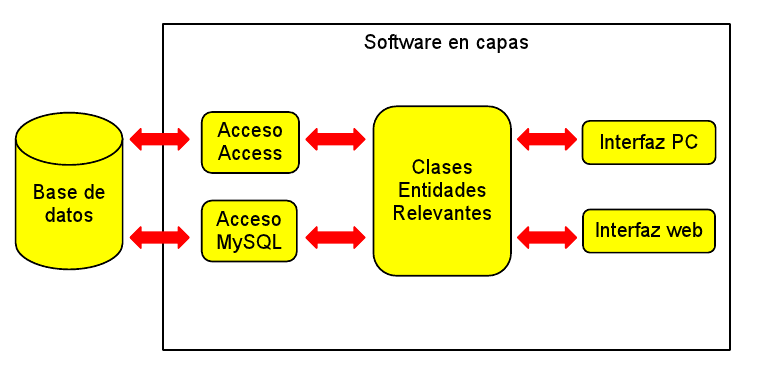
El principal beneficio de este tipo de arquitectura radica en que es posible desarrollar sistemas escalables cuyas capas son independientes y se comunican a través de una API[[1]](#footnote-1). Esto permite, por ejemplo, tener varios sistemas de gestión de bases de datos o incluso realizar una migración o ampliación de forma mucho más eficiente. De cara a grupos de trabajo, esta solución permite paralelizar el desarrollo.

Figura 2 Arquitectura de software por capas

## Modelo Vista Controlador

La unión entre capa de presentación y capa de negocio conocido en el paradigma de la Programación por capas representaría la integración entre Vista y su correspondiente Controlador de eventos y acceso a datos.

MVC no pretende discriminar entre capa de negocio y capa de presentación pero si pretende separar la capa visual gráfica de su correspondiente programación y acceso a datos, algo que mejora el desarrollo y mantenimiento de la Vista y el Controlador en paralelo, ya que ambos cumplen ciclos de vida muy distintos entre sí.

### El modelo y sus componentes

Se trata de un modelo de arquitectura creado en 1979 por Trygve Reenskaug . El patrón permite separar la presentación de datos de la interacción con el usuario y de la lógica del negocio apoyándose en tres componentes:

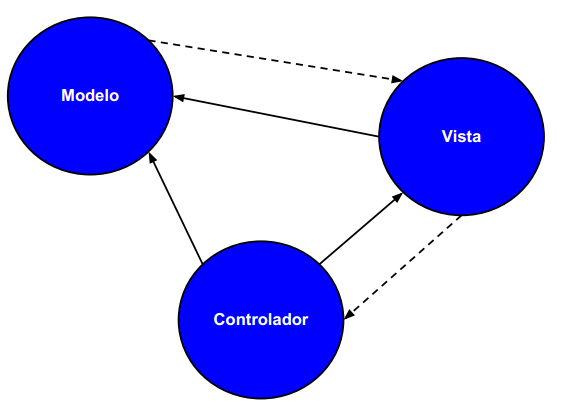


Figura 3 Patrón MVC

**Modelo:** Esta es la representación de los datos y las reglas del negocio. Encapsula los datos y las funcionalidades. El modelo es independiente de cualquier representación de salida y/o comportamiento de entrada.

**Vista:** Muestra la información al usuario. Obtiene los datos del modelo. Pueden existir múltiples vistas del modelo. Cada vista tiene asociado un componente controlador.

**Controlador:** Reciben las entradas, usualmente como eventos que del tipo pulsación de botones del ratón, pulsaciones de teclas, etc. Los eventos son traducidos a peticiones de servicio para el modelo o la vista. El usuario interactúa con el sistema a través de los controladores.

### Comunicación entre componentes

Cuando se registra un evento, por ejemplo, una petición a una página web el MVC lo gestiona de la siguiente manera:

1. Un evento es interceptado por un controlador.
2. El Controlador evalúa el evento y lo “mapea” al controlador apropiado dentro del Modelo.
3. El Modelo lleva adelante la acción o cambia el estado y el resultado es retornado al Controlador.
4. El controlador determina la Vista apropiada para desplegar y provoca que la aplicación se dirija a esa Vista.
5. Una vez cargada, la Vista podría recuperar datos desde el Modelo a través de una interfaz de datos, no pudiendo acceder directamente a los mismos.
6. La Vista es desplegada al usuario.

### Beneficios de la aplicación del Patrón MVC

* Desacopla vistas y modelos. Se puede a llegar a tener múltiples vistas para un mismo modelo.
* Mayor cohesión.
* Especialidad de los elementos (no se mezclan responsabilidades entre el modelo, la vista y el controlador).
* Cada vista puede concentrarse en una parte específica del modelo.
* Facilidad para la creación de nuevas vistas en función del dispositivo o canal (una vista para la web y otra para el móvil o para una aplicación de escritorio).
* Claridad de diseño.
* Facilidad de crecimiento. Pueden crearse vistas nuevas sobre un mismo modelo sin necesidad de borrar las antiguas.
* Facilita el mantenimiento.
* Mayor escalabilidad.

Por lo tanto el patrón MVC representa un mecanismo de mejora de procesos de desarrollo de software, fácil de comprender y aplicar.

## BMI (Índice de Masa Corporal)

Se trata de una relación aritmética entre el peso y la altura de un individuo desarrollada por el científico belga Adolphe Quetelet entre 1830 y 1850.

Fórmula 1. Cálculo de BMI

Esta relación establece unos valores orientativos un tanto arbitrarios que determinan si un individuo sufre delgadez, posee un peso normal o padece sobrepeso u obesidad.

|  |  |
| --- | --- |
| <18.5 | Delgadez |
| 18,5 – 24.9 | Normal |
| 25 – 29,9 | Sobrepeso |
| >30 | Obesidad |

Tabla 2. Rangos de BMI saludables

Los valores límite fueron revisados por el NIH (National Institute of Health) y la WHO (World Health Organization). por última vez a principios de los ´90 donde se fijaron los valores que aún utilizamos, aunque la revisión se realizó sin hacer referencia a ningún estudio científico.

El grado de precisión de esta medida es limitado ya que solo depende del peso y la altura, subestimando la distribución entre músculo y grasa, la edad y el sexo del individuo.

Si bien el grado de precisión del BMI es limitado, aún sigue utilizándose con fines orientativos en conjunción con otras medidas tales como el radio cintura cadera, el porcentaje de grasa corporal, etc.

En este proyecto se utiliza una variante de esta fórmula para calcular el rango de peso mínimo - máximo recomendado para el individuo en función de su altura y peso.

|  |  |
| --- | --- |
| Fórmula 2. Peso mínimo saludable | Fórmula 3. Peso máximo saludable |

# Técnicas y herramientas

## Arquitectura Android

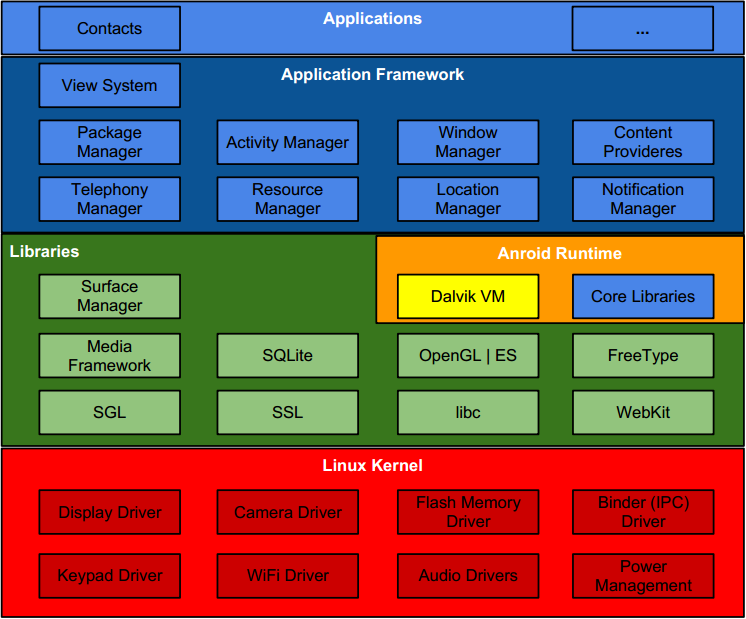
Android es un sistema operativo para dispositivos móviles y tabletas propietario de Google y distribuido bajo el Standard Open Source.

Figura 4 Arquitectura SO Android

Como se puede observar en la imagen, Android se basa en un Kernel Linux que cuenta con un gran número de controladores. Es una capa de abstracción con el hardware, ninguna otra capa se encargará de esto. Incluso la seguridad del dispositivo se controla desde aquí.

En la capa de librerías nos encontramos mayormente con librerías escritas en C y C#. Aquí se maneja un gran número de recursos gráficos. Se utiliza un esquema de base de datos relacional SQLite integrado.

A la hora de implementar nos moveremos sobre la capa de aplicaciones, trabajando únicamente sobre el framework de las aplicaciones. El manejo de los niveles de abstracción inferiores será prácticamente transparente.

### Aplicación

Una aplicación es un set de actividades, servicios, vistas y recursos unidos dentro de un mismo paquete. En Android lo más natural es escribir las aplicaciones en Java y a partir de las herramientas provistas por el SDK generar nuestros ejecutables “.apk”. También es posible generar una aplicación utilizando otros lenguajes de desarrollo, por ejemplo, empieza a ganar popularidad el desarrollo de aplicaciones para Android en HTML5.

Una aplicación de Android sigue un patrón similar al MVC[[2]](#footnote-2) ampliamente utilizado en entornos web.

En general una aplicación puede hacer uso de:

* Actividades - Una idea vaga de este concepto es una pantalla
* Servicios – Servicios ejecutados en background, no requiere una Vista.
* Broadcast Receivers – Componentes que están a la espera de un cambio para responder.
* Content Providers – Proveedores de contenido que a través de una interfaz común hacen una consulta y devuelven un resultado.

### Actividad

Normalmente cada Actividad estará relacionada con una Vista y deberá ser lo más independiente posible. En términos de modularidad, una actividad equivale a un módulo independiente. Si cogemos como modelo el patrón de diseño MVC, la actividad se entendería como una especie de controlador.

Por ejemplo, en la CalorieCalc una de las actividades: EjercicioActualActivity.java se encarga de iniciar un servicio que recibe una actualización cada vez que se registra un cambio en la ubicación. Cuando una actualización es recibida, realiza cálculos sobre la distancia recorrida, calorías quemadas, etc. La actividad interactúa con una base de datos a través de un DAO y finalmente actualiza la vista presentando al usuario toda la información pertinente.

En sí misma esta actividad es casi autónoma. Toda la gestión realizada sobre un ejercicio en curso se realiza desde esta actividad sobre la vista asociada a la misma.

### Vista

El concepto de vista está sobrecargado, de forma que se utiliza tanto para referirse a la interfaz de usuario, a los elementos dentro de la interfaz de usuario, o a una plantilla de estilo.

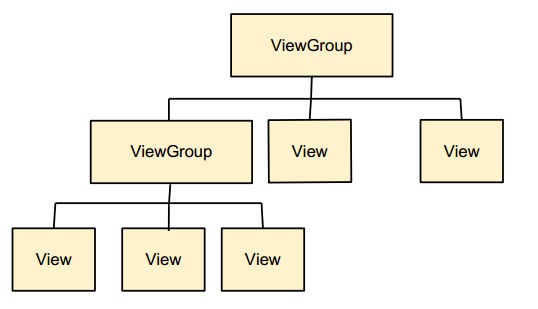
Siguen el Patrón Compuesto[[3]](#footnote-3). La raíz de un ViewGroup puede contener a su vez más ViewGroups. Si cogemos como modelo el patrón de diseño MVC, la actividad se entendería como una vista.

Figura Vistas

Continuando con el ejemplo descrito en el apartado anterior, una vista utilizada en esta aplicación sería: ejercicio\_actual.xml. Cuando nos referimos a esta vista nos podemos estar refiriendo a la pantalla presentada al usuario. Pero también, nos podemos estar refiriendo a un elemento contenido dentro esta pantalla, por ejemplo de tipo TextView tal como se ilustra en la siguiente figura.

|  |  |
| --- | --- |
|  |  |

Figura 6. Una vista en nuestra aplicación

### activityLifecycle.PNGCiclo de vida de una Actividad

Figura 7 Ciclo de vida de una actividad

El esquema anterior representa todos los posibles estados que por los que puede pasar una actividad durante su vida. Estos métodos están implementados en la clase Activity y son heredados en cada una de nuestras actividades de forma transparente. De forma que, en la mayoría de los casos, no necesitamos sobrecargar el método del constructor.

Un caso aparte, es el método onCreate() que debemos implementar siempre; es llamado cuando se inicia la actividad.

Dentro de éste diagrama de flujo es importante notar que existen tres posibles estados en los que se puede “terminar/iniciar” una actividad.

onPause() 🡪 onResume()

onStop() 🡪 onRestart() 🡪 onStart()

onDestroy() 🡪 onCreate()

## Sensores integrados en Smartphones

Un sensor es un dispositivo capaz de detectar magnitudes físicas o químicas, llamadas variables de instrumentación, y transformarlas en variables eléctricas. Las variables de instrumentación pueden ser por ejemplo: intensidad lumínica, distancia, aceleración, inclinación, desplazamiento, presión, fuerza, etc.

Algunos de estos sensores se utilizan en teléfonos desde hace tiempo, otros se han popularizado con el auge de los teléfonos inteligentes. El desarrollo de aplicaciones móviles puede sacar gran provecho de la información provista por estos sensores para aportar a sus usuarios funcionalidades noveles, cálculos más precisos, etc.

A continuación se presenta una selección de los sensores más popularmente integrados en Smartphones recogidos de la especificación provista por fabricantes y distribuidores en los últimos dos años.

|  |  |  |
| --- | --- | --- |
| Magnitud | Nombre | Usos |
| **Aceleración** | Acelerómetro de 3 ejes | Conocer la orientación del terminal y ajustar la posición de la pantalla consecuentemente. |
| **Velocidad lineal y angular** | Giroscopio | Medir ángulo de giro en controladores de juegos. |
| **Sensores fotoeléctricos** | Sensor de luz ambiental , ALS | Ajustar automáticamente el brillo del sistema en base a los niveles de luminosidad del ambiente |
| **Sensor de proximidad** | Sensor de proximidad | Ahorrar energía, bajar el brillo de la pantalla cuando el terminal tiene un objeto muy próximo, ejemplo: se encuentra en un bolsillo o en modo llamadas. |
| **Sensores de presencia**  **(ópticos o capacitivos)** | Sensor táctil, multitouch, etc | Obtener la posición de varios puntos de contacto de manera simultánea |
| **Sensor acústico** | Micrófono | Captura de audio |
| **Sensores inerciales** | GPS | Captura de movimiento |
| Visión artificial | Cámaras de vídeo | Procesamiento digital |

Tabla 3 Sensores habitualmente integrados en Smartphones

Es importante apreciar la diferencia entre un sensor que mide la aceleración y uno que mide la velocidad lineal y angular y comprender la potencia que aporta su utilización conjunta de cara al desarrollo.

Un acelerómetro, mide la aceleración lineal cuando el dispositivo se encuentra derecho, si el ángulo de giro cambia puede confundir la influencia de la gravedad con aceleración lineal.

El giroscopio mide el ángulo de giro, con lo cual las medidas son claras y rápidas.

Si se combinan ambos se obtiene un sistema inercial de medida de 6 ejes, ya que el acelerómetro utiliza la información medida por el giroscopio y luego puede medir sin interferencias la aceleración lineal en cualquier ángulo.

Estos sensores ya se están usando en mandos de consolas como Wii Sport para traducir movimientos complejos. Algunos móviles ya están empezando a incorporarlo (ej. HTC Evo 4G).

Cabe mencionar que en este proyecto se explotó sobre todo el uso del sensor GPS utilizando la clase LocationManager para interactuar con el sensor.

## Desarrollo

### Eclipse

Se trata de una herramienta totalmente trasversal usada a lo largo de todo el ciclo de vida de la aplicación. Resulta complejo resumir todo lo que se ha aprendido durante el transcurso de este proyecto en referencia a esta herramienta.

Algunas de las características que se han explorado son:

* Editor de texto
* Resaltado de sintaxis
* Compilación en tiempo real
* Pruebas unitarias con JUnit
* Control de versiones con eGit
* Integración con Ant
* Asistentes (wizards) para creación de proyectos, clases, test, etc.
* Refactorización

Y en referencia a los plugins, algunos de los que se han utilizado incluyen:

### Plugin: eGit

**Egit** (Plugin de Eclipse para gestionar las versionas con gitHub a través de Eclipse). Las funcionalidades ofrecidas a través de este cliente se explican en el apartado “Versionado” más adelante.

### Plugin: ADT

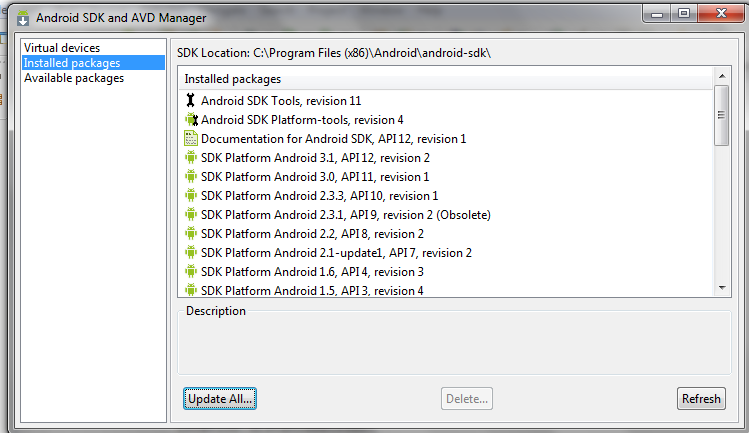
****Android Developer Tools integra una serie de herramientas necesarias para desarrollar, depurar y probar. Este plugin se utilizó, por ejemplo, para instalar todos los sdk que necesarios así como paquetes adicionales desde el mismo IDE.

Figura 8. Eclipse ADT. Android SDK Manager

De esta forma resulta casi trivial mantener actualizados nuestros paquetes con los ultimos updates.

**SDK Tools**: Son herramientas integradas en eclipse tanto como perspectivas, menús o incluso procesos ejecutados en background.

**AVD  Manager**: En este proyecto se ha utilizado esta herramienta para crear, configurar y mantener emuladores con distintos sdk y resoluciones y probar cómo se comporta la aplicación en estos contextos.

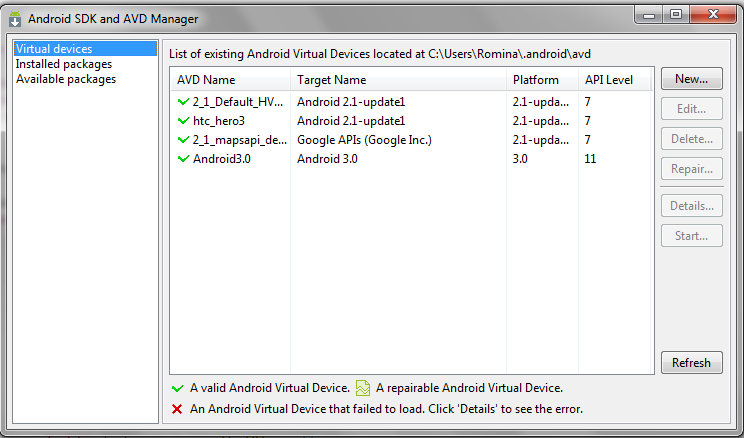


Figura 9. Eclipse ADT. Android AVD Manager

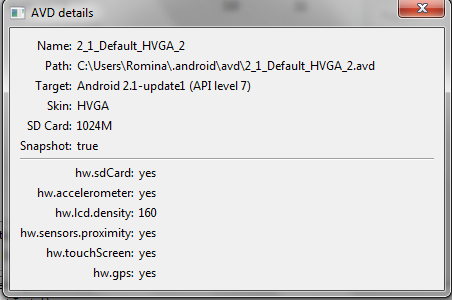
Para crear un emulador nos basta con revisar las especificaciones técnicas del fabricante donde se detallan todas las características aplicables (cámara, tamaño de tarjeta SD, hardware disponible, piel, etc.). A continuación debemos acceder al AVD Manager y replicar todas estas características a través del wizard de creación de un nuevo AVD.

Figura 10. Características técnicas definidas a través del wizard

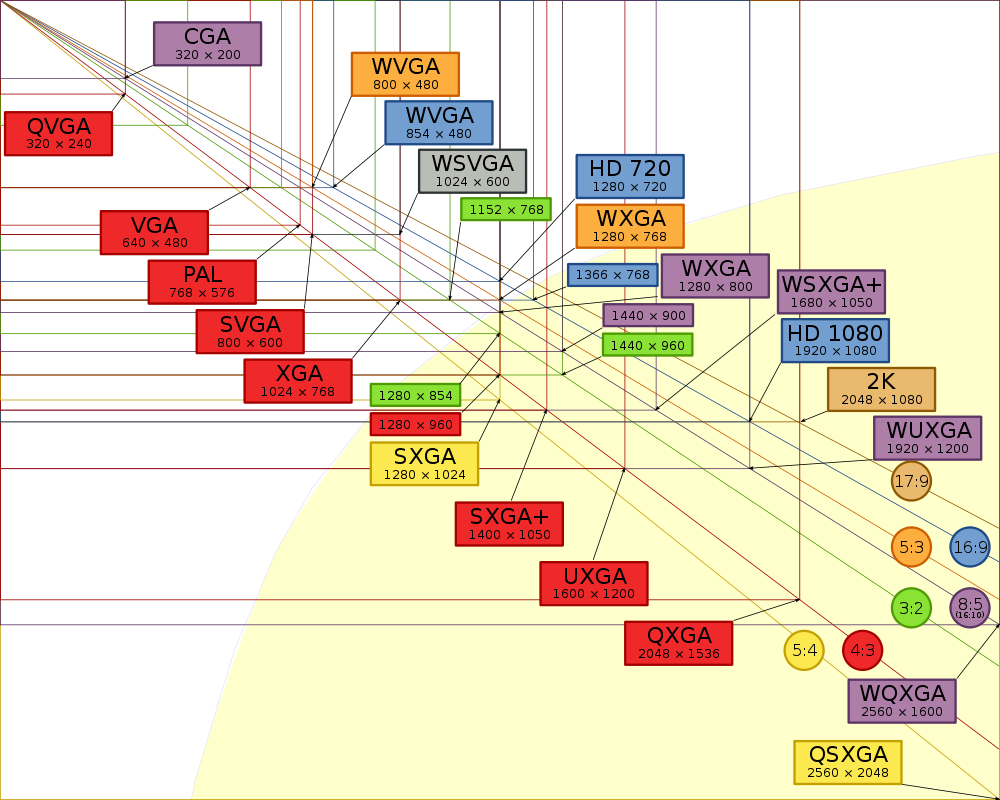
Uno de los aspectos más importantes a tener en cuenta a la hora de configurar un emulador consiste en la resolución, los tamaños más habituales se encuentran recogidos en la siguiente figura. 

Figura 11 Esquema de resoluciones relativas estandarizadas

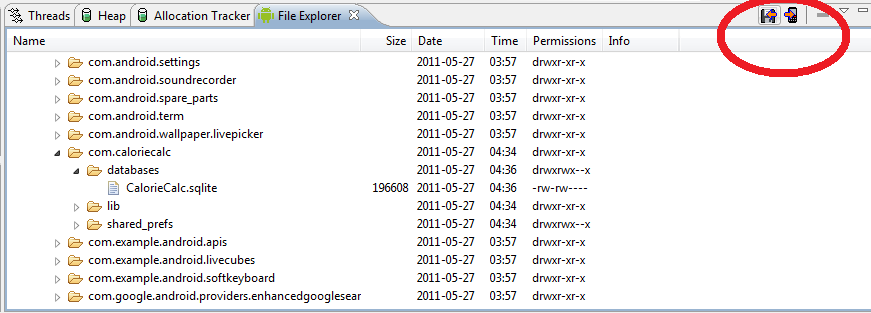


Figura 12 Vista de un emulador estándar

**DDMS**: Dalvik Debug Monitor Server. Perspectiva de eclipse. Gestiona la comunicación tanto con un emulador como con un terminal físico conectado al ordenador en modo debug. En el proyecto lo he usado mucho, para capturar las pantallas mostradas en el terminal, estudiar los logs,  información sobre los threads, explorar ficheros persistidos (bases de datos y sharedPreferences) y sobre todo para enviar coordenadas GPS en ficheros .kml generados con Google Earth.

A través de esta vista podemos monitorizar el estado de los datos de la aplicación a través del explorador de ficheros del DDMS, en el directorio:

Data/data/<nombre del paquete>/databases



Tenemos la opción de bajar datos desde el terminal a nuestro PC desde esta pestaña, también es posible realizar el camino inverso.

### Persistencia de datos en Android

Android ofrece diferentes opciones a la hora de decidir el tipo de almacenamiento que utilizaremos para nuestra aplicación. Una de las opciones presentadas es la creación de un esquema de base de datos relacional SQLite. Todas las operaciones relacionadas con la base de datos se recomienda sean llevadas a cabo desde una clase DBHelper que hace de interfaz entre la base de datos y la aplicación. Una vez se entiende la dinámica, no es difícil entender su funcionamiento.

Durante el transcurso de este proyecto se estudiaron las opciones ofrecidas por Android en cuanto a persistencia de datos, se ha creado una base de datos sencilla, que contiene todos los datos vitales para el funcionamiento de la aplicación.

Toda la capa de comunicación con la base de datos está encapsulada en un paquete DAO donde la clase padre extiende de un objeto SQLiteOpenHelper.

Por otra parte, según la recomendación para salvar parte de nuestros datos utilizamos un método más sencillo que se ajusta a nuestro caso. En Android este método de persistencia se denomina SharedPreferences, básicamente consiste en un HashMap que utiliza un par “key”, “value” y tan solo soporta tipos de datos primitivos (int, char, string o boolean).

En resumen en nuestra aplicación:

**Base de datos interna SQLite**: En la implementación del proyecto se utiliza una base de datos integrada en la aplicación que se guarda en la memoria interna del terminal.

**SharedPreferences**: Se utiliza para enseñar el End User Licence Agreement la primera vez que se ejecuta la aplicación tras la instalación. También la utilizamos para guardar los datos del usuario que no tiene sentido persistir en la base de datos porque:

1. No encajan dentro del esquema
2. Son datos simples que a los que es necesario acceder a nivel de aplicación.

Otra consideración a tener en cuenta en nuestro caso radica en la creación del modelo de datos a nivel local.

La base de datos de la aplicación se crea al instalar por primera vez y se actualiza cada vez que lo deseemos, pero es importante que las versiones sean compatibles tras su lanzamiento. En la primera creación, se creará el esquema bajo el directorio:

Data/data/<nombre del paquete>/databases

### sqlitemanager.PNGSQLite Manager

Plugin de Firefox para crear y mantener la BBDD de la aplicación.

Durante este proyecto se utilizó un **plugin para Firefox** llamado **SQLite Manager**, que permite realizar las operaciones más comunes sobre la base de datos, desde realizar SELECTS, crear VIEWS e incluso crear las bases de datos desde aquí de forma práctica y cómoda.

Una vez tenemos terminado el esquema es posible guardarlo como un fichero .sql y arrastrarlo dentro del contexto de nuestro espacio de trabajo a través de nuestro IDE.

## Versionado

Durante el desarrollo de este proyecto fue necesario recurrir a más herramientas de las que suelen exigirse en proyectos de este nivel, por ejemplo la definición, configuración y utilización de un gestor de versiones. Por motivos laborales la mayor parte de la realización de este proyecto fue llevada a cabo a distancia, de manera que para salvar este obstáculo se recurrió a todo tipo de herramientas que permitieran un seguimiento normal.

### gitHub

Herramienta estudiada en dos fases, primero se investigó la utilización del cliente nativo realizando los submits a través de la línea de comandos. Esta opción no era lo suficientemente cómoda por forzarte a especificar los ficheros que se habían modificado a la hora de subir los cambios. La segunda opción que se investigó para solventar este problema es el cliente que se distribuye como plugin del IDE Eclipse. Esta opción demostró ser una opción mucho más potente, ya que ofrecía una representación gráfica en tiempo real de los ficheros que se habían modificado y estos eran agregados a la lista pendiente de submitar que se poblaba de forma automática.

Otra importante aportación de este gestor de versiones consiste en incluir una herramienta integrada de gestión de incidencias, que envía notificaciones configurables a los colaboradores con cada update.

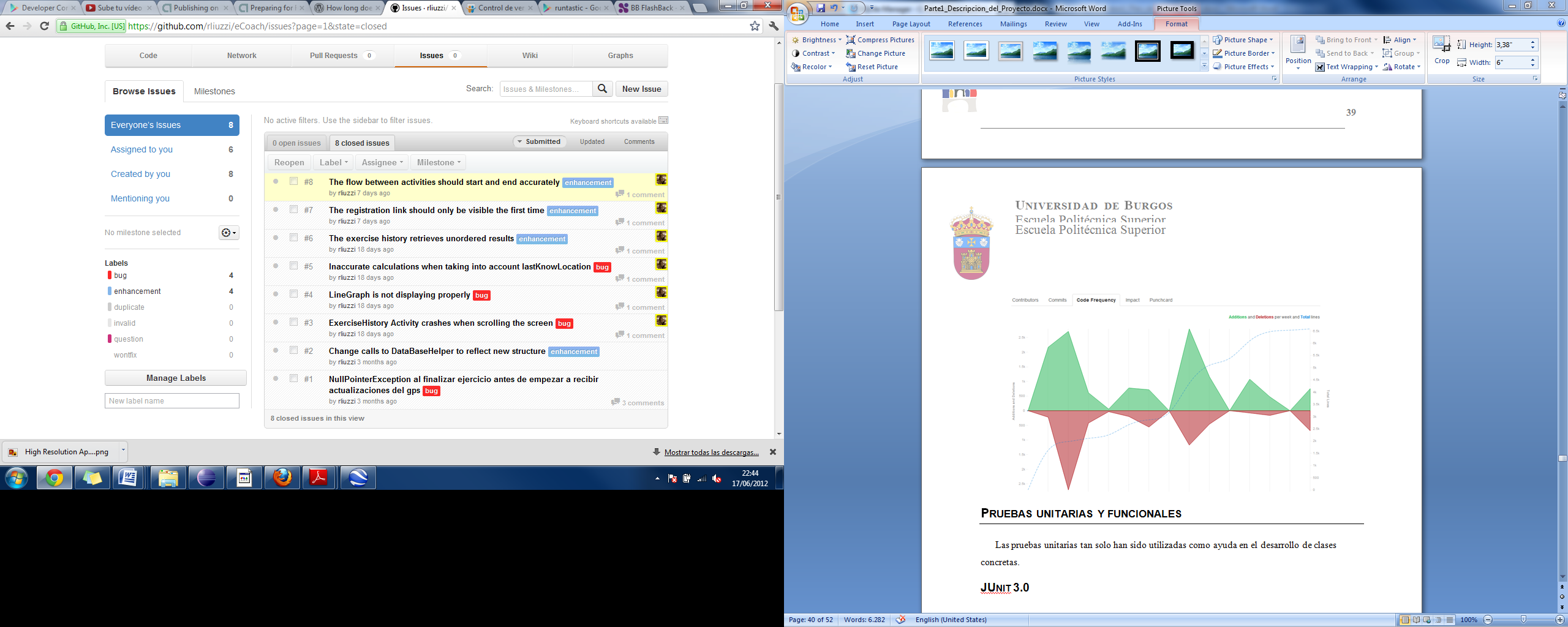


Figura 13. GitHub herramienta integrada de gestión de incidencias

Es interesante también pararse a observar los gráficos generados a partir de los submits que se van realizando donde se mide el riesgo de los cambios a partir de la cantidad de líneas de código agregadas o borradas.

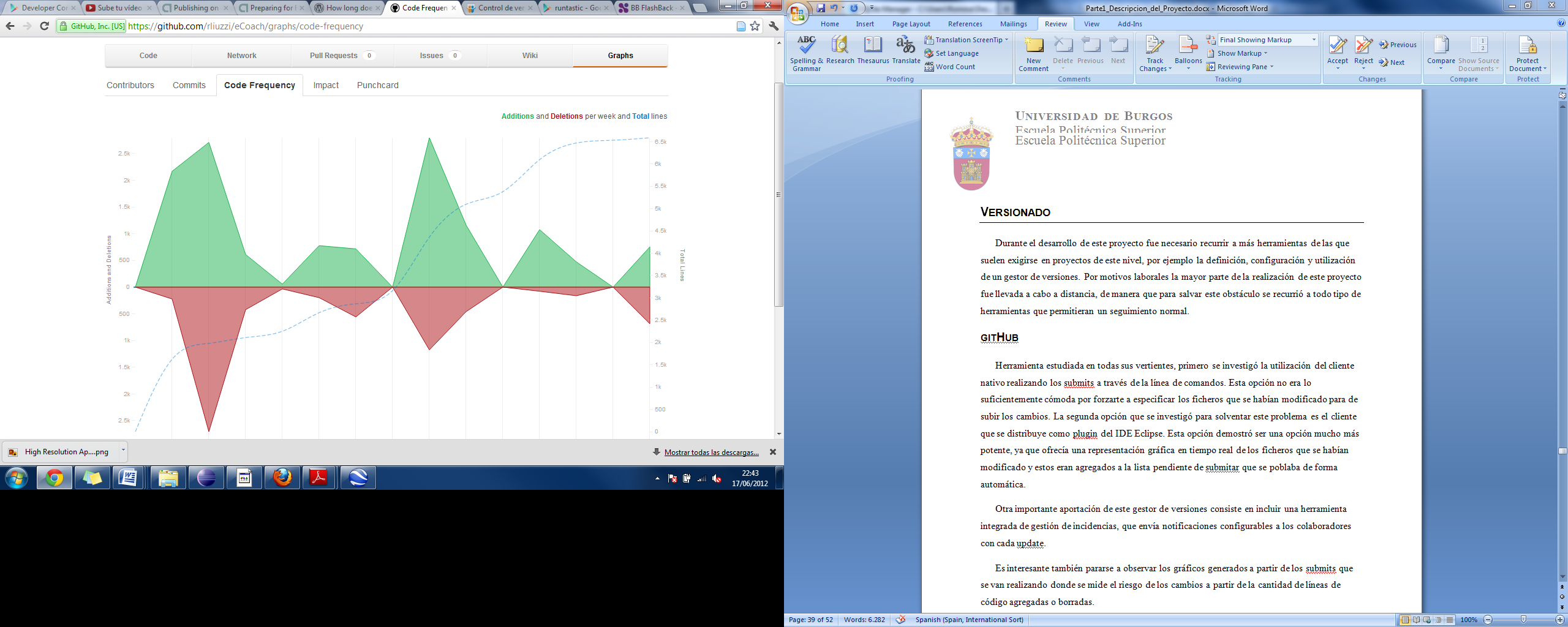


Figura 14 GitHub estadística de frecuencia de código

## Pruebas unitarias y funcionales

Las pruebas unitarias tan solo han sido utilizadas como ayuda en el desarrollo de clases concretas.

### JUnit 3.0

Herramienta utilizada para realizar pruebas unitarias sobre clases/métodos. Cabe particularizar que en Android la suite se guarda en un proyecto aparte, a diferencia de proyectos web donde todo es parte del mismo paquete. Este plugin fue utilizado para probar métodos antes de utilizarlos en la aplicación. Incluye un wizard que permite crear un proyecto de tests con facilidad. Las clases a probar deben luego, ser importadas debido a esta particular distribución (visibilidad a nivel de paquete).

### Google Earth

Se trata de un programa informático que permite visualizar imágenes en 3D del planeta combinando imágenes de satélite, mapas y el motor de búsqueda de Google.

Dentro del ámbito de este proyecto, el programa fue utilizado para crear rutas con ubicaciones simuladas conocidas y exportarlas con formato .kml. Estos ficheros fueron vitales para probar la aplicación en la fase de desarrollo ya que permitían conocer la distancia del recorrido para depurar la aplicación.



Figura 15. Google Earth capturar ruta de longitud determinada

## Librerías externas

### AChartEngine

AChartEngine es una librearía de gráficos que nos permite dibujar gráficos a partir de unos objetos de tipo DataSeries evitando la complejidad de tener que generar todo el código, simplemente limitándonos a interactuar con su API.

Algunas particularidades que es útil conocer radican en la versatilidad que nos permite a la hora de configurar la presentación de multitud de tipos de gráficos (de barras, de pastel, de dispersión, de dial, etc.).

|  |  |
| --- | --- |
| device-2012-06-17-204840.png | device-2012-06-17-204904.png |

Figura 16. AChartEngine biblioteca de gráficos

### Joda-Time

Esta librera fue utilizada para realizar cálculos relacionados con fechas, calcular edad, etc. El nivel al que se utiliza esta librería es muy superficial, pero en vista de su creciente popularidad se decidió invertir un mínimo de tiempo en conocer algunas de las funcionalidades que ofrece.

En la aplicación se utiliza para calcular la edad a partir de la fecha de nacimiento de forma casi transparente.

### device-2012-06-17-115922.pngGoogle MapView

Figura 17. Google Maps con overlays

Esta librería externa se utiliza para pintar un mapa en la aplicación implementando Overlays. A la hora de pintar el recorrido se exploraron distintas opciones y finalmente se decidió utilizar una superposición de Overlays. Uno utilizado para pintar el recorrido y el otro para marcar puntos de interés con los que el usuario pudiera interactuar, por ejemplo marcas relativas en un punto concreto del recorrido.

## Servidores

### Apache Tomcat

Esta herramienta fue estudiada para montar una página web estática para presentar la aplicación, pero fue sustituida por la página de la aplicación generada en Google Play. La información se encontraba repetida y era mucho más atractiva en la página de Google.

Para el desarrollo de la página estática fue preciso montar un servidor en local y avanzar hasta un nivel de configuración básico. La experiencia resulto muy útil para aprender sobre .jar, .war y la forma en la que se despliega una aplicación web.

## Publicación de Aplicación

### Android Market (Google Play)

La aplicación fue firmada y se encuentra disponible (o en proceso de ser aceptada) en el Android Market, recientemente renombrado Google Play. El proceso de publicación implica registrarse como Desarrollador de Google con una cuenta activa y abonar un arancel único de $25.00.

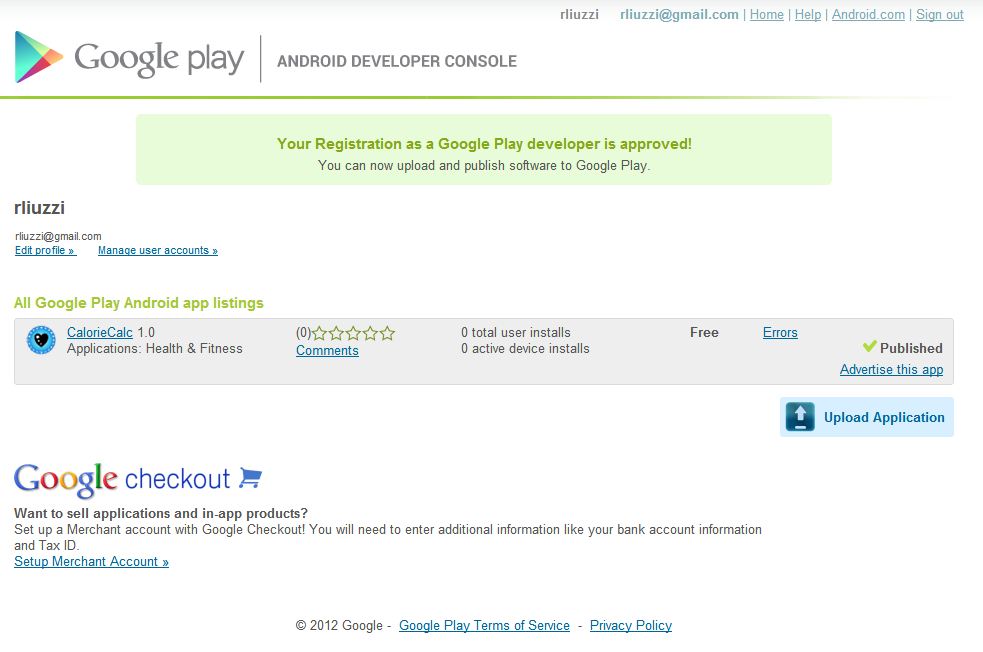


Figura 18. Google Developer Console. Proceso de publicación

### BB Flahsback screen recorder

Este software en su versión de distribución gratuita es suficiente para capturar en formato de video la actividad que toma lugar en la pantalla del ordenador y audio, en un tamaño de marco configurable o pantalla completa. Una vez grabado, es posible exportarlo en formato .avi o .flash.

La versión de pago permite además editar el video, agregar texto, y muchas otras opciones. Es posible en esta versión exportar el video en multitud de formatos.

Para la realización de este proyecto fue suficiente la utilización de la versión gratuita. El video se utilizó para ir reportando el progreso a distancia según las entregas pautadas.

También se utilizó esta herramienta para crear el video opcional demostrando la utilización de la aplicación y subirlo en youtube para ser posteriormente utilizado en la página de promoción de la aplicación en Google Play.

### InkScape

Este programa de distribución gratuita permite crear y editar gráficos de forma sencilla. Algunos de los archivos gráficos incluidos en la aplicación fueron creados con esta herramienta. Otros fueron obtenidos en repositorios gratuitos y retocados. También se utilizó esta herramienta para generar el material promocional asociado a la aplicación para su correcta publicación.

## Documentación

Para la gestión de la documentación del proyecto fue preciso utilizar herramientas variadas cuya naturaleza no entra dentro del alcance de este proyecto. A título enumerativo, las herramientas utilizadas para documentar durante todo el cicle de vida de la aplicación incluyen:

* Microsoft Office Word 2007
* Microsoft Office Excel 2007
* Microsoft Office Visio 2007
* Google Draw
* Google Docs
* Google Drive
* IrfanView

# Aspectos relevantes del desarrollo del proyecto

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| OS | 2009 | 2010 | 2011 | 2012 | 2015 |
| Symbian | 46.9 | 37.6 | 19.2 | 5.2 | 0.1 |
| Android | 3.9 | 22.7 | 38.5 | 49.2 | 48.8 |
| Research in Motion (Blackberry) | 19.9 | 16.0 | 13.4 | 12.6 | 11.1 |
| iOS | 14.4 | 15.7 | 19.4 | 18.9 | 17.2 |
| Microsoft | 8.7 | 4.2 | 5.6 | 10.8 | 19.5 |
| Otros | 6.1 | 3.8 | 3.9 | 3.4 | 3.3 |

Tabla 4 Ventas por SO. Datos de Gartner. Abril 2011

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

Figura 19 Ventas por SO. Datos de Gartner. Abril 2011

# Trabajos relacionados

Actualmente podemos encontrar algunas aplicaciones muy conocidas y bien reputadas que se encuentran en la línea del desarrollo llevado a cabo en este proyecto. En general el tema está muy de moda, de forma que es posible que el número de este tipo de aplicaciones crezca pronto:

|  |  |  |  |
| --- | --- | --- | --- |
| My Tracks | Endomondo Sports Tracker | RunKeeper | runtastic |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | iOS | Android | RIM | Microsoft |
| myTracks | No | Si | No | No |
| endomondo | Si | Si | No | No |
| RunKeeper | Si | Si | No | No |
| Runtastic | Si | Si | Si | Si |

Tabla aplicaciones similares

**Características:**

Todas estas aplicaciones disponen de funcionalidades básicas para llevar a cabo un control de los entrenamientos en tiempo real, como duración, distancia, velocidad, ritmo, calorías consumidas o frecuencia cardíaca. En todos los casos para poder disponer de las mediciones del ritmo cardíaco es necesario adquirir de forma separada un monitor de frecuencia cardiaca que generalmente trasmite las actualizaciones via señales bluetooth.

Por otra parte todas hacen uso intensivo del GPS por lo que el gasto de batería es elevado durante el entrenamiento y solo son aptas para ejercicios al aire libre donde la recepción de los datos del GPS sea óptima.

Todas ellas se caracterizan por ofrecer estadísticas relacionadas a los ejercicios realizados con el objetivo de ayudar a planificar un programa de entrenamientos.

Otro punto que tienen en común es que se presentan en dos versiones, gratis y Premium. Normalmente la versión Premium ofrece más funcionalidades a la vez que prescinde de los anuncios optimizando así el gasto de la batería.

Además de las características comunes descritas anteriormente cada una se especializa en alguna de las vertientes comerciales, por ejemplo Endomondo es la más social y ofrece la página más completa, Runastic soporta el mayor número de actividades, Runkeeper está planeada para usuarios nóveles y MyTracks se especializa en la gestión de rutas que pueden ser exportadas en formato .kml en visualizadas en numerosas plataformas, por ejemplo, Google Maps.

# Conclusiones y líneas de trabajo futuras

En las líneas del trabajo realizado surgen una serie de ampliaciones funcionales que se dejan planteadas de cara a trabajo futuros.

Si bien se trata de aplicaciones muy completas, es posible ir agregando funcionalidades de forma iterativa al modelo ofrecido en este PFC como prototipo inicial.

* Servidor REST para persistir los datos y mantenerlos sincronizados, completando las funcionalidades ofrecidas a usuarios móviles a través de un portal web.
* Soportar monitores de ritmo cardíaco inalámbricos. Sería preciso implementar una interfaz para recibir actualizaciones de ritmo cardíaco vía bluetooth u otras señales de radio.
* Convertir la aplicación en multiusuario y filtrar los resultados enseñados en función del usuario seleccionado. La utilidad de esta mejora está pendiente de evaluar, ya que, normalmente un dispositivo móvil es monousuario.

1. API: De sus siglas en inglés, Interfaz de Programación de Aplicaciones. Una API es una especificación utilizada como interfaz entre componentes software para comunicarse entre sí. [↑](#footnote-ref-1)
2. MVC: Modelo Vista Controlador. [↑](#footnote-ref-2)
3. De la definición en inglés “composite pattern” [↑](#footnote-ref-3)