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 |

Tabla Lista de cambios

Tabla de contenidos

[Introducción 6](#_Toc323457025)

[Objetivos del proyecto 9](#_Toc323457026)

[Objetivos marcados por los requisitos funcionales 9](#_Toc323457027)

[Objetivos de carácter técnico 10](#_Toc323457028)

[Conceptos teóricos 12](#_Toc323457029)

[Paradigma de la programación por capas 13](#_Toc323457030)

[Modelo Vista Controlador 17](#_Toc323457031)

[Arquitectura Android 20](#_Toc323457032)

[Aplicación 21](#_Toc323457033)

[Actividad 22](#_Toc323457034)

[Vista 23](#_Toc323457035)

[Ciclo de vida de una Actividad 24](#_Toc323457036)

[Sensores integrados en Smartphones 26](#_Toc323457037)

[Técnicas y herramientas 28](#_Toc323457038)

[Técnicas 28](#_Toc323457039)

[PERSISTENCIA DE DATOS 28](#_Toc323457040)

[PROGRAMACIÓN POR CAPAS 28](#_Toc323457041)

[Herramientas 28](#_Toc323457042)

[DESARROLLO 28](#_Toc323457043)

[PERSISTENCIA DE DATOS 29](#_Toc323457044)

[VERSIONADO 29](#_Toc323457045)

[PRUEBAS UNITARIAS Y FUNCIONALES 30](#_Toc323457046)

[LIBRERIAS EXTERNAS 30](#_Toc323457047)

[SERVIDORES 31](#_Toc323457048)

[PUBLICACIÓN DE APP 31](#_Toc323457049)

[ADT de Android 31](#_Toc323457050)

[Base de datos integrada SQLite 36](#_Toc323457051)

[Aspectos relevantes del desarrollo del proyecto 40](#_Toc323457052)

[Trabajos relacionados 42](#_Toc323457053)

[Conclusiones y líneas de trabajo futuras 43](#_Toc323457054)

# 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. Una aproximación al modelo MVC adaptado para Android.

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.
* Profundizar en el uso de servidores Web. Concretamente Apache Tomcat, pero extrapolable a otras plataformas.
* Aprender sobre la tecnología Java JSP (Java Server Pages), que proporciona de una forma rápida y sencilla la creación de páginas Web con contenido dinámico.
* 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

La programación por capas es una arquitectura cliente-servidor en el que el objetivo primordial es la separación de la lógica de negocios de la lógica de diseño; un ejemplo básico de esto consiste en separar la capa de datos de la capa de presentación al usuario.

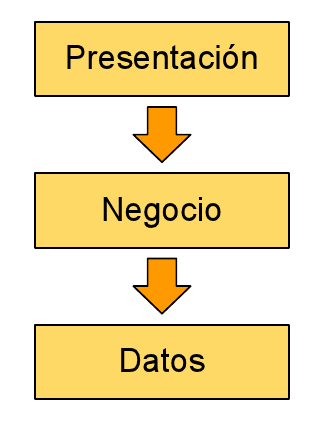
La ventaja principal de este estilo es que el desarrollo se puede llevar a cabo en varios niveles y, en caso de que sobrevenga algún cambio, sólo se ataca al nivel requerido sin tener que revisar entre código mezclado.

Figura Arquitectura de 3 capas

Además, permite distribuir el trabajo de creación de una aplicación por niveles; de este modo, cada grupo de trabajo está totalmente abstraído del resto de niveles, de forma que basta con conocer la API que existe entre niveles.

En el diseño de sistemas informáticos actual se suelen usar las arquitecturas multinivel o Programación por capas. En dichas arquitecturas a cada nivel se le confía una misión simple, lo que permite el diseño de arquitecturas **escalables** (que pueden ampliarse con facilidad en caso de que las necesidades aumenten).

Capas y niveles

El diseño más utilizado actualmente es el diseño en tres niveles (o en tres capas).

**Capa de presentación:** es la que ve el usuario (también se la denomina "capa de usuario"), presenta el sistema al usuario, le comunica la información y captura la información del usuario en un mínimo de proceso (realiza un filtrado previo para comprobar que no hay errores de formato). También es conocida como interfaz gráfica y debe tener la característica de ser "amigable" (entendible y fácil de usar) para el usuario. Esta capa se comunica únicamente con la capa de negocio.

**Capa de negocio:** es donde residen los programas que se ejecutan, se reciben las peticiones del usuario y se envían las respuestas tras el proceso. Se denomina capa de negocio (e incluso de lógica del negocio) porque es aquí donde se establecen todas las reglas que deben cumplirse. Esta capa se comunica con la capa de presentación, para recibir las solicitudes y presentar los resultados, y con la capa de datos, para solicitar al gestor de base de datos almacenar o recuperar datos de él. También se consideran aquí los programas de aplicación.

**Capa de datos:** es donde residen los datos y es la encargada de acceder a los mismos. Está formada por uno o más gestores de bases de datos que realizan todo el almacenamiento de datos, reciben solicitudes de almacenamiento o recuperación de información desde la capa de negocio.

Todas estas capas pueden residir en un único ordenador, si bien lo más usual es que haya una multitud de ordenadores en donde reside la capa de presentación (son los clientes de la arquitectura cliente/servidor). Las capas de negocio y de datos pueden residir en el mismo ordenador, y si el crecimiento de las necesidades lo aconseja se pueden separar en dos o más ordenadores. Así, si el tamaño o complejidad de la base de datos aumenta, se puede separar en varios ordenadores los cuales recibirán las peticiones del ordenador en que resida la capa de negocio.

Si, por el contrario, fuese la complejidad en la capa de negocio lo que obligase a la separación, esta capa de negocio podría residir en uno o más ordenadores que realizarían solicitudes a una única base de datos. En sistemas muy complejos se llega a tener una serie de ordenadores sobre los cuales corre la capa de negocio, y otra serie de ordenadores sobre los cuales corre la base de datos.

En una arquitectura de tres niveles, los términos "capas" y "niveles" no significan lo mismo ni son similares. El término "capa" hace referencia a la forma como una solución es segmentada desde el punto de vista lógico. En cambio, el término "nivel" corresponde a la forma en que las capas lógicas se encuentran distribuidas de forma física.

Beneficios de esta solución

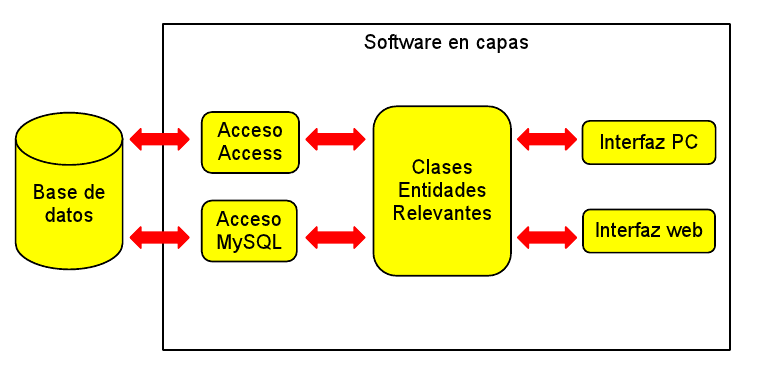
El principal beneficio de este tipo de arquitectura radica en que es posible desarrollar sistemas escalables cuyas capas son independientes y se comunican a través de una API[[1]](#footnote-1). Esto permite, por ejemplo, tener varios sistemas de gestión de bases de datos o incluso realizar una migración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

Figura 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:

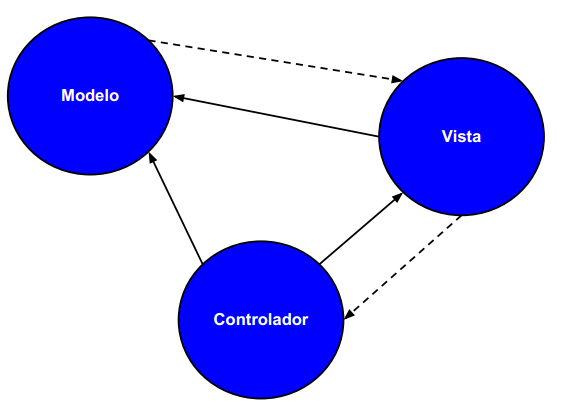


Figura 3 Patrón MVC

**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

* **Menor acoplamiento:**
  + Desacopla las vistas de los modelos.
  + Desacopla los modelos de la forma en que se muestran e ingresan los datos.
* **Mayor cohesión.**
* Cada elemento del patrón está altamente especializado en su tarea (la vista en mostrar datos al usuario, el controlador en las entradas y el modelo en su objetivo de negocio).
* Las vistas proveen mayor flexibilidad y agilidad.
* Se puede crear múltiples vistas de un único modelo.
* Se puede crear, añadir, modificar y eliminar nuevas vistas dinámicamente.
* Las vistas pueden anidarse.
* Se puede cambiar el modo en que una vista responde al usuario sin cambiar su representación visual.
* Se puede sincronizar las vistas.
* Las vistas pueden concentrarse en diferentes aspectos del modelo.
* Mayor facilidad para el desarrollo de clientes ricos en múltiples dispositivos y canales.
* Una vista para cada dispositivo que puede varias según sus capacidades.
  + Una vista para la Web y otra para aplicaciones de escritorio.
* Más claridad de diseño.
* **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.

Arquitectura Android

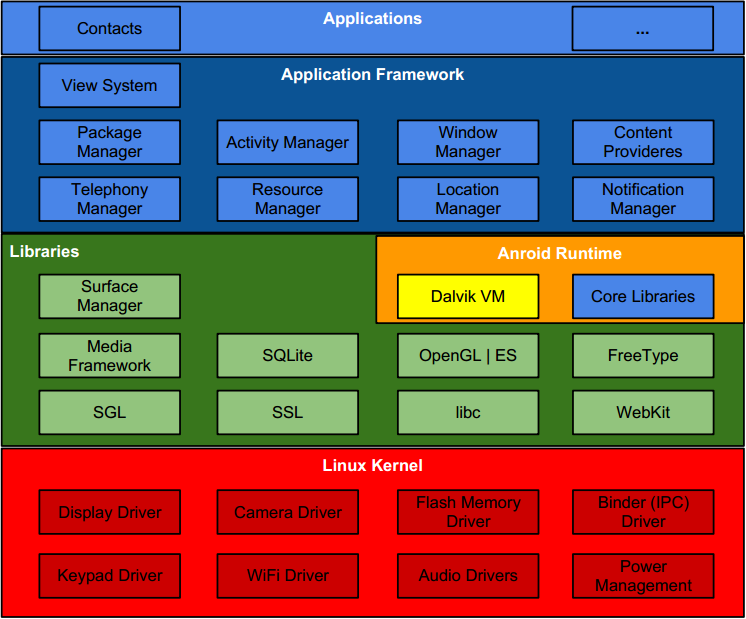
Android es un sistema operativo para dispositivos móviles y tabletas propietario de Google y distribuido bajo el Standard Open Source.

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”.

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

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.

* 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.

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.

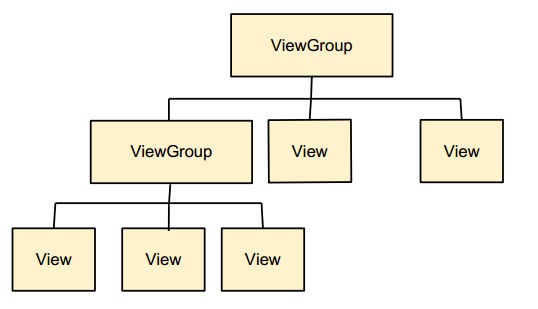
Siguen el Patrón Compuesto[[3]](#footnote-3). La raíz de un ViewGroup puede contener a su vez más ViewGroups. Si cogemos como modelo el patrón de diseño MVC, la actividad se entendería como una vista.

Figura Vistas

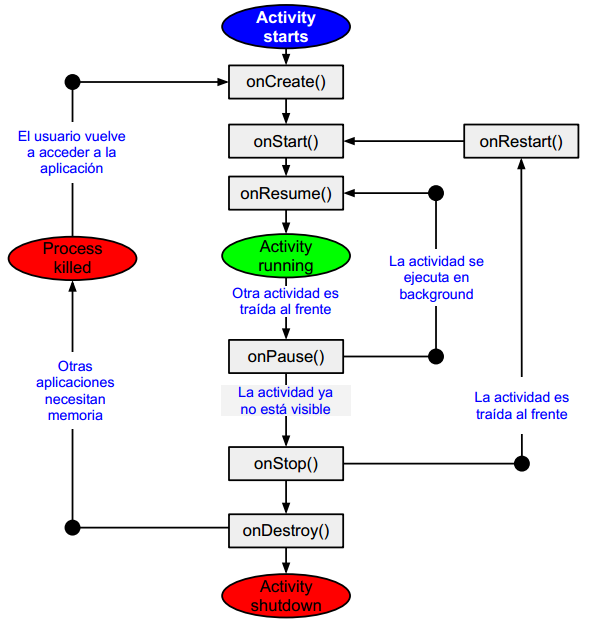
Ciclo de vida de una Actividad

Figura 6 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 2 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).

# Técnicas y herramientas

## Técnicas

### PERSISTENCIA DE DATOS

He estudiado las soluciones ofrecidas por Android antes de empezar a desarrollar y las decisiones están justificadas en la memoria. Creo que el nivel es medio-alto. He 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 SQLiteOpenHelper.

* Sistemas de persistencia de datos en aplicaciones Android
  + SQLite: Tengo una base de datas metida en la aplicación que se guarda en la memoria interna del teléfono.
  + SharedPreferences: Se utiliza para enseñar el End User Licence Agreement la primera vez que se ejecuta la aplicación tras la instalación. También la utilizo 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 necesito acceder a nivel de aplicación.

### PROGRAMACIÓN POR CAPAS

Spring Tool Suit

## Herramientas

### DESARROLLO

El nivel al que uso la plataforma de desarrollo es elevado, es una herramienta trasversal que se usa a lo largo de todo el ciclo de vida de la aplicación.

#### Eclipse

* Entorno y herramientas de Desarrollo: **Eclipse** con los siguientes plugins:

#### Plugin: eGit

* + **Egit** (Plugin de Eclipse para gestionar las versionas con gitHub a través de Eclipse)

#### Plugin: ADT

* + **ADT**: Android Developer Tools. Integra una serie de herramientas necesarias para desarrollar, depurar y probar.
    - **SDK Tools**: Integradas en eclipse como perspectivas, menus o procesos ejecutados en background.
    - **AVD  Manager**: Para crear emuladores con distintos SO y resoluciones y probar como se comporta la aplicación.
    - **DDMS**: Dalvik Debug Monitor Server. Perspectiva de eclipse. 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.

### PERSISTENCIA DE DATOS

#### SQLite Manager

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

### VERSIONADO

Nivel intermedio. Con esta herramienta me pasa lo mismo que con eclipse, he aprendido sobre la herramienta pero no sé como incluirla en la documentación.

* Gestor de versiones distribuido: gitHub

#### gitHub

### PRUEBAS UNITARIAS Y FUNCIONALES

Nivel bajo. Realmente solo se las he utilizado como ayuda con la parte de desarrollo.

#### JUnit 3.0

Pruebas unitarias sobre algunas clases. En android la suite se guarda en un proyecto aparte. Me sivió para probar métodos antes de utilizarlos en la aplicación y ver las diferencias con un proyecto de JUnits normales que se guardan dentro del mismo proyecto.

#### Google Earth

Para crear rutas y exportarlas con formato .kml.

### LIBRERIAS EXTERNAS

#### AChartEngine

* Presentar distintos tipos de gráficos a partir de datos de la aplicación utilizando el API desarrollado por AChartEngine. Intermedio.

#### Joda-Time

* Cálculos relacionados con fechas, calcular Edad, etc. El nivel al que se utiliza esta librería es muy superficial, pero pensé que sería bueno incluirla para conocerla un poco, dicen que el estándar va en esta dirección.

#### Google Maps

* Para pintar un mapa en la aplicación. superficial.

### SERVIDORES

#### Apache Tomcat

* Página web de presentación de la aplicación. Tengo montado el servidor en local y todas las páginas son estáticas. nivel medio-bajo. En cualquier caso aprendí sobre .jar, .war y la forma en la que se despliega una aplicación web.

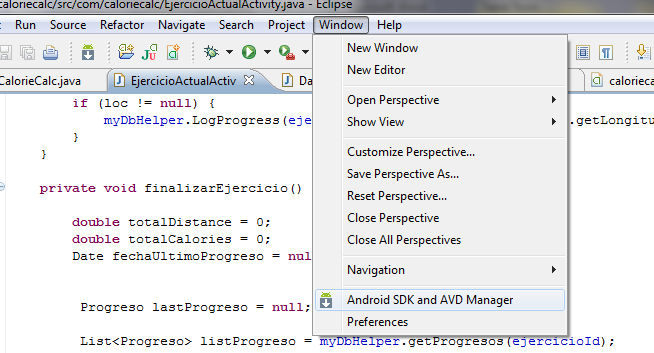
### PUBLICACIÓN DE APP

#### Android Market

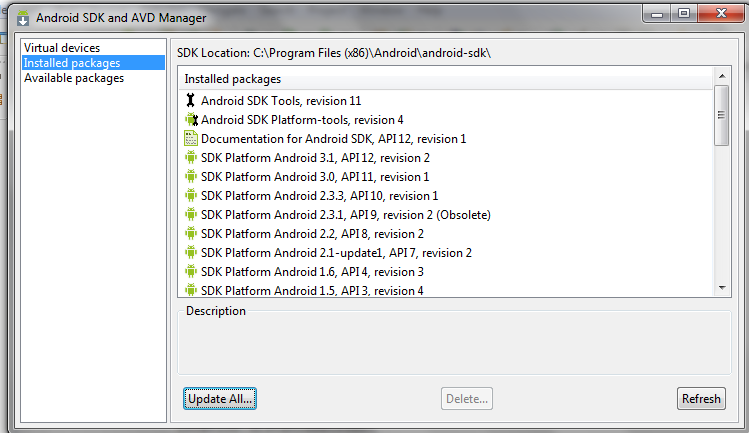
* Para el día de la presentación la aplicación estará firmada y colgada (o en proceso de ser aceptada) en el Android Market. Incluyo el proceso de publicación como parte del proyecto. Nivel bajo.

ADT de Android

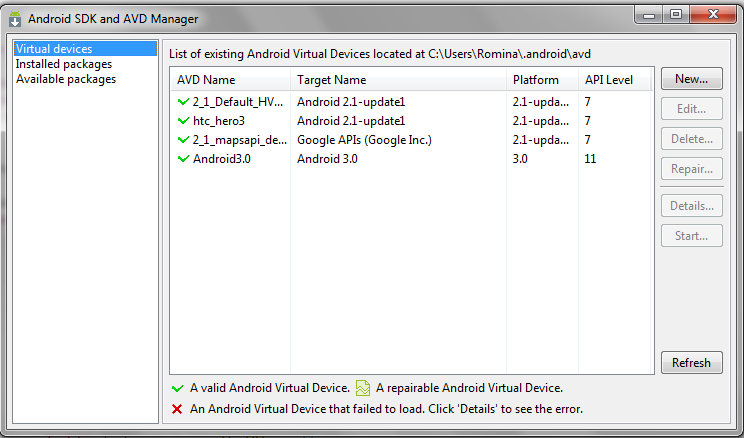
El plugin de Android para eclipse nos permite instalar todos los sdk que necesitemos asi como paquetes adicionales desde el mismo IDE.



Resulta casi trivial mantener actualizados nuestros paquetes con los ultimos updates.



Es realmente sencillo crear un emulador que represente de forma feaciente nuestro modelo de móvil.



Nos basta con revisar las especificaciones técnicas del fabricante donde se detallan todos los datos técnicos que podamos necesitar. Una de las características más importantes a tener en cuenta durante el desarrollo consiste en la piel, los tamaños y configuraciones se pueden revisar en el esquema que se adjunta. De esta manera podemos validar que nuestro diseño de las vistas se adapta correctamente tanto a distintos tamaños de pantalla, interfaces gráficas, cambios de orientación, etc.

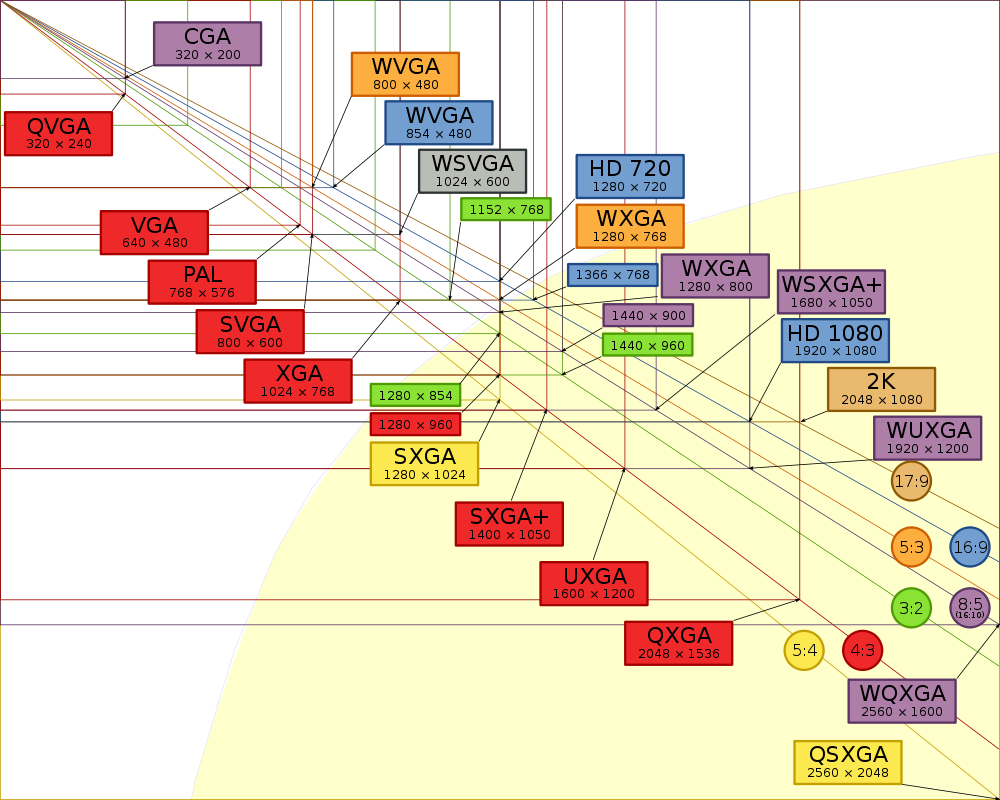
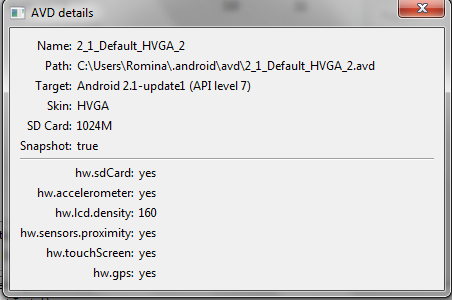
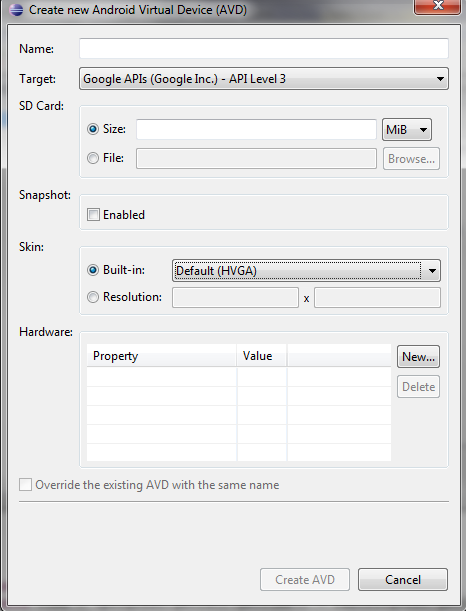


Figura 7 Esquema de resoluciones relativas estandarizadas

El emulador que utilizamos durante el desarrollo de este proyecto así como las características técnica predefinidas durante su creación.







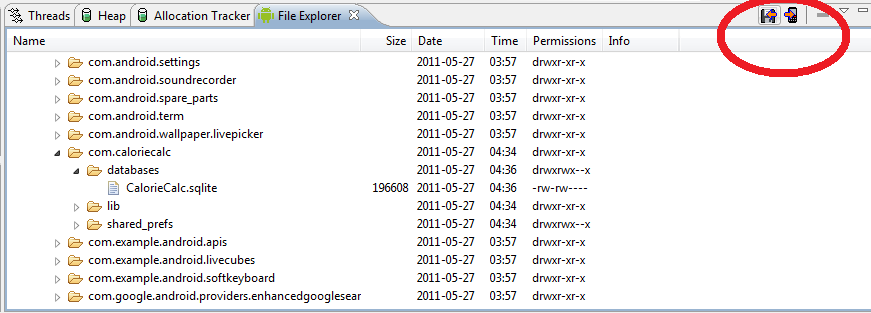
Base de datos integrada SQLite

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.

La base de datos de la aplicación se crea al instalar por primera vez y se actualiza cada vez que lo deseemos. En la primera creación, se creará el esquema bajo el directorio:

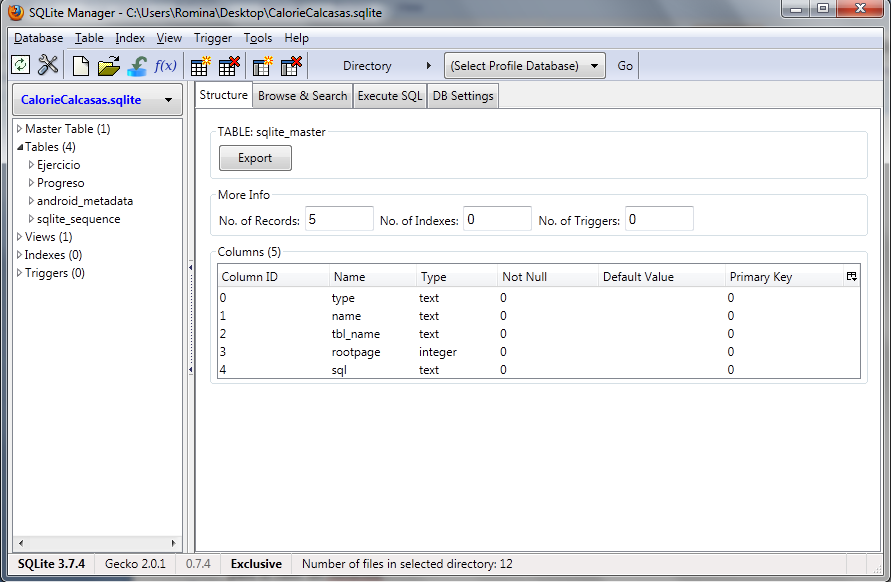
Data/data/<nombre del paquete>/databases

Y podemos monitorizarlo a través del File Explorer:

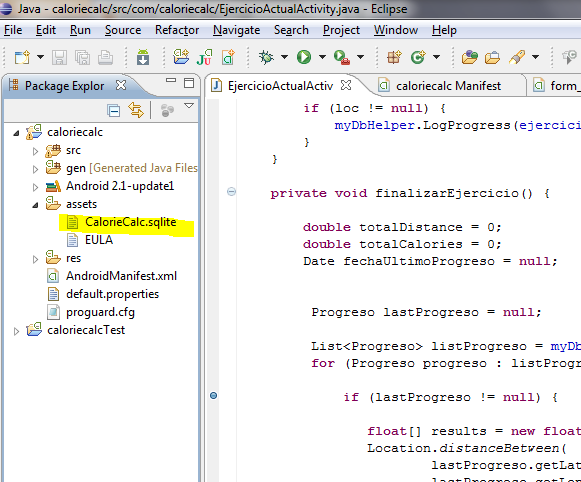


Tenemos la opción de bajar datos desde el terminal a nuestro PC desde esta pestaña, también es posible realizar el camino inverso.

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. En la especificación técnica se explora una alternativa a esta opción.



Una vez tenemos terminado el esquema es posible arrastrarlo dentro del contexto de nuestro espacio de trabajo en la carpeta Assets, de tal forma que al instalar el .apk de nuestra aplicación se cree dentro del registro correspondiente (data/data…).



# 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 3 Ventas por SO. Datos de Gartner. Abril 2011

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

Figura 8 Ventas por SO. Datos de Gartner. Abril 2011

# Trabajos relacionados

Actualmente podemos encontrar tres aplicaciones muy conocidas y bien reputadas que se encuentran en la línea del desarrollo llevado a cabo en este proyecto:

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

Tabla aplicaciones similares

MyTracks:

Esta aplicación permite al usuario guardar sus rutas de deporte, compartirla vía Google maps, generar estadísticas a partir de ellas, así como mostrar por pantalla una serie de datos calculados a partir de la actividad y datos de sensores.

# 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.

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)