

**Víctor Cano Farré**

**BIKETRACKING**

**PROYECTO DE FINAL DE CARRERA**

**dirigido por el Dr. Carlos Molina Clemente**

**Ingeniería Técnica en Informática de Gestión**



**UNIVERSITAT ROVIRA I VIRGILI**

**Tarragona**

**2014**





## Resumen

Actualmente, en la vida diaria, los dispositivos móviles o smartphones ya forman parte importante de nuestra vida. En los últimos años ha aumentado de forma vertiginosa la utilización de todo tipo de aplicaciones para ser utilizadas para mejorar nuestra experiencia en todo tipo de actividades al aire libre. Hoy en día está de moda hacer deporte más que nunca: correr, ir en bicicleta ....

Android es un sistema operativo para dispositivos móviles muy extendido y que goza de un montón de desarrolladores y usuarios. Todo indica que seguirá evolucionando y creciendo en utilización por lo que es muy interesante profundizar en su conocimiento y estudio.

Viendo estas realidades tan relevantes se ha decidido hacer un proyecto de una aplicación programada en Android para hacer el registro de rutas en bicicleta o cualquier tipo de vehículo. Para ello se ha trabajado en el conocimiento del lenguaje de programación para hacer aplicaciones en Android y el montaje / preparación de un conjunto de web services en un servidor web Linux para manipular una base de datos remota de los datos de los usuarios. Todo esto ha resultado en una aplicación plenamente funcional donde los usuarios pueden guardar sus rutas y compartirlas para que otros usuarios puedan verlas.



## Resum

Actualment, en la vida diària, els dispositius mòbils o smartphones ja formen una part important de la nostra vida. En els últims anys ha augmentat de forma vertiginosa la utilització de tot tipus d'aplicacions per a ser utilitzades per a millorar la nostra experiència en tot tipus d'activitats l'aire lliure. A dia d'avui està de moda fer esport més que mai: córrer , anar en bicicleta....

Android és un sistema operatiu per a dispositius mòbils molt estes i que gaudeix de molts desenvolupadors i usuaris. Tot indica que seguirà evolucionant i creixent en utilització per lo que és molt interessant profunditzar en el seu coneixement i estudi.

Veient aquestes realitats tan rellevants s'ha decidit fer un projecte sobre una aplicació programada en Android per a fer el enregistrament de rutes en bicicleta o qualsevol altre vehicle. Per a això, s'ha treballat en conèixer el llenguatge de programació per a fer aplicacions en Android i per al muntatge / preparació de un conjunt de web services en un servidor web Linux per a manipular una base de dades remota de les dades dels usuaris. Tot això ha donat com a resultat una aplicació plenament funcional amb la que els usuaris poden guardar les seves rutes i compartir-les per a què els demes usuaris puguin veure-les.



## Abstract

Nowadays, in everyday life, the mobile devices or smartphones are already one of the most important parts of our life. In the last years, the use of different kind of applications have grown spectacularly in order to enhance all our open-air activities. At present it's trendy to do sport more than ever: running, riding bike....

Android is an OS for mobile devices that is very widespread and it's enjoying to have a lot of developers and users. Everything points that the use of this OS will be continuing to evolve and growing. All these cause that it's an interesting thing to be learned and studied.

These relevant realities cause that I decided to do a project about an Android application to register the tracks of the users riding bicycle or another kind of vehicle. To do that I had to learn Android programming language and I had to discover how to mount / ready several web services into a web server in Linux for be ready to work with a remote database of users' data. This has culminated in a fully functional application that enables the users to save their tracks and share them with the others users of the application.



# ÍNDICE

Resumen .....	3
Resum .....	4
Abstract .....	5
<b>1.- Introducción .....</b>	<b>9</b>
1.1 Diferentes soluciones en sistemas operativos .....	10
1.1.1 Symbian .....	10
1.1.2 Windows phone .....	11
1.1.3 BlackBerry.....	12
1.1.4 iOS .....	13
1.1.5 Android .....	14
1.2 Motivaciones personales .....	15
1.3 Aplicaciones en el mercado para seguimiento de rutas .....	16
<b>2.- Objetivos del proyecto .....</b>	<b>17</b>
<b>3.- Especificaciones de biketracking .....</b>	<b>19</b>
<b>3.1 Introducción .....</b>	<b>19</b>
<b>3.2 Requerimientos.....</b>	<b>19</b>
3.2.1 Usuarios.....	19
3.2.1.1 Usuario registrado.....	19
3.2.1.2 Usuario no registrado o invitado.....	19
3.2.2 Servidor web y de base de datos .....	20
3.2.3 Visualización de rutas. ....	20
3.2.4 Datos asociados a cada ruta.....	20
3.2.5 Compartición de rutas.....	20
<b>4.- Diseño .....</b>	<b>22</b>
<b>4.1 Diseño de la base de datos .....</b>	<b>22</b>
4.1.1 Tabla rutas.....	23
4.1.2 Tabla datosgps .....	23
<b>4.2 Diseño de las pantallas de las actividades.....</b>	<b>24</b>
4.2.1 Pantalla inicial .....	24
4.2.2 Pantalla menú principal .....	24
4.2.2.1 Usuario registrado: .....	25
4.2.2.2 Usuario no registrado.....	25
4.2.3 Pantalla de tracking o registro de ruta .....	26
4.2.3 Pantalla de listado de rutas .....	27
4.2.4 Pantalla de menú de la ruta seleccionada .....	28
4.2.4.1 Proveniente de una selección del listado propio. ....	28
4.2.4.2 Proveniente de una selección del listado público .....	29
4.2.5 Pantalla del mapa .....	30
4.2.6 Pantalla de los detalles de la ruta .....	30
4.2.7 Pantalla de cambio de descripción de la ruta .....	31
4.2.8 Pantalla de ajustes .....	32



<b>4.3 Flujo de las actividades de la aplicación</b>	33
<b>4.4 Diseño de los webservices</b>	34
4.4.1 Operaciones de creación	34
4.4.1.1 Creación de una nueva ruta	34
4.4.1.2 Inserción de los datos GPS de una ruta	35
4.4.2 Operaciones Read ( obtención de datos)	35
4.4.2.1 Obtención de las posiciones de una ruta	35
4.4.2.3 Obtención de cálculo de la distancia de una ruta	36
4.4.3 Operaciones Delete ( eliminación )	36
4.4.3.1 Eliminación de una ruta	36
4.4.4 Operaciones Update ( modificación )	37
4.4.4.1 Cambio de la descripción de una ruta	37
4.4.4.2 Cambio de la privacidad de una ruta	37
<b>4.5 Decisiones de diseño</b>	38
4.5.1 La no utilización de una tabla de usuarios.	38
4.5.2 Utilización de json y web y no vía driver jdbc	38
4.5.3 Utilización de tareas asíncronas (ASYNCTASKS) para accesos a la red	39
4.5.4 Elección del algoritmo del cálculo de la distancia entre dos puntos GPS	39
4.5.5 Cálculo de la distancia de la ruta en el servidor y no en el dispositivo móvil	39
4.5.6 La creación de una pantalla para la cambiar un par de parámetros	39
<b>5.- Desarrollo</b>	40
<b>5.1 Instalación y configuración de aplicaciones</b>	40
5.1.1 Instalación de las APIS de google	41
5.1.1.2 API Sign-in Google +	41
5.1.1.3 Google Maps Android API v2	46
<b>5.2 Instalación de nuestro servidor web Raspberry pi rev 2</b>	49
5.2.1 Habilitar SSH	52
5.2.2 Servidor Web – Apache	52
5.2.3 Servidor web – PHP	53
5.2.4 Servidor web - MySQL	54
5.2.5 Servidor web – phpMyAdmin	54
5.2.6 Configuración del router para permitir el acceso desde internet	56
<b>5.3 Implementación</b>	62
5.3.1 Implementación aplicación Android	62
5.3.1.2 Estructura de nuestro proyecto Android	62
5.3.1.3 Fichero androidmanifest.xml	65
5.3.1.4 Clases	67
5.3.1.4.1 GlobalClass.java	67
5.3.1.4.2 Ruta.java	68
5.3.1.4.3 Posicion.java	68
5.3.1.4.4 JSONParser.java	69
5.3.1.5 Actividades	70
5.3.1.5.1 Inicial.java	71
5.3.1.5.2 Menus.java	73
5.3.1.5.3 Settings.java	74
5.3.1.5.4 Tracking.java	75
5.3.1.5.5 Lista.java	79
5.3.1.5.6 MenuRuta.java	80



5.3.1.5.7 DetallesRuta.java .....	81
5.3.1.5.8 CambiaDescripcion.java.....	82
5.3.1.5.9 Mapa.java .....	83
5.3.2 Implementación en servidor web .....	84
5.3.2.1 db_config.php.....	85
5.3.2.2 db_connect.php.....	86
5.3.2.3 borra_ruta.php.....	86
5.3.2.4 calculo_distancia.php.....	87
5.3.2.5 cambia_descripcion.php.....	88
5.3.2.6 cambia_privacidad.php.....	88
5.3.2.7 get_posiciones.php.....	89
5.3.2.8 inserir_posiciones.php.....	89
5.3.2.9 lista_ruta.php.....	90
5.3.2.10 max_rutas.php.....	91
5.3.2.11 nueva_ruta.php.....	92
<b>6.- Evaluación .....</b>	<b>93</b>
<b>7.- Costes.....</b>	<b>94</b>
<b>8.- Planificación.....</b>	<b>95</b>
8.1 Desglose por fases del proyecto.....	95
8.2 Diagrama de Gantt de la planificación.....	96
<b>9.- Conclusiones.....</b>	<b>97</b>
<b>10.- Recursos utilizados.....</b>	<b>98</b>
10.1 Bibliografía.....	98
10.2 Páginas web.....	98
10.3 Software.....	99
10.4 Recursos hardware.....	99





## 1.- Introducción

Hoy en día ya es indiscutible que los Smartphone forman parte de nuestra vida diaria. La mayoría de estos smartphones cuentan con conexión a internet permanente y con un montón de sensores a su disposición: GPS, acelerómetro, barómetro, etc....

Para cualquier actividad que queramos hacer seguramente que ya existen cientos de aplicaciones para ayudarnos o asistirnos en ella; desde hacer la lista de la comprar, comprar entradas de cine y hasta lo que nos interesa en este proyecto: registrar rutas vía GPS y obtener datos sobre ella.

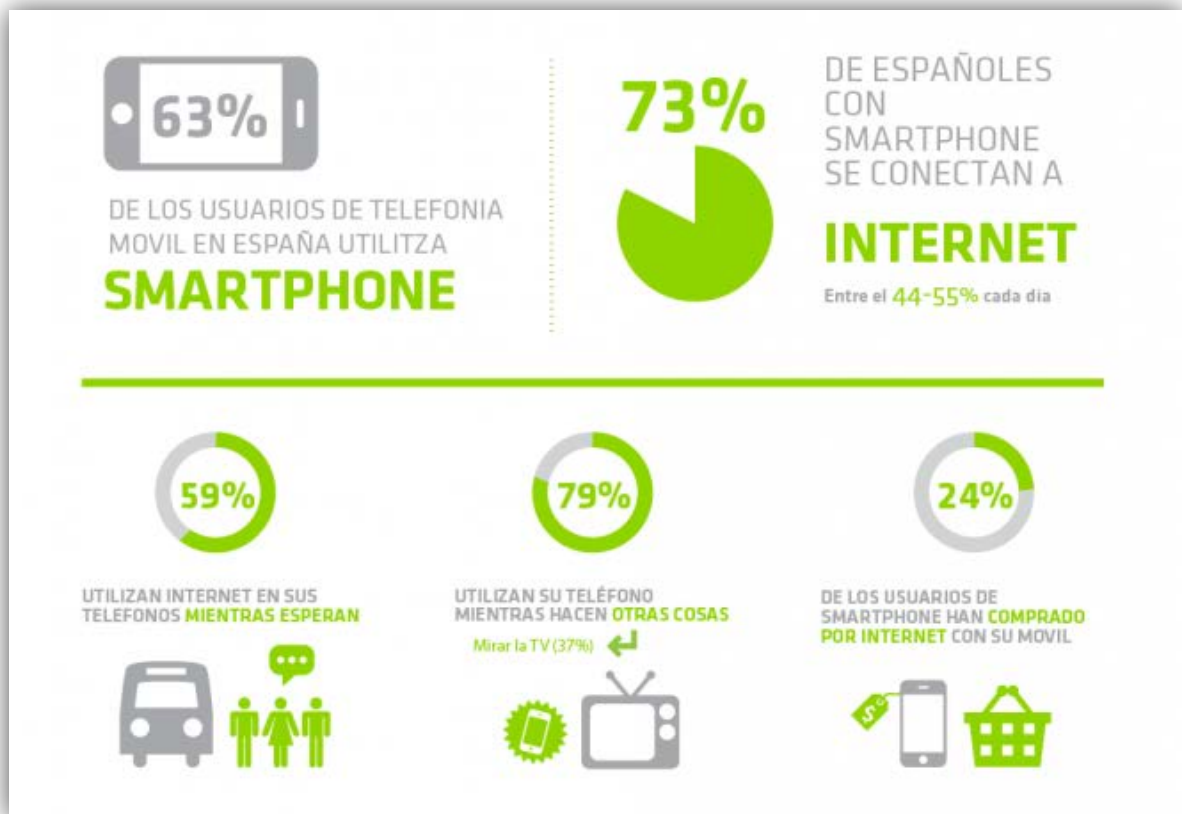


Figura 1. Utilización de Smartphones en España en 2013 [1].

A continuación se expondrán las diferentes tecnologías existentes en el mercado en temas de dispositivos móviles:

[1] <http://www.emfasi.com/blog/cuando-como-y-para-que-usamos-los-smartphones>

## 1.1 Diferentes soluciones en sistemas operativos de los dispositivos móviles

Actualmente tenemos las siguientes soluciones:



Figura 2. SO móviles.

### 1.1.1 Symbian OS

Symbian es un sistema operativo propiedad de Nokia por lo que la mayoría de los móviles con Symbian son de Nokia. En el pasado fue producto de la alianza de varias empresas de telefonía móvil, entre las que se encontraban Nokia, Sony Mobile Communications, Psion, Samsung, Siemens, Arima, Benq, Fujitsu, Lenovo, LG, Motorola, Mitsubishi Electric, Panasonic, Sharp, etc. Sus orígenes provenían de su antepasado EPOC32, utilizado en PDA's y Handhelds de PSION.

Las principales características de este SO son:

- Uso eficiente de todos los recursos por parte de la máquina (especialmente batería, RAM y ROM).



- Está basado en un micro kernel, es decir, una mínima porción del sistema tiene privilegios de kernel, el resto se ejecuta con privilegios de usuario, en modo de servidores.
- Acceso inmediato a los datos.
- Manejo fiable de los datos incluso en caso de fallo en la comunicación o falta de recursos, como memoria, disco o batería.
- Adaptabilidad al hardware específico y a las pilas de telefonía de los fabricantes.
- Consistencia en la comunicación entre los datos propios del dispositivo y otros.

Symbian fue uno de los sistemas operativos más famosos de los últimos tiempos, sobre todo antes de la llegada de iOS y Android, hoy en día ya no dispone de soporte con lo que ya ha quedado sentenciado a muerte.



Figura 3. Logo Symbian

### **1.1.2 Windows Phone**

Windows Phone, anteriormente llamado Windows Mobile es un S.O. móvil compacto desarrollado por Microsoft, se basa en el núcleo del sistema operativo Windows CE y cuenta con un conjunto de aplicaciones básicas, actualmente va por la versión 8.1.

Está diseñado para ser similar a las versiones de escritorio de Windows estéticamente y existe una gran oferta de software de terceros disponible para Windows Mobile, la cual se puede adquirir a través de la tienda en línea Windows Marketplace for Mobile.

Las principales características:

- Sistema en tiempo real con interrupciones anidadas.



- Quantums de tiempo por hilo de ejecución
- 256 niveles de prioridad para los hilos de ejecución

Windows phone está hoy en día en muy buen momento, ganando poco a poco cuota de mercado y ya se ha situado en la tercera posición de SOs utilizados en los smartphones del mercado.



Figura 4. Logo Windows phone,

### **1.1.3 Blackberry OS**

Blackberry OS fue desarrollado por la empresa canadiense RIM ( Research In Motion ) para sus dispositivos. Este SO permite multitarea y tiene soporte para diferentes métodos exclusivos de RIM: trackwheel, trackball, touchpad y pantallas táctiles.

Las primeras versiones de este S.O. se remonta a los primeros dispositivos portables en 1999. Empezaron con el acceso a correo electrónico, navegación web y sincronización con programas como Microsoft Exchange o Lotus Notes aparte de poder hacer las funciones usuales de un teléfono móvil, otras marcas utilizan el cliente de correo electrónico de BlackBerry como Siemens, HTC, Sony Ericsson entre otros, porque cuentan con el teclado QWERTY completo.

Está orientado a un uso corporativo por su gestor de correo electrónico y agenda, con el software BlackBerry Enterprise Server (BES) proporciona el acceso y organización del correo electrónico a grandes empresas identificando a cada usuario con el conocido BlackBerry PIN, la última versión de este sistema operativo a fecha de hoy es la Blackberry OS 7.1.



A día de hoy Blackberry no cuenta con muy buena salud ya que tenía el tercer lugar en su utilización como SO y ha pasado al cuarto puesto en 2013 con una bajada importante del 40 % en crecimiento respecto al 2012, quedándose un escueto 1.7 % de cuota de mercado.



Figura 5. Logotipo de blackberry.

#### **1.1.4 iOS**

iOS, anteriormente denominado iPhone OS fue creado por Apple originalmente para el iPhone, siendo después usado en el iPod Touch e iPad. Deriva del Mac OS X y fue lanzado al mercado el año 2007, aumento su interés al tenerlo también el iPod Touch e iPad, dispositivos con las capacidades multimedia del iPhone pero sin la capacidad de hacer llamadas telefónicas, en si su principal revolución fue una combinación ideal entre hardware y software, y un manejo muy bueno de la pantalla multitáctil.

Una de las características que lo hacen muy interesante entre los mejores son el manejo de correo electrónico, la navegación web con el navegador Safari, la reproducción multimedia, llamadas con videoconferencia y sobre todo una estética de hardware muy refinada y agradable que ha caracterizado a los productos de Apple desde siempre.

La última versión del sistema operativo es el iOS 7.1.

iOS cuenta con una posición muy buena en el panorama de los dispositivos móviles. Se sitúa en el segundo lugar, después de Android con una cuota del mercado a finales de 2013 del 13 % y con una ganancia respecto al 2012 de un 25.6 %. [2]

[2] <http://www.idc.com/getdoc.jsp?containerId=prUS24442013>



Figura 6. Logo de Apple

### ***1.1.5 Android***

Llegamos al SO que más nos interesa ya que es en el que hemos hecho la aplicación.

El sistema operativo Android es sin duda el líder del mercado móvil en S.O, está basado en Linux y diseñado originalmente para dispositivos móviles Smartphone pero después se adaptó para ser usado en tabletas como es el caso del Galaxy Tab de Samsung.

Actualmente se encuentra en desarrollo para usarse en netbooks y PCs.

El desarrollador de este S.O. es Google, fue presentado en el año 2007 y liberado en el año 2008. Paralelamente se creó la Open Handset Alliance, conjunto compuesto por 78 compañías de hardware, software y telecomunicaciones dedicadas al desarrollo de estándares abiertos para dispositivos móviles; esto ha ayudado mucho a Google a expandir el S.O. Muchas empresas utilizan este SO, tales como: Samsung, HTC ,LG y Motorola entre muchos otros más .

Las aplicaciones se ejecutan en un “framework Java de aplicaciones” orientadas a objetos en la que se ejecuta en una máquina virtual Dalvik que es parecida a la máquina virtual de Java propuesta por James Gosling (creador del lenguaje java) con algunas modificaciones

Dispone de correo, navegación, reproducción multimedia, multitarea y el soporte de pantallas multitáctil muy parecido al iPhone de Apple.



Con una cuota del mercado del 81 % a finales de 2013 se erige como el indiscutible líder en SO del panorama Smartphone. Creció en un 51 % respecto al 2012 por lo que goza de una gran expansión.



Figura 7. Logo de Android

## 1.2 Motivaciones personales

Hoy en día hay multitud de aplicaciones para el seguimiento y registro de rutas, el deporte está de moda más que nunca, empezando con el running hasta pasando por el ciclismo, el principal objetivo de esta aplicación. Por este motivo el mercado de nuestra aplicación es muy amplio ya que abarca a muchísimos usuarios potenciales para nuestro proyecto.

La idea surgió un día saliendo a hacer una vuelta en bicicleta y llegar a un sitio con unas vistas impresionantes desde arriba de la Mussara. Más tarde se lo expliqué a un amigo que también sale mucho en bicicleta y no supe describir cómo llegar a ese mirador tan bueno. De aquí surgió la idea de crear una aplicación para poder trackear las rutas en bicicletas y poder compartirla con la gente. El hecho es que después de aquella vez poco he salido ya en bicicleta pero se ha valorado que no importa mucho en qué tipo de vehículo se utilice para desplazarse y utilizar una aplicación de este tipo: se puede utilizar en multitud de casos, a pie o corriendo.

En nuestro proyecto nos basamos en ser una aplicación **gratuita y muy eficaz**, que no nos sature con mucha información como hace la mayoría de aplicaciones de tracking.



### 1.3 Aplicaciones en el mercado para seguimiento de rutas

A continuación daremos un repaso por las diferentes aplicaciones que existen en el mercado para ver que ofrecen y así poder analizar que nos ofrece la competencia

Estas son las 6 aplicaciones más relevantes a día de hoy:

- **MapMyRide:** Considerada una de las aplicaciones para bicicletas más completas que existen para smartphones, MapMyride permite al usuario medir la distancia, tiempo, velocidad y calorías quemadas, en poco pasos. Otras de las funciones de MapMyRide es la de "Seguimiento de dieta, comida y nutrición" y la sincronización con tu ritmo cardíaco.
- **BikeMap.net:** Su nombre lo dice todo: Mapa de Bicicletas.. Tiene más de 850.000 rutas en todo el mundo, formando una comunidad con otros ciclistas y entregándole al usuario la ruta más cercana respecto a su ubicación.
- **BikeBrain:** A diferencia de las apps anteriores, BikeBrain no cuenta con la función de rutas, sino más bien la mostrarte a ti mismo en el mapa. En general, el GPS sirve para compartir fotos en redes sociales de forma geo-referenciada; es decir, que tus amigos en Twitter y Facebook sepan por dónde vas. Solo disponible para productos Apple.
- **Strava:** Esta aplicación permite descubrir nuevas rutas en todo el mundo y medir la velocidad, distancia y calorías quemadas en cada sesión. Todo esto, comparable con sesiones deportivas pasadas y ver tu progreso. Tiene un modo que sirve para hacer competencias con uno mismo y también tiene un ranking con los mejores tiempos para una ruta en concreto.
- **B-iCycle:** ofrece al ciclista valores para la velocidad actual, velocidad promedio, máxima velocidad, distancia total, altitud, calorías quemadas y tu posición en el mapa. Otra de las características de B.iCycle es su comunidad; más de 500.000 mapas han sido compartidos por usuarios alrededor del mundo, entregando la información actualizada.
- **Endomondo:** es toda una comunidad deportiva para los usuarios de smartphone, permitiendo registrar la velocidad, distancia y calorías en cualquier deporte al aire libre.

Ya hemos hecho un repaso a las aplicaciones más importantes del sector de aplicaciones para Smartphone de rutas. Vemos que hay una competencia muy importante y con





aplicaciones muy potentes. La nuestra es más simple y no por eso desmerece de las demás, muchos usuarios seguro que agradecen el concepto más simple y práctico de nuestra aplicación y seguro que contaremos con un buen número de usuarios ya que la hacemos en el SO que cuenta con más terminales: Android.

Más adelante se valoraría el hacer otra aplicación similar para otra plataforma de SO reaprovechando la parte del servidor web y así poder llegar a muchos más usuarios.



## 2.- Objetivos del proyecto

La finalidad y motivación de este proyecto es la creación de una aplicación Android que permita registrar rutas hechas por cualquier usuario utilizando el GPS de su Smartphone y poder elegir si compartir esta ruta con todos los usuarios de la aplicación. También es motivador el hecho de aprender a programar una aplicación en el SO para móviles predominante.

Se quiere con este proyecto:

- Una aplicación Android sencilla de utilizar.
- Utilización de recursos eficientemente y trabajar con datos de rutas que nos interesan.
- Compartición de rutas.
- Visualización simple a través de google maps en el dispositivo móvil.
- Creación de webservices que interaccionen con la base de datos que nos servirán y se podrán reaprovechar en caso de querer ampliar las plataformas soportadas como iOS, etc...
- Buscando la eficiencia: montaje de un servidor web en un raspberry PI rev 2.

Durante el transcurso del proyecto apareció la posibilidad de utilizar un raspberry Pi como servidor web por lo que se configuró para su uso como tal. Como **objetivo secundario** ha aparecido la preparación y utilización de un servidor web barato, sencillo y eficiente ( sólo gasta 3.5 w ).



### 3.- Especificaciones de biketracking

#### 3.1 Introducción

Esta aplicación se hará para Android y se encargara de guardar las rutas hechas en bicicleta ( o en cualquier otro vehículo) por el usuario, utilizando el GPS incorporado en el Smartphone del cliente. Estas rutas ser guardaran en un servidor externo ( teniendo en cuenta la posible conectividad deficiente en según qué lugares remotos se pondrá énfasis en evitar pérdidas de datos por este tema ).

Posteriormente estas rutas guardadas podrán ser visualizadas gráficamente y se podrán acceder a estadísticas sobre la misma. Los usuarios podrán compartir las rutas que quieran para que cualquier usuario pueda consultarlas pero no modificarlas ni eliminarlas.

#### 3.2 Requerimientos

A continuación se procede a detallar cada uno de los requerimientos:

##### 3.2.1 Usuarios

Se distinguirán **dos** tipos de usuario: **Registrado / NO registrado ( Invitado )**

##### 3.2.1.1 Usuario registrado

Este tipo de usuario se identificara a través de un API externo y todas sus rutas estarán asociadas a esta identificación. El usuario podrá en todo momento decidir si es una ruta suya es privada o pública ( para que puedan verla todos los usuarios ).

El usuario registrado podrá en todo momento eliminar cualquiera de sus rutas y tendrá acceso a cambiar la descripción de la misma y que sea de tipo pública o privada .

##### 3.2.1.2 Usuario no registrado o invitado



A este tipo de usuario no se le pedirá ningún tipo de autenticación para entrar en la aplicación. Las rutas que haga este tipo de usuario se guardaran como públicas y todos los usuarios podrán acceder a ellas. También podrá visualizar todas aquellas rutas que sean públicas.

### ***3.2.2 Servidor web y de base de datos ( para guardar las rutas, datos gps, etc..)***

Cuando el usuario haya acabado la ruta, la aplicación procederá a guardar la misma en una base de datos remota. Primeramente se comprobara si hay conectividad para enviarla al servidor remoto y en caso contrario se tendrá que avisar al usuario de que hay problemas de conectividad y se le dará opción a establecer la comunicación más adelante.

Queremos una solución de servidor web **barato y eficiente.**

En la parte de implementación se comentara exactamente como se ha llevado a cabo estas especificaciones.

### ***3.2.3 Visualización de rutas.***

La visualización de rutas se hará visualmente a través de gráficos de un mapa, en él se dibujara el recorrido de la misma.

### ***3.2.4 Datos asociados a cada ruta***

De cada ruta se podrá consultar varios valores y estadísticas:

- Tiempo total
- Velocidad máxima
- Altitud ganada
- Velocidad media
- Kms recorridos totales.



### ***3.2.5 Compartición de rutas***

La aplicación tendrá la posibilidad de compartir las rutas, es decir: hacerlas públicas. Todos los usuarios podrán ver las rutas públicas. El usuario no registrado si realiza alguna ruta será del tipo pública, y si es registrado podrá elegir si comparte o no la ruta de la que es propietario, esta opción siempre será reversible.



## 4.- Diseño

En este apartado de la documentación entraremos en detalle sobre:

- Diseño base de datos
- Diseño de las diferentes pantallas de las actividades
- Flujo entre las diferentes actividades ( interfaz gráfica )
- Diseño de los webservices
- Decisiones de diseño

### 4.1 Diseño de la base de datos

La base de datos para la aplicación consta de dos tablas, en un primer diseño se proyectó el uso de tres tablas pero ha sido posible eliminar una de ellas por no ser necesaria al utilizar la API de Google+: la tabla de usuarios. Por lo que nos queda el diseño final siguiente:

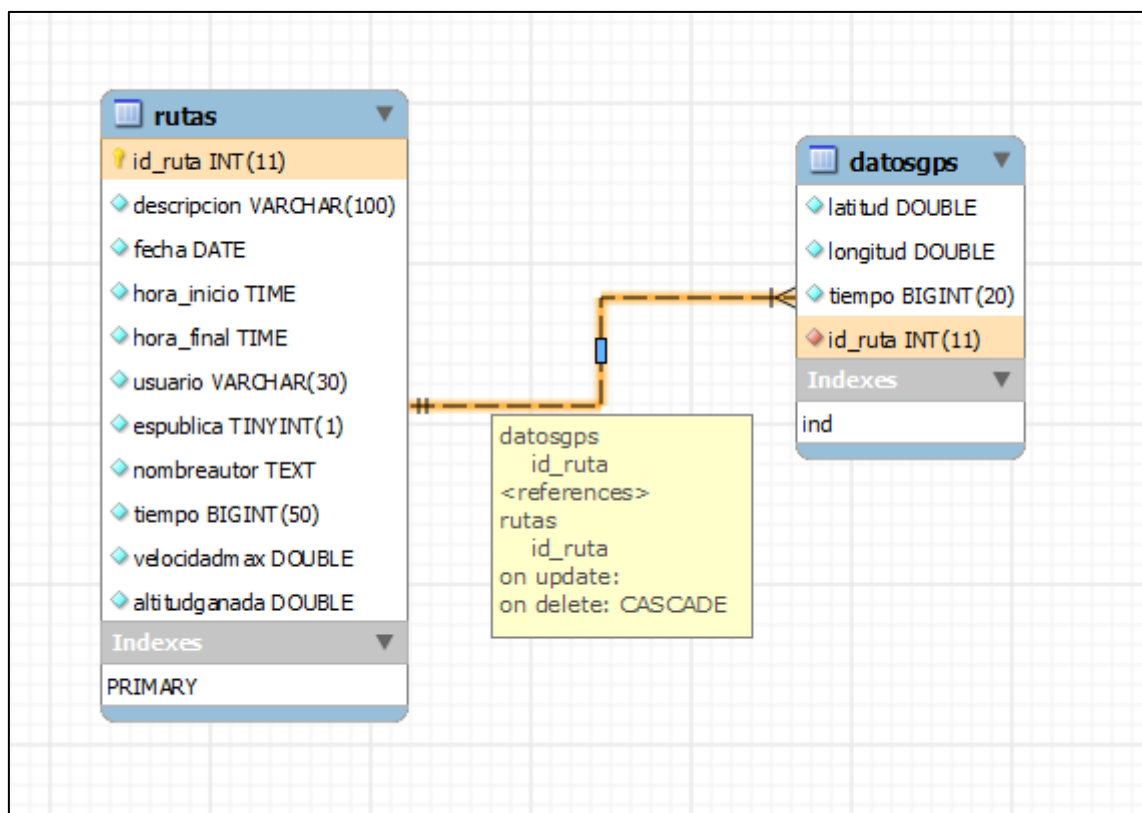


Figura 8. Esquema de la base de datos



#### 4.1.1 Tabla rutas

Esta tabla se encarga de almacenar todos los datos relacionados con la ruta, teniendo como clave primaria un número que se asigna cuando se crea.

```
CREATE TABLE `rutas`  
(  
  `id_ruta` int(11) NOT NULL AUTO_INCREMENT,  
  `descripcion` varchar(100) NOT NULL,  
  `fecha` date NOT NULL,  
  `hora_inicio` time NOT NULL,  
  `hora_final` time NOT NULL,  
  `usuario` varchar(30) NOT NULL,  
  `espublica` tinyint(1) NOT NULL,  
  `nombreautor` text NOT NULL,  
  `tiempo` bigint(50) NOT NULL,  
  `velocidadmax` double NOT NULL,  
  `altitudganada` double NOT NULL,  
  PRIMARY KEY (`id_ruta`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Habría que comentar que se tuvo un problema para manejar el id de usuario (campo usuario de la tabla) que nos proporciona la API de google+. Se intentó utilizar el campo como bigint pero nos daba error por tener este 21 dígitos y superar el valor máximo establecido en mysql de bigint : 9223372036854775807 y 19 dígitos, por lo que se tuvo que solucionar guardándolo como un campo de String de caracteres.

#### 4.1.2 Tabla datosgps

En esta tabla guardamos cada uno de las posiciones GPS de la ruta y se guardara el tiempo de cada una de ellas.

```
CREATE TABLE `datosgps`  
(  
  `latitud` double NOT NULL,  
  `longitud` double NOT NULL,  
  `tiempo` bigint(20) NOT NULL,  
  `id_ruta` int(11) NOT NULL,  
  KEY `ind` (`id_ruta`)  
) ENGINE=InnoDB DEFAULT CHARSET=latin1;  
  
ADD CONSTRAINT `datosgps` FOREIGN KEY (`id_ruta`) REFERENCES `rutas`  
(`id_ruta`) ON DELETE CASCADE;
```

Importante este último **constraint** ya que nos permitirá que todos los datos GPS asociados a una ruta puedan ser eliminados cuando la ruta sea eliminada automáticamente.



## 4.2 Diseño de las pantallas de las actividades

A continuación se mostrará cómo deben de ser las diferentes interfaces gráficas de las actividades.

### 4.2.1 Pantalla inicial

En esta pantalla es donde el usuario inicia el registro o en caso contrario entrar cómo usuario no registrado.

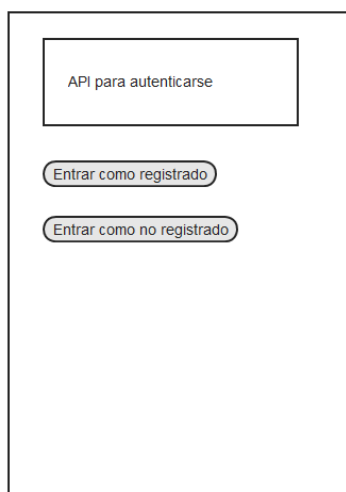


Figura 9. Diseño pantalla inicial.

Habrán dos botones a parte de los que pueda tener la API:

- 1.- Entrar como **registrado**.
- 2.- Entrar como **no registrado**.

### 4.2.2 Pantalla menú principal

Tanto si entramos como usuario registrado o no, llegaremos a esta pantalla. Mostraremos las diferentes opciones disponibles dependiendo de qué tipo de usuarios seamos en ese momento.

#### 4.2.2.1 Usuario registrado:



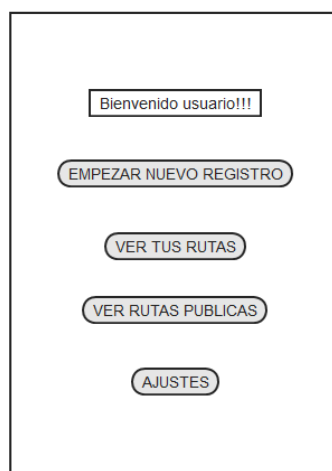


Figura 10. Diseño de la pantalla del menú principal para el usuario registrado

El usuario registrado tendría acceso a ver sus rutas privadas, la publicas, iniciar un nuevo registro de ruta y poder cambiar los ajustes de la aplicación.

#### 4.2.2.2 Usuario no registrado

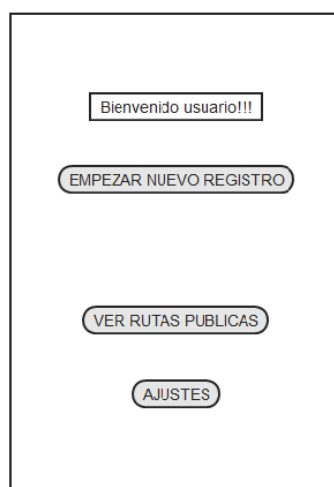


Figura 11. Diseño pantalla menú principal para el usuario no registrado

En esta pantalla lo único que cambia respecto a la del usuario registrado es el botón de acceso a las rutas privadas de las que carece el usuario no registrado.



#### 4.2.3 Pantalla de tracking o registro de ruta

Antes de acceder a esta pantalla se comprobará si hay disponibilidad del GPS y en caso no tenerlo activado nos saltaría el siguiente mensaje:

A rectangular dialog box with a thin black border. Inside, there are three centered rectangular buttons stacked vertically. The top button contains the text "ATENCION". The middle button contains the text "NO TIENES EL GPS ACTIVADO". The bottom button contains the text "CUANDO LO ACTIVES TENDRAS ACCESO".

Figura 12. Diseño del dialogo del aviso de GPS deshabilitado.

En el caso que el GPS este en servicio entraremos en el siguiente diseño de pantalla:

A rectangular screen layout with a thin black border. At the top, there are two rectangular buttons stacked vertically: "VELOCIDAD" and "TIEMPO". Below these is a rectangular button labeled "Numero de registros". Underneath that is a larger rectangular container with a thin black border, which contains three rows of data: "LATITUD" with an input field, "LONGITUD" with an input field, and "ALTURA" with an input field. At the bottom center of the screen is an oval button labeled "STOP".

Figura 13. Diseño de la pantalla de registro de la ruta

Esta pantalla constará de lo siguiente:

- Indicación de la velocidad en ese momento
- Tiempo que llevamos de registro de ruta
- Número de registros que llevamos hechos



- Latitud de la posición en ese momento
- Longitud de la posición en ese momento
- Botón de acabar el registro de la ruta ( STOP )

En caso de no tener conexión a internet no podríamos enviar los datos y deberíamos tener un mensaje de aviso como este:

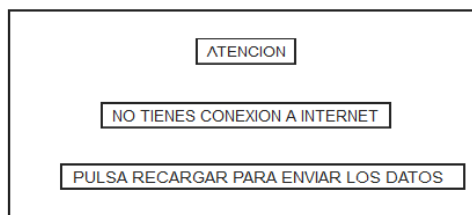


Figura 14. Dialogo de error al no tener conectividad

Después de este dialogo de aviso debería cambiar el botón de STOP de la pantalla de registro de ruta a RECARGAR para que el usuario puede volver a probar de enviar los datos registrados.

#### 4.2.3 Pantalla de listado de rutas

El diseño de la pantalla en la que se muestran las rutas ( tanto privadas como públicas ) es el siguiente:

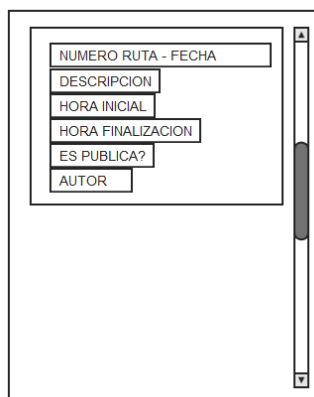


Figura 15. Diseño de la pantalla del listado de rutas



Aparecerán listadas todas las rutas y de cada una habrá los datos que se han mostrado. Se podrá clicar sobre la ruta y entonces se abrirá la siguiente pantalla de menú de ruta.

Si no hubiese ninguna ruta no saldría esta pantalla, se comunicaría en texto que no hay ninguna ruta aún y se volvería a la pantalla de menú principal.

#### ***4.2.4 Pantalla de menú de la ruta seleccionada***

En esta pantalla habrá también que diferenciar de dónde venimos. En caso de haber seleccionado ver las rutas propias del usuario o de venir de haber seleccionado una ruta publica del listado público. Tendremos que diferenciar dos casos:

##### ***4.2.4.1 Proveniente de una selección del listado propio.***

Esta pantalla se mostrara de esta manera:

El diagrama muestra una interfaz de usuario con los siguientes elementos:

- Un campo de entrada etiquetado "Ruta numero" con un cuadro de texto adyacente.
- Un campo de entrada etiquetado "Autor" con un cuadro de texto adyacente.
- Un botón redondeado etiquetado "Ver mapa".
- Un botón redondeado etiquetado "Detalles sobre la ruta".
- Un botón redondeado etiquetado "Eliminar ruta".
- Un botón redondeado etiquetado "Cambiar descripción".
- Un botón redondeado etiquetado "Cambiar privacidad".

Figura 16. Diseño del menú de la ruta seleccionada.

Se tendrán los siguientes botones:

- Ver la ruta representada en el mapa de google maps
- Detalles y datos de la ruta en cuestión.
- Eliminar la ruta
- Cambiar el texto descriptivo de la ruta.



- Cambiar la privacidad de la ruta, siempre saldrá la opción opuesta al valor de la privacidad que tenga la ruta. ( Cambiar a privada / Cambiar a publica ).

En el caso de la selección de eliminar ruta nos tendrá que aparecer un mensaje de aviso como el siguiente:

Figura 17. Diseño de la pantalla de dialogo aviso de eliminación de ruta

#### 4.2.4.2 Proveniente de una selección del listado público.

Sería la siguiente:

Figura 18. Diseño del menú de la ruta seleccionada pública

Se tendrán los siguientes botones:



- Ver la ruta representada en el mapa de google maps
- Detalles y datos de la ruta en cuestión.

Como se muestra no se podrá eliminar, ni cambiar privacidad o descripción de la ruta

#### 4.2.5 Pantalla del mapa

Esta pantalla tendrá el mapa que nos proporcionara la API de google Maps, donde aparecerá la ruta gráficamente representada con su marcador de inicio y fin.

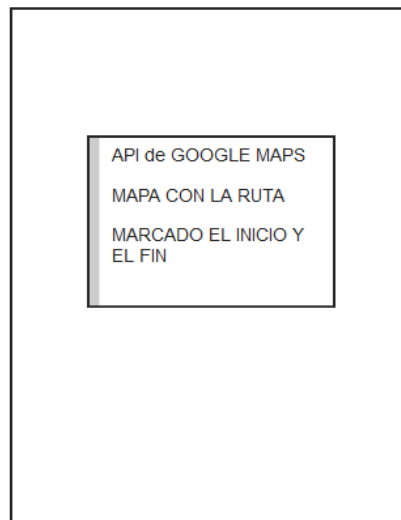


Figura 19. Diseño de la pantalla del mapa de la ruta

#### 4.2.6 Pantalla de los detalles de la ruta

El diseño seria de esta manera:

DATOS DE LA RUTA	
DISTANCIA	<input type="text"/>
VELOCIDAD MEDIA	<input type="text"/>
TIEMPO TOTAL	<input type="text"/>
PUBLICA ?	<input type="text"/>
ALTITUD GANADA	<input type="text"/>



Figura 20. Diseño de la pantalla de los detalles de la ruta

Constará de los siguientes campos de texto a mostrar:

- Distancia total de la ruta.
- Velocidad media de la ruta.
- Tiempo total de la duración de la ruta.
- Si es pública o privada.
- La altitud acumulada durante la ruta

#### ***4.2.7 Pantalla de cambio de descripción de la ruta***

DESCRIPCION ACTUAL

CAMPO DE TEXTO  
PARA INTRODUCIR  
LA NUEVA DESCRIPCION

GUARDAR CAMBIOS CANCELAR

Figura 21. Diseño de la pantalla de cambio de descripción

Tendremos los botones de guardar cambios o cancelar.



#### 4.2.8 Pantalla de ajustes

La pantalla de ajustes tendrá la siguiente configuración:

DIRECCION SERVIDOR ACTUAL

CAMPO PARA INTRODUCIR TEXTO

CAMBIAR

TIEMPO ADQUISICION DATOS GPS

CAMPO PARA INTRODUCIR TEXTO

CAMBIAR

Figura 22. Diseño de la pantalla de cambio de ajustes

En esta pantalla nos aparecerán dos campos de introducción de texto para cambiar tanto la IP del servidor como el tiempo de adquisición de datos por parte del GPS.





### 4.3 Flujo de las actividades de la aplicación

Habr  en total **9** actividades:

1. Actividad de login
2. Actividad del men  general
3. Actividad del registro de la ruta
4. Actividad del listado de rutas
5. Actividad de los ajustes
6. Actividad del men  de la ruta
7. Actividad para cambiar la descripci n de la ruta
8. Actividad para visualizar los datos de la ruta
9. Actividad que contiene el mapa que nos proporciona google maps.

En este esquema se representa el flujo entre actividades de las aplicaciones:

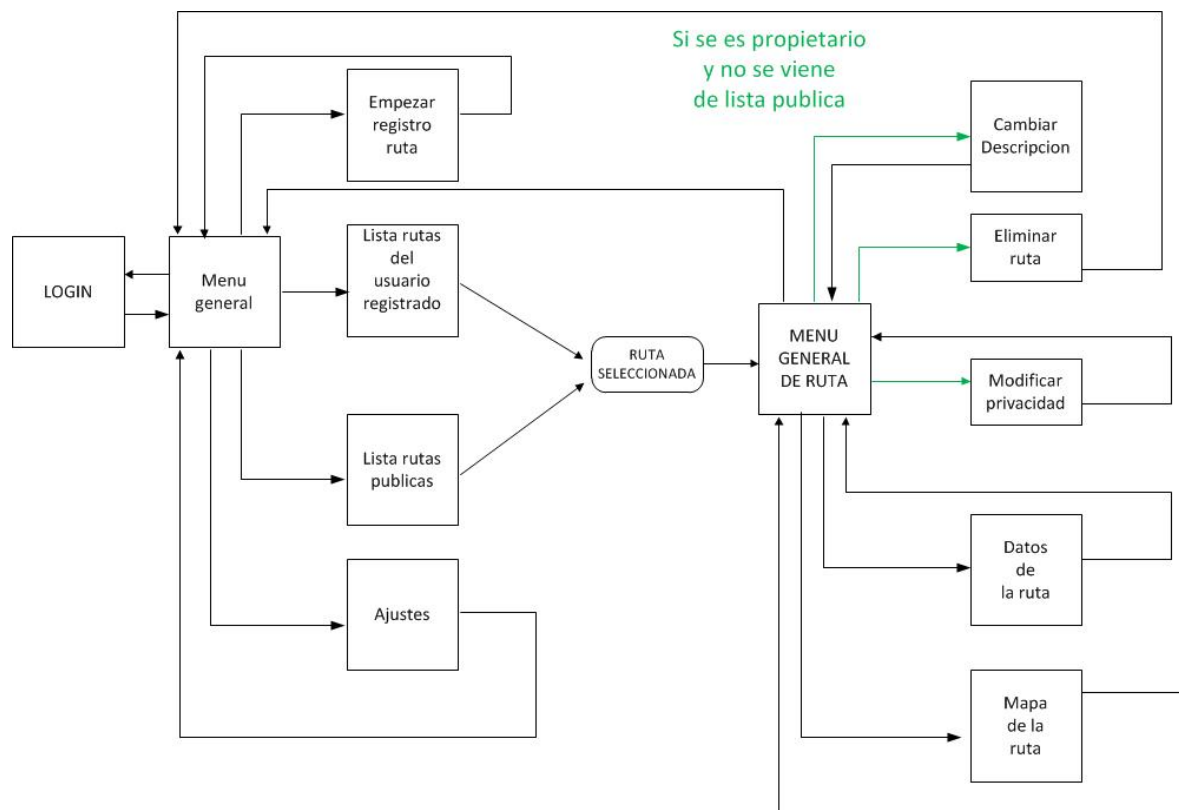


Figura 23. Diagrama de flujo entre las actividades ( pantallas ) de la aplicaci n



## 4.4 Diseño de los webservices

En nuestro servidor web dispondremos de varios servicios para gestionar los datos (en nuestro caso lo haremos con php + mysql ) a través del paso de parámetros . A continuación se detallaran los parámetros y las salidas tipo resultado para cada una de estas páginas.

Seguiremos un esquema básico basado en lo que ingles de denomina **CRUD**. [3] basándonos en operaciones como indica el acrónimo:

- **Create**: creación de rutas y posiciones de datos GPS.
- **Read**: leer de la base de datos información sobre las rutas.
- **Update**: actualización de datos sobre una ruta.
- **Delete**: eliminación de una ruta y con ello los registros GPS asociados a ella.

A continuación se irán comentado los diferentes diagramas del diseño de webservices que debería tener en nuestro servidor web.

### 4.4.1 Operaciones de creación

Tenemos 2:

#### 4.4.1.1 Creación de una nueva ruta

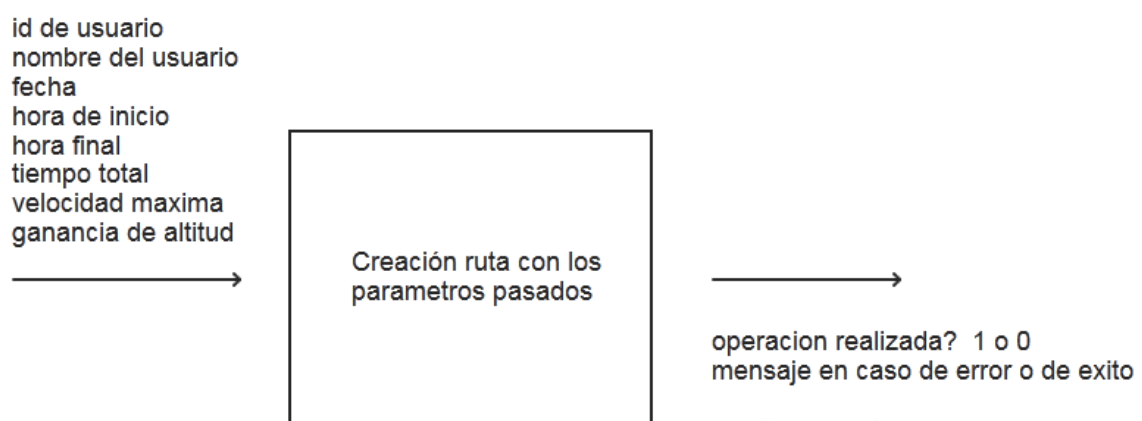


Figura 24. Diagrama entradas y salidas de la creación de una ruta

[3] [http://en.wikipedia.org/wiki/Create,\\_read,\\_update\\_and\\_delete](http://en.wikipedia.org/wiki/Create,_read,_update_and_delete)



#### 4.4.1.2 Inserción de los datos GPS de una ruta

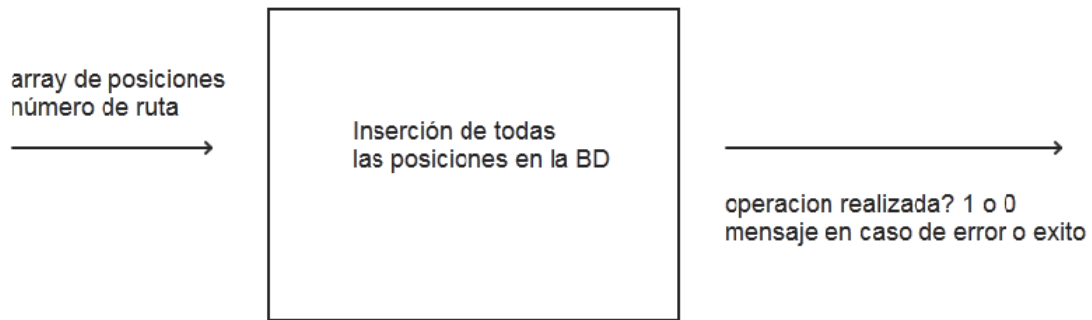


Figura 25. Diagrama entradas y salidas de la inserción de las posiciones

#### 4.4.2 Operaciones Read ( obtención de datos)

Hay tres diferentes:

##### 4.4.2.1 Obtención de las posiciones de una ruta

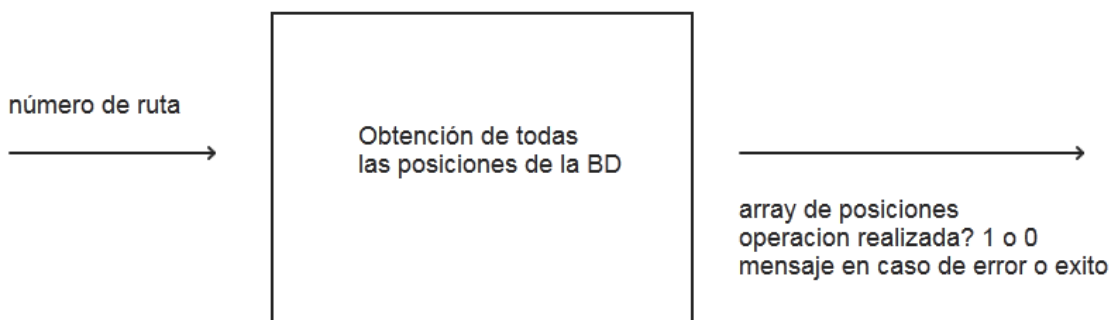


Figura 26. Diagrama entradas y salidas del listado de las posiciones

##### 4.4.2.2 Obtención del listado de rutas de un usuario o de las públicas

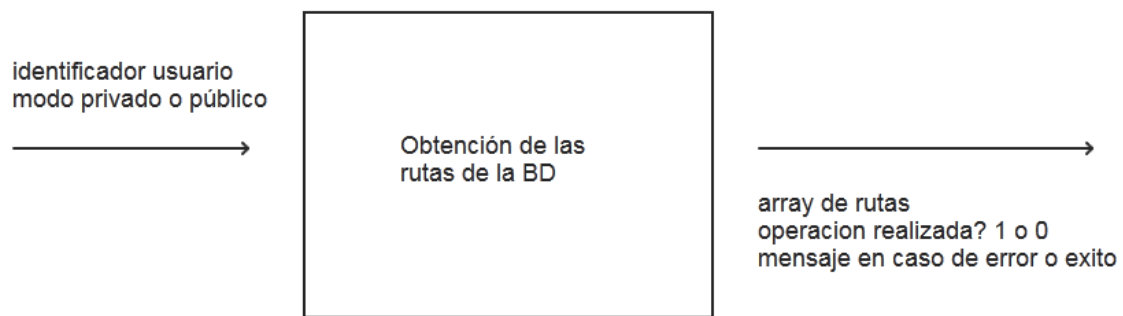


Figura 27. Diagrama entradas y salidas del listado de las rutas

#### 4.4.2.3 Obtención de cálculo de la distancia de una ruta

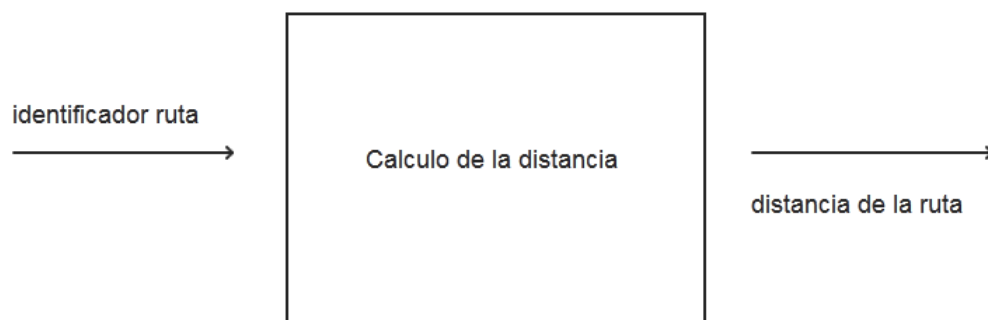


Figura 28 Diagrama entradas y salidas del cálculo de distancia

#### 4.4.3 Operaciones Delete ( eliminación )

Solo tendremos una: la de borrar una ruta.

##### 4.4.3.1 Eliminación de una ruta

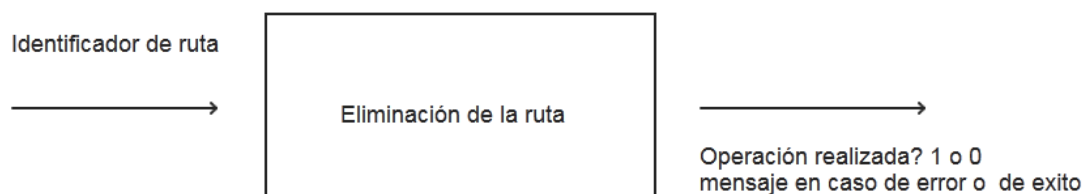


Figura 29. Diagrama entradas y salidas de la eliminación de una ruta



#### 4.4.4 Operaciones Update ( modificación )

Tenemos 2 en total:

##### 4.4.4.1 Cambio de la descripción de una ruta

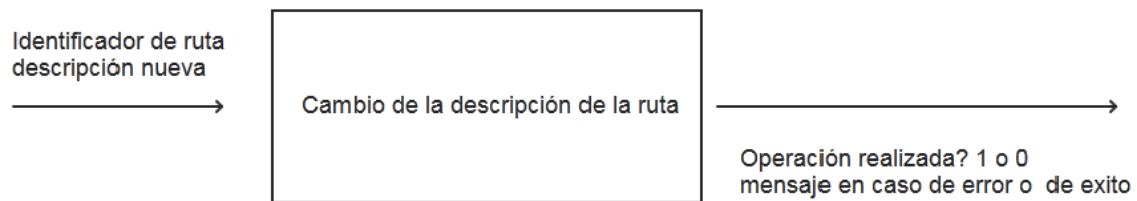


Figura 30. Diagrama entradas y salidas del cambio de la descripción de una ruta

##### 4.4.4.2 Cambio de la privacidad de una ruta

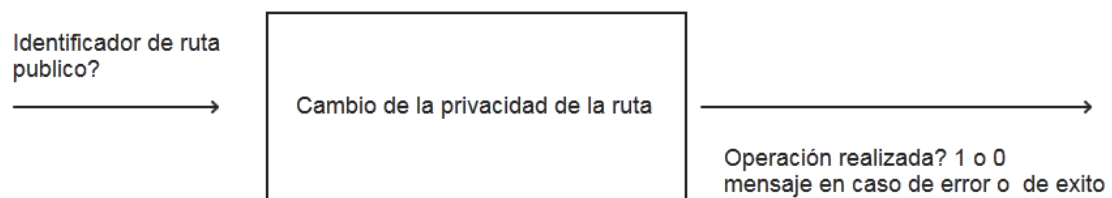


Figura 31. Diagrama entradas y salidas del cambio de la privacidad de una ruta



## 4.5 Decisiones de diseño

### 4.5.1 La no utilización de una tabla de usuarios.

La API de google plus nos ofrece la identificación de sus usuarios con su sistema de autenticación, con lo que podemos obtener un identificador único para cada usuario que es un número de **21 dígitos**. ( OEj : 108250409176348733273, mi id de usuario). Con todo esto nos ofrece el ahorro:

- Google nos ahorra la tarea de tener de autenticar a los usuarios
- De una tabla menos y el consiguiente ahorro de gestión y espacio en la base de datos.

### 4.5.2 Utilización de json y web services vía paginas php y no utilizar la conexión directa vía driver jdbc [4] a la base de datos mysql.

En una primera etapa se implementó la solución de conexión a la base de datos a través de una conexión directa por jdbc ( driver java para mysql ) pero más adelante se valoró que esta forma de conexión tiene un gran fallo de seguridad, cualquier persona que acceda a los datos internos de la aplicación podrá descifrar los datos y credenciales de la base de datos y así permitir un acceso total a ella.

Leyendo e informándome por internet averigüé que siempre se recomienda tener una capa de seguridad vía webservices ( php + mysql ) y así evitar problemas de obtención de datos potencialmente peligrosos ( contraseñas y acceso libre a nuestra base de datos ).

Por lo que se decidió montar las comunicaciones de la aplicación vía webservices. Esto no aporta una gran ventaja a la de la conexión directa a la base de datos: podemos hacer que nuestro webservices sea accedido por diferentes plataformas ya que la implementación de la manipulación de la base de datos ya nos sirve para cualquier conexión independientemente del origen de esta.

[4] <https://dev.mysql.com/doc/connector-j/en/connector-j-overview.html>



#### ***4.5.3 Utilización de tareas asíncronas (ASYNCTASKS) para accesos a la red por parte de la aplicación***

Se ha comprobado durante la realización los problemas que presentan los accesos a la red en el hilo principal de ejecución ( cuelgues, retrasos, cierres forzados ) por lo que en todas las peticiones a red de la aplicación se harán a través de este tipo de tareas.

#### ***4.5.4 Elección del algoritmo del cálculo de la distancia entre dos puntos GPS.***

Para realizar el cálculo de la distancia entre dos puntos GPS existen dos algoritmos: **Haversine y el Vincenty**. Para el proyecto se ha elegido el **Vincenty**, ya que ofrece más precisión a costa de requerir más cálculo computacional. Este gasto computacional será asumido por el servidor.

#### ***4.5.5 Cálculo de la distancia de la ruta se realiza en el servidor y no en el dispositivo móvil***

Se decide calcular la distancia total del recorrido de la ruta en el servidor para así poder quitar carga de cálculo al dispositivo con lo que ayudamos a no que gaste tanta batería y tener un poco más de autonomía. El algoritmo de cálculo de la distancia entre dos puntos GPS es un poco complejo.

#### ***4.5.6 La creación de una pantalla para la cambiar un par de parámetros importantes de la aplicación.***

Se ha estimado interesante la creación una pantalla para poder introducir y poder cambiar dos parámetros clave en la aplicación:

- **Tiempo entre cada registro de GPS:** el usuario podrá cambiar este parámetro para ofrecerle la oportunidad de mejorar la precisión de seguimiento de la ruta o en caso contrario si así lo desea. Esta precisión de seguimiento redundante en una mayor precisión en el cálculo de distancias ya que tenemos un muestreo mucho mejor en caso de hacer el registro de posiciones GPS más breve.
- **IP del servidor:** En caso de que el servidor cambie de IP este no será problema si el usuario no tiene actualizada la aplicación ya que podrá ingresar este valor en la pantalla de modificación.



## 5.- Desarrollo

Para nuestro proyecto necesitaremos instalar y configurar diversas aplicaciones. Iremos paso a paso la instalación de todas las herramientas utilizadas para este proyecto.

### 5.1 Instalación y configuración de aplicaciones.

Por orden este es el conjunto de aplicaciones y repositorios que debemos de descargar e instalar antes de empezar:

- **Java SDK.** Por si no lo tuviésemos previamente deberíamos de instalarlo. Es el conjunto de herramientas de desarrollo de aplicaciones Java. Lo podemos descargar de <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- **Android SDK:** Nos hará falta la plataforma que contiene todos los paquetes necesarios para desarrollar aplicaciones para dispositivos Android. Lo podemos descargar de : <http://developer.android.com/sdk/index.html?hl=sk>
- **Eclipse IDE.** Este es el programa base que nos facilitara la programación ya que incluye el editor, compilador, etc... para empezar a programar en Android. Como ya indica el nombre, es el entorno de desarrollo integrado para poder programar. Lo podremos descargar desde la página de <http://www.eclipse.org/downloads/>
- **ADT Plugin:** Incorpora las utilidades del *Android SDK* al Eclipse, como son: el ADB, DDMS, editor de la interfaz gráfica, compilador para Android SDK, plantillas de las clases y proyectos Android.

Con todo esto ya instalado ya podremos empezar a programar nuestra aplicación.





### 5.1.1 Instalación de las APIS de google

A continuación se mostrara como instalar las APIS de Google que nos harán falta en nuestra aplicación Android:

#### 5.1.1.2 API Sign-in Google +

Google plus es una parte de los *Google Play Services APIs*, por lo que lo primero que tenemos que hacer es descargar los *google play services* en el Android SDK manager. Tendremos que abrir SDK manager e instalar los *play services* que encontraremos en la sección de **Extras**:

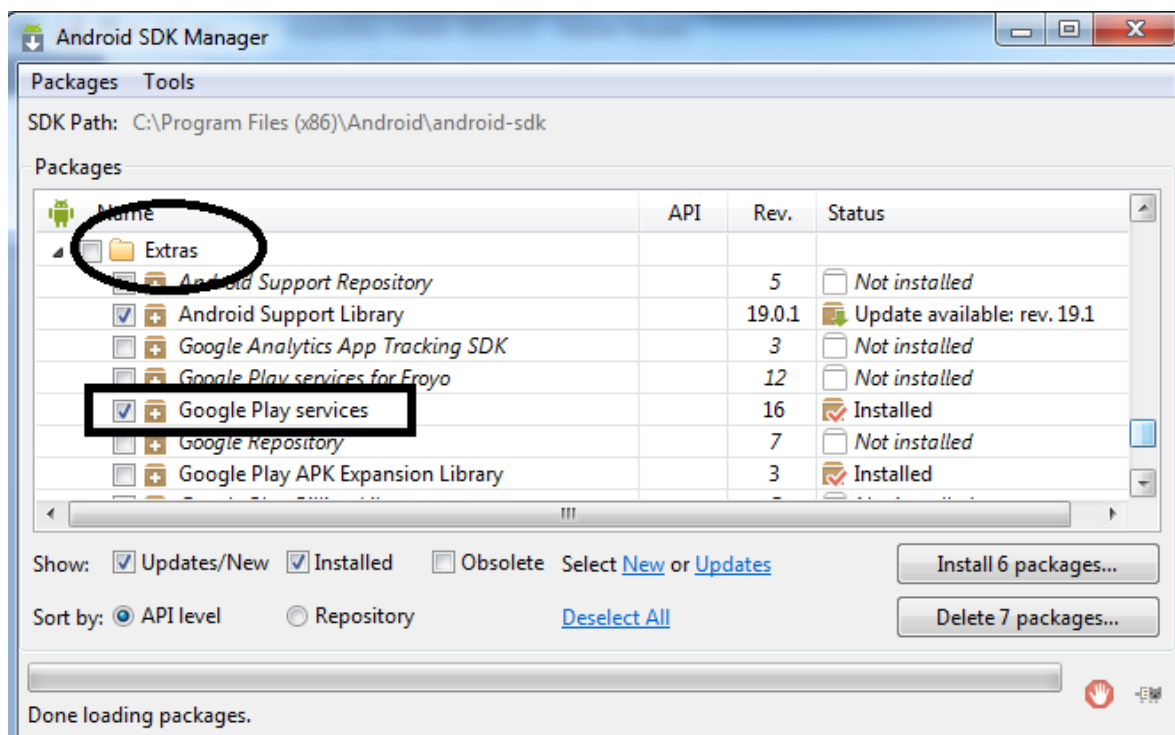


Figura 32. Instalación de google play services

Para poder usar los servicios de google plus necesitaremos activar el API en la consola de google y registrar nuestro fichero de instalación APK con el certificado público en la consola de las google apis.



Encontraremos este certificado en la página de preferencias del eclipse en el apartado de Android → build:

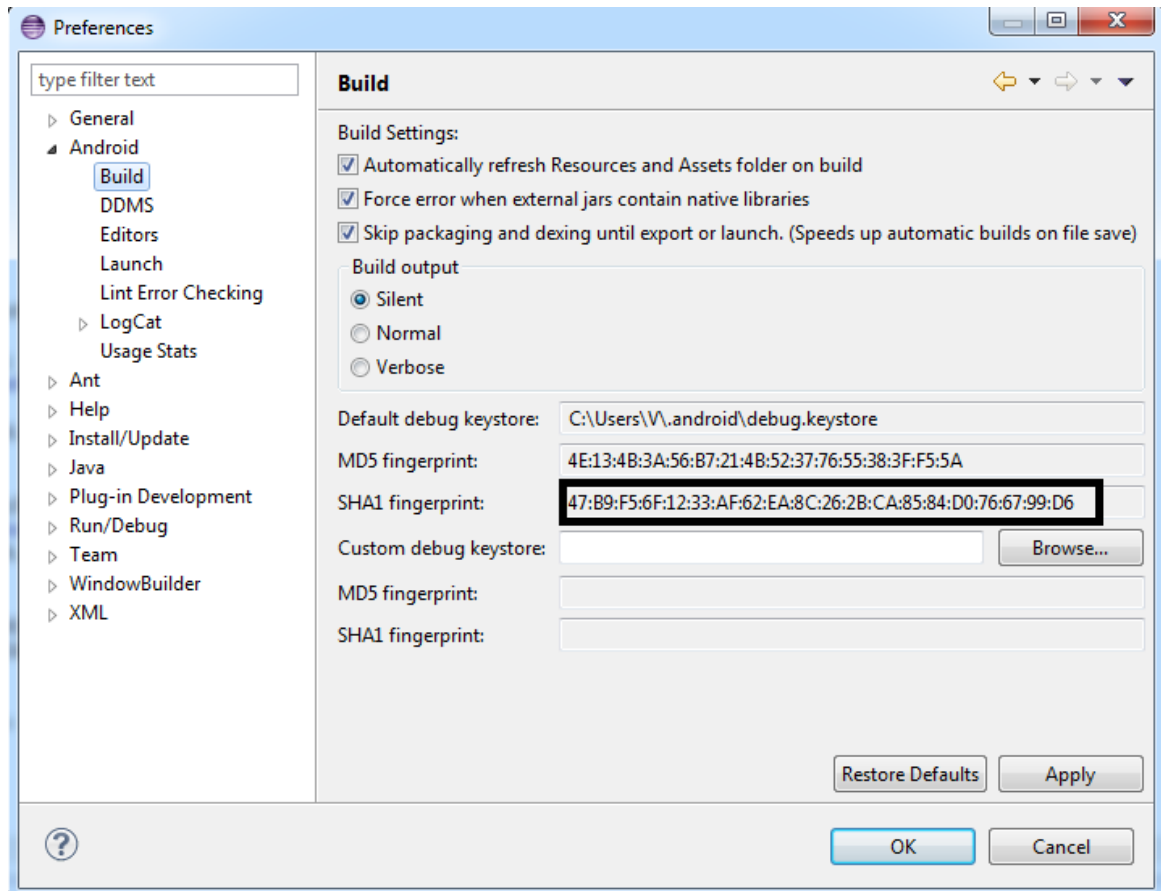


Figura 33. Certificado público de nuestra aplicación

A continuación abriremos la consola de las google APIS:

<https://code.google.com/apis/console>

Y activaremos el servicio de google+ en la sección de **APIS**:

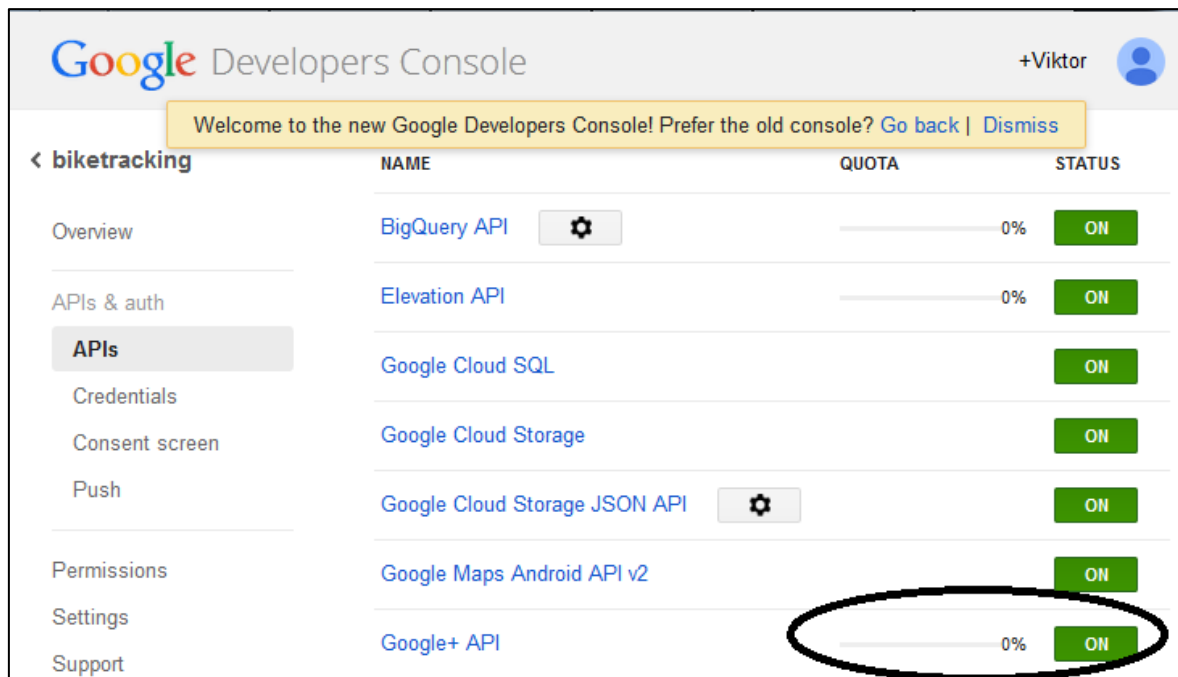
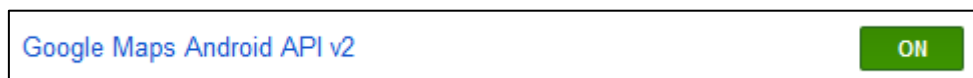


Figura 34. Consola de google developers.

Ya de paso activaremos el uso también la api de google Maps que nos hará falta también más adelante.



Y a continuación en el apartado de Credentials, debajo de APIs:

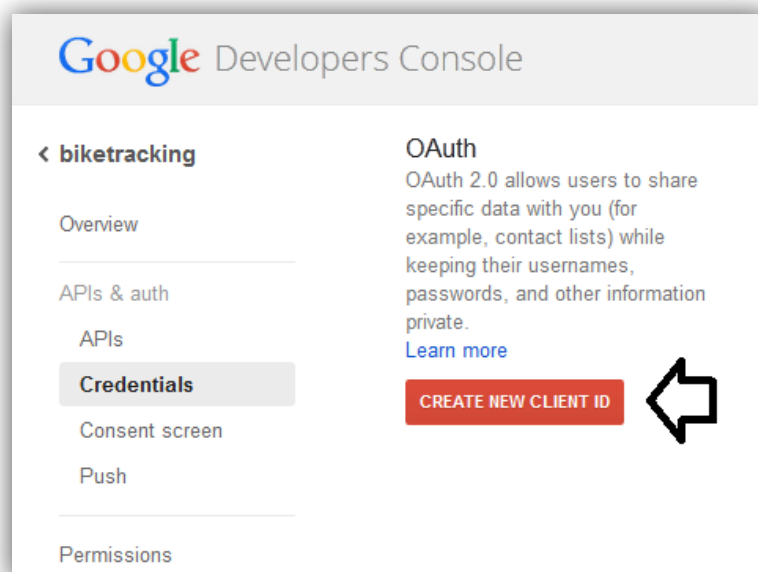


Figura 35. Creación del id de cliente.



Accedemos y pondremos todos los datos que nos piden y nos quedará como a continuación:

Create Client ID

**Application type**

- ☐ Web application  
Accessed by web browsers over a network.
- ☐ Service account  
Calls Google APIs on behalf of your application instead of an end-user. [Learn more](#)
- ☒ Installed application  
Runs on a desktop computer or handheld device (like Android or iPhone).

**Installed application type**

- ☒ Android [Learn more](#)  
API requests are sent directly to Google from your clients' Android devices. Google verifies that each request originates from an Android application that matches the package name and SHA1 signing certificate fingerprint name listed below.

Package name:

Signing certificate fingerprint (SHA1):

Deep linking:  
☒ Enabled  
☐ Disabled

- ☐ Chrome Application [Learn more](#)
- ☐ iOS [Learn more](#)
- ☐ Other

Figura 36. Datos a introducir para la creación del id de cliente.

El certificado *SHA1* es el que habíamos recuperado del menú de preferencias del IDE Eclipse que se ha comentado antes.

El **package** es el que tenemos en nuestra aplicación Android.

Habiendo llegado a este punto ya tendremos configurado el uso de google+ en la consola de google plus:



Client ID for Android application	
Client ID	1072973638467-cl3ui7j0uk55i9tfk16gn449d0bg6ech.apps.googleusercontent.com
Redirect URIs	urn:ietf:wg:oauth:2.0:oob http://localhost
Package name	project.vcanof.biketracking
Certificate fingerprint (SHA1)	47:b9:f5:6f:12:33:af:62:ea:8c:26:2b:ca:85:84:d0:76:67:99:d6
Deep linking	Enabled

[Edit settings](#) [Download JSON](#) [Delete](#)

Figura 37. Id cliente de nuestra aplicación.

A continuación nos faltara configurar el Eclipse importando la librería de google play en nuestro workspace de Eclipse, desde **File ⇒ Import ⇒ Android ⇒ Existing Android Code Into Workspace** y seleccionando la carpeta donde tenemos el archivo **google-play-services\_lib**. Debería tener una ruta como la siguiente:

**android-sdk \extras\google\google\_play\_services\libproject\google-play-services\_lib**

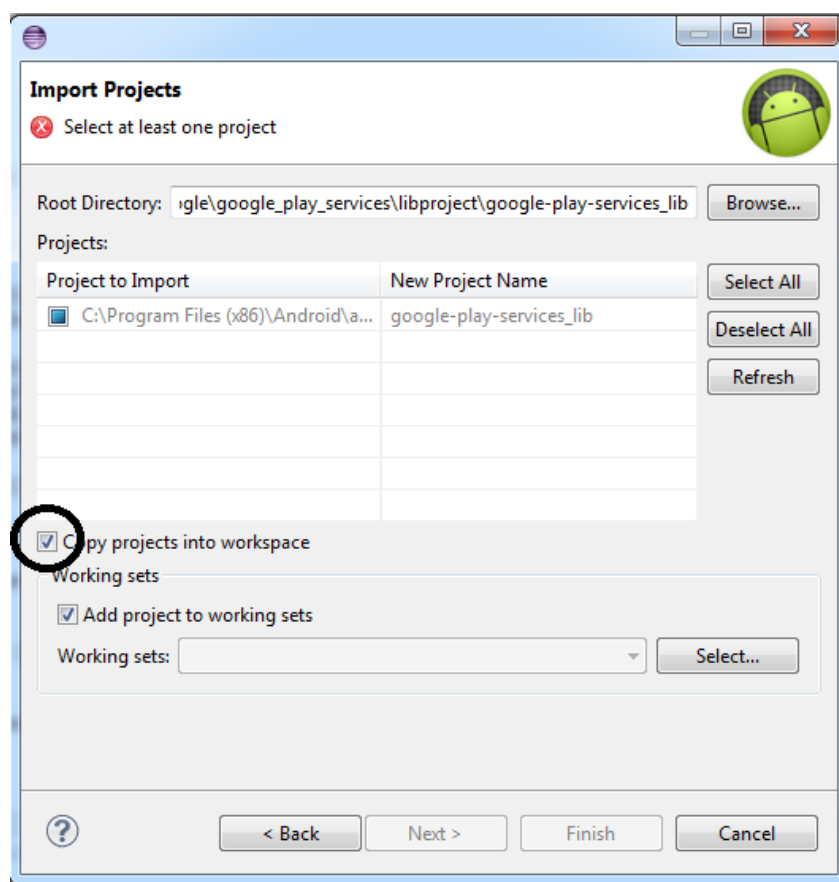


Figura 38. Importación a nuestro proyecto de google play services



Y ya para acabar deberíamos asociar esta librería de google play services a nuestro proyecto Android desde la opción de preferencias del proyecto como se muestra.

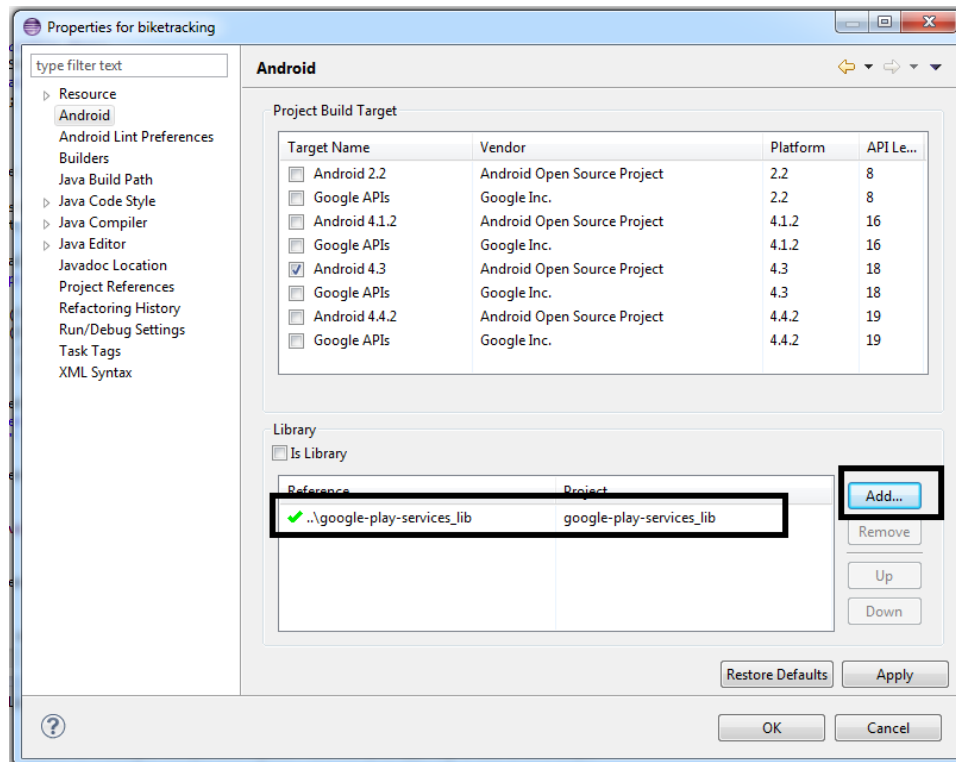


Figura 39. Importación a nuestro proyecto de google play services

Lo último es modificar el **AndroidManifest.xml** tener los permisos y metadatos adecuados para utilizar las APIS añadiendo las siguientes líneas:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.INTERNET"/>

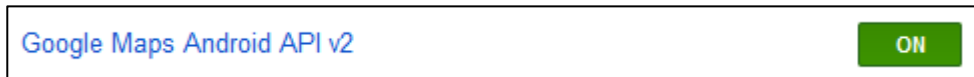
<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

### 5.1.1.3 Google Maps Android API v2

Ahora ya teniendo instalado la carpeta de la librería de google play services en nuestro proyecto de Android ya es tarea fácil habilitar la utilización de google maps.



Lo siguiente sería ir otra vez a la consola de las google apis como hicimos en la del google+ y asegurarnos que tenemos el servicio de google maps activado como se comentó también anteriormente.



Hecho esto ya no queda crear una key para habilitar el uso, lo haremos desde aquí:

#### Public API access

Use of this key does not require any user action or consent, does not grant access to any account information, and is not used for authorization.

[Learn more](#)

CREATE NEW KEY



Figura 39. Creación de una nueva llave de google maps.

Seleccionaremos **Android Key** en la ventana que nos saldrá, y tendremos esta pantalla:

Aquí tendremos que introducir el ya conocido certificado y seguido de punto y coma el nombre completo de nuestro proyecto tal como se muestra.



Create an Android key and configure allowed Android applications

**This key can be deployed in your Android application.**  
API requests are sent directly to Google from your client Android device. Google verifies that each request originates from an Android application that matches one of the certificate SHA1 fingerprints and package names listed below. You can discover the SHA1 fingerprint of your developer certificate using the following command:  
`keytool -list -v -keystore mystore.keystore`  
[Learn more](#)

**Accept requests from an Android application with one of the certificate fingerprints and package names listed below**  
 One SHA1 certificate fingerprint and package name (separated by a semicolon) per line. Example:  
 45:B5:E4:6F:36:AD:0A:98:94:B4:02:66:2B:12:17:F2:56:26:A0:E0;com.example

47:B9:F5:6F:12:33:AF:62:EA:8C:26:2B:CA:85:84:D0:76:67:99:D6;project.vcanof.biketracking

➡

Create

Cancel

Figura 40. Creación de una nueva llave de google maps.

Ya con esto hecho tendremos la siguiente key:

Key for Android applications

API key	AIZA5yCbXj4M1e-lzJvn63nUmXwrejfGp0dGowk
Android applications	47:B9:F5:6F:12:33:AF:62:EA:8C:26:2B:CA:85:84:D0:76:67:99:D6;project.vcanof.biketracking
Activation date	Mar 30, 2014 2:36 PM
Activated by	viktoronline2@gmail.com (you)

Edit allowed Android applications

Regenerate key

Delete

Figura 41. Llave de google maps.

Esta key la tendremos que poner en nuestro **Androidmanifest.xml** como metadatos junto a otros permisos que nos harán falta:

```
<meta-data android:name="com.google.android.maps.v2.API_KEY"
           android:value="AIZA5yCbXj4M1e-lzJvn63nUmXwrejfGp0dGowk"/>
```





```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.INTERNET"/> ya lo teniamos
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
```

Con todo este proceso ya tendremos funcional el uso de **Google Maps Android API v2** en nuestro proyecto.

## 5.2 Instalación de nuestro servidor web Raspberry pi rev 2.

Para el soporte web de nuestro proyecto se requiere de un servidor web, a continuación se explicara su instalación.

En un primer momento se optó por instalar un servidor web local tipo xampp para Windows que llegó a ser funcional y accesible desde internet, pero al final al tener oportunidad de poder utilizar un **raspberry pi** se procedió a hacer la migración a este último.

Cabe mencionar que la instalación y puesta en servicio del servidor web Apache + php + mysql para Windows con el aplicativo XAMPP fue más complicada que de la del raspberry pi, dado que se tuvo que modificar muchos más parámetros de configuración en ficheros de inicialización.

Tenemos a nuestra disposición un Raspberry PI Rev2 que cuenta con las siguientes características:



Figura 42. Nuestro raspberry Pi.

- SoC: Broadcom BCM2835 (CPU, GPU y SDRAM)
- CPU ARM1176JZF-S a 700 MHz
- GPU Broadcom VideoCore IV
- Memoria RAM 512 MB (compartidos con la GPU)
- Conexiones
  - 2 x USB 2.0
  - 1 x Salida audio mini jack 3.5 mm
  - 1 x Salida audio/vídeo HDMI
  - 1 x Salida vídeo compuesto RCA
  - 1 x Micro USB
  - 1 x RJ45 10/100 Ethernet RJ45
- Conectividad
  - LAN Red local 10/100
  - WiFi (mediante adaptador USB WiFi compatible, no incluido)
- Slot SD para tarjetas
- Alimentación: 5V/700 mA (3.5 W) vía microUSB



- Dimensiones: 85.6 mm x 53.98 mm

Partiremos de una base de **Linux Raspbian** ( basada en debian ) desde cero en la que la instalaremos en una tarjeta SD preparada para la ocasión. Primeramente descargaremos la imagen de la distribución desde la web de raspbian:

<http://www.raspberrypi.org/downloads>


Version	1.3.4
Documentation	<a href="#">Link</a>
<b>Raw Images</b>	
The following raw images are intended for advanced users. To use an image file, you will need to unzip it and write it to a suitable (2GB or larger, 4GB or larger for Raspbian) SD card using the UNIX tool <a href="#">dd</a> . Windows users should use <a href="#">Win32DiskImager</a> . Do not try to drag and drop or otherwise copy over the image without using dd or Win32DiskImager – it won't work. If you're still not clear on what to do, the community on the Raspberry Pi Wiki has written a <a href="#">guide for beginners</a> on how to set up your SD card.	
<b>Raspbian</b>	
	
Image	<a href="#">2014-01-07-wheezy-raspbian.zip</a>
Torrent	<a href="#">2014-01-07-wheezy-raspbian.zip.torrent</a>
SHA-1 Checksum	9d0afb932ec22e3c29d793693f58b0406bcab86
Default login	pi / raspberry
Description	A community-created port of Debian wheezy, optimised for the Raspberry Pi
Release Date	2014-01-07
Version	wheezy
Kernel	3.10
URL	<a href="#">Link</a>
Release Notes	<a href="#">release-notes.txt</a>

Figura 43. Descarga raspbian.

Instalamos la imagen de Linux en la tarjeta SD con el programa [Win32DiskImager](#). Con esto hecho ya podremos arrancar el sistema desde la tarjeta SD que acabamos de grabar.

**Iniciado el S.O. procederemos a instalar/configurar 5 tipos de programas:**

(1/5) Acceso SSH (para administrar nuestro servidor remotamente)



(2/5) Apache (servidor)

(3/5) PHP ( para las páginas de acceso a la base de datos )

(4/5) MySQL ( base de datos )

(5/5) phpMyAdmin ( para administrar fácilmente las bases de datos con las que trabajaremos)

### **5.2.1 Habilitar SSH**

Primeramente habilitaremos el SSH ya que nos permitirá ejecutar comandos de consola remotamente desde otro ordenador.

Los comandos utilizados para ello son:

```
sudo service ssh start
```

Y para iniciar el servicio:

```
sudo insserv ssh
```

### **5.2.2 Servidor Web – Apache**

Primero crearemos el grupo de usuarios para el servidor:

```
sudo groupadd www-data
```

```
sudo usermod -a -G www-data www-data
```

Para evitarnos posibles problemas actualizaremos los repositorios de la distribución:

```
sudo apt-get update
```

Y tampoco estará de más actualizar los programas, ejecutando:



```
sudo apt-get upgrade
```

Y ya para instalar apache en nuestro servidor:

```
apt-get install apache2
```

Con todo esto ya deberíamos tener operativo apache y accediendo a la IP que tiene la raspberry podríamos ver ya la página de prueba por defecto con el mensaje de “**It Works!**”; como aparece en la siguiente figura:

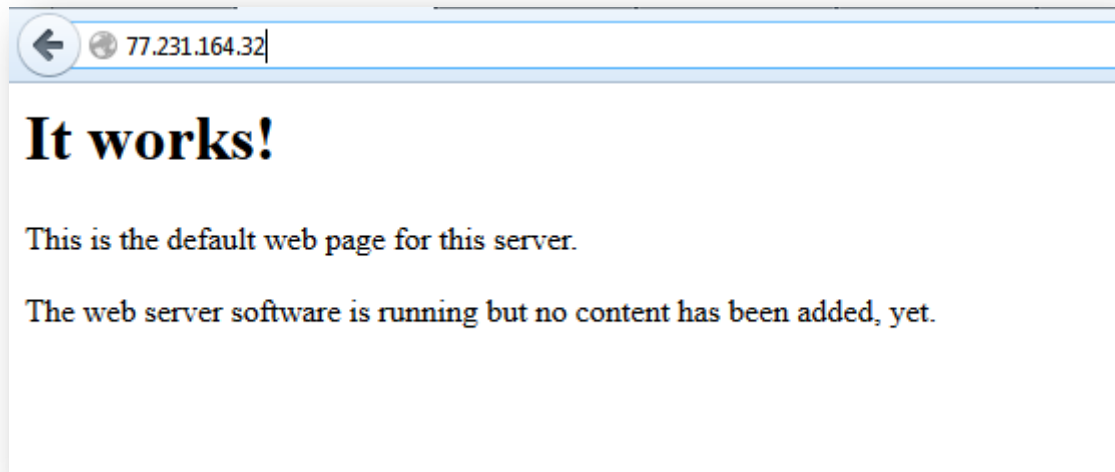


Figura 44. Prueba servidor web.

Cabe comentar que todos ficheros a los que se podrán acceder a través del servidor web están ubicados en el directorio `/var/www/`. Mas adelante ya comentare que permisos tendremos que darle para poder acceder a ellos ya que sin los permisos adecuados nos saldrá un error de *Permission not allowed* al intentar acceder a ellos.

### 5.2.3 Servidor web – PHP

Instalaremos **php** a través de este comando de consola:



```
sudo apt-get install php5
```

A continuación instalaremos más paquetes que nos podrán ser de utilidad y/o necesarios para nuestro servidor:

```
sudo apt-get install libapache2-mod-php5 libapache2-mod-perl2 php5 php5-  
cli php5-common php5-curl php5-dev php5-gd php5-imap php5-ldap php5-mhash  
php5-mysql php5-odbc
```

En este paso está recomendado reiniciar el SO ( `sudo reboot` ).

Después de este reinicio ya deberíamos tener funcional nuestro php.

#### 5.2.4 Servidor web - MySQL

Para utilizar las bases de datos nos hará falta instalar Mysql. Por lo que se tendrá que instalar de la siguiente manera:

```
sudo apt-get install mysql-server mysql-client php5-mysql
```

Durante la instalación nos pedirá una contraseña para el administrador de la base de datos.

Al acabar la instalación haremos lo mismo que en la instalación de php, reiniciaremos el S.O. ( `sudo reboot` ) e iniciaremos el servidor de Mysql:

```
sudo service mysql start
```

Al acabar con todo esto debería estar funcionando sin problemas nuestro mysql.

#### 5.2.5 Servidor web – phpMyAdmin



Para gestionar las bases de datos hemos optado por la utilización del ya superconocido phpMyAdmin, con una interfaz web con mucha ayuda y muy fácil de utilizar. Vamos a por la instalación:

```
sudo apt-get install libapache2-mod-auth-mysql php5-mysql phpmyadmin
```

Para que phpMyAdmin funcione correctamente nos hará falta añadir una línea en el fichero **/etc/php5/apache2/php.ini**, que es la siguiente:

```
extension=mysql.so
```

Insertada en la sección de “Dynamics Extension”.

A partir de ahora ya podríamos acceder a la web del phpMyAdmin → <http://IPRaspberry/phpmyadmin/> ), el usuario será root y la contraseña será la que hayamos elegido cuando lo instalamos. Debería salirnos la siguiente pantalla:

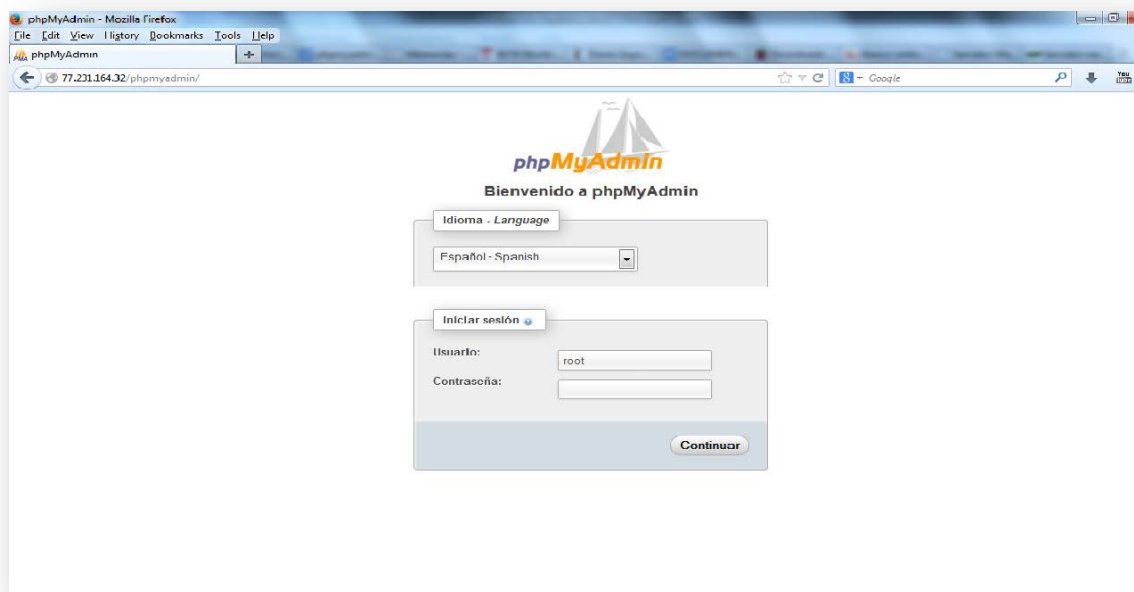


Figura 45. Página login phpMyadmin

Introduciendo nuestros datos de administrador podremos acceder a administrar muy



fácilmente nuestra base de datos:

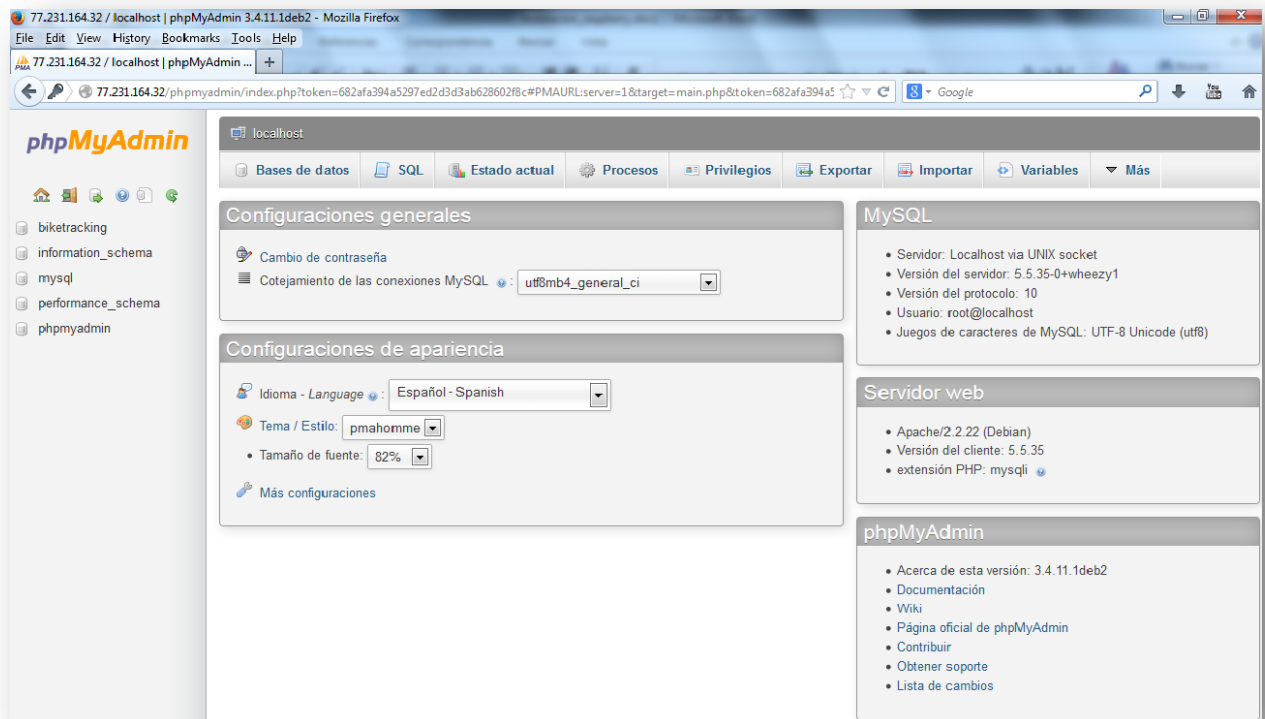


Figura 46. phpMyadmin

Para acabar, para que tengamos acceso desde internet / lan a nuestra carpeta con las páginas php deberemos de asignar los permisos adecuados para ello:

```
sudo chmod -r 775 /var/www
```

Ahora ya podemos decir que ya tenemos nuestro servidor web Raspberry completo y listo para la acción: **con apache, mysql, php y phpmyadmin en servicio.**

Como tenemos instalado el SSH no precisamos de ratón y teclado, y ni de una pantalla externa para acceder a su configuración.

### ***5.2.6 Configuración del router para permitir el acceso desde internet a nuestro servidor Raspberry Pi ( aplicable también a xampp )***

Ahora ya podremos conectar nuestro raspberry al router pero necesitaremos algo más.





Figura 47. Router con el raspberry.

Para poder acceder a nuestro servidor nos hará falta configurar el router de nuestra conexión ADSL para que a través del puerto 80 se dirija a nuestra IP local que tiene nuestro raspberry.

El proceso es el siguiente:

- 1.- Accedemos a la página del router de entrada a través de un explorador web:

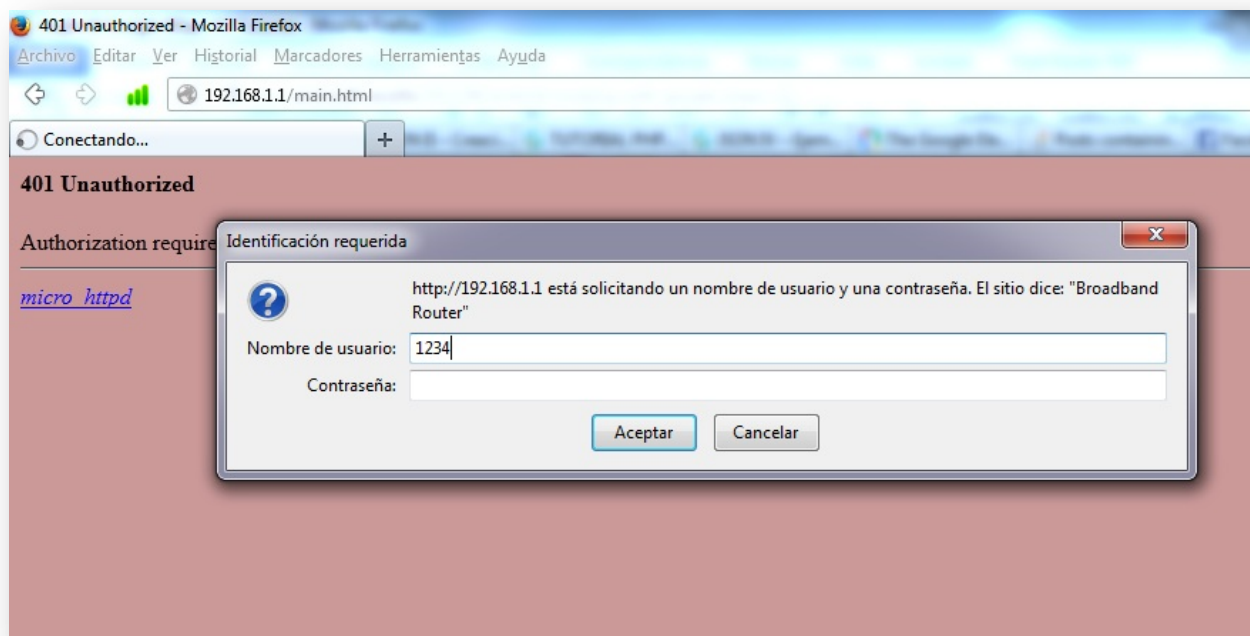


Figura 48. Login router.

La IP normalmente es del tipo 192.168.x.1 siendo casi siempre 192.168.1.1. Introduciendo el usuario y contraseña ya podremos acceder al menú de configuraciones:

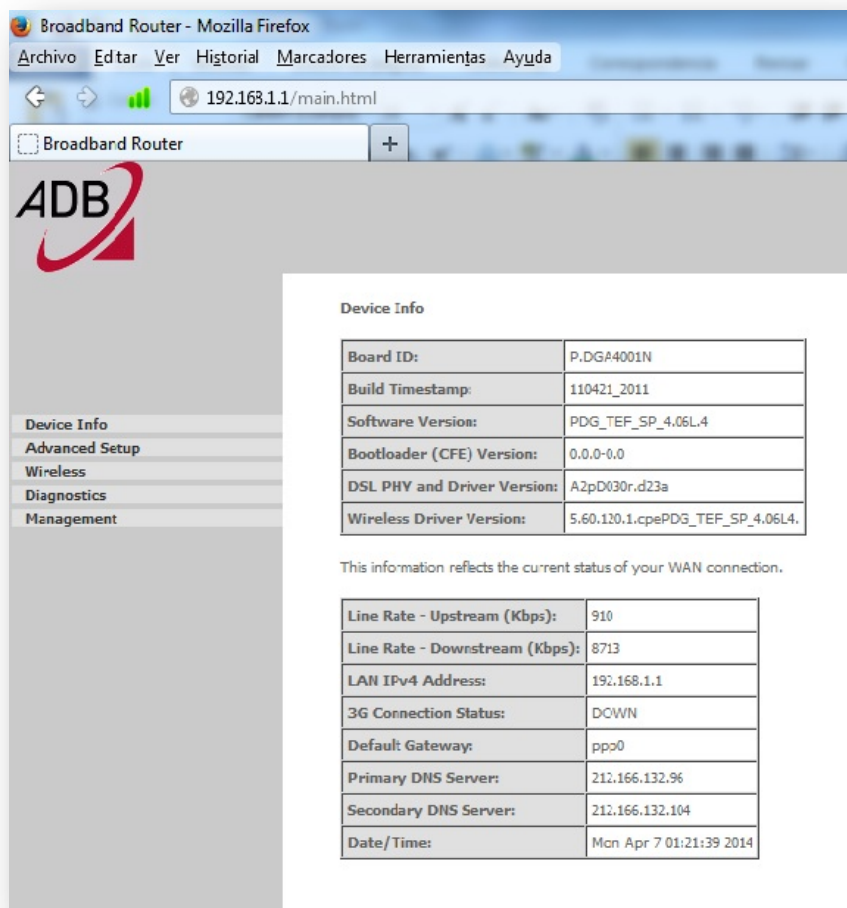


Figura 49. Menú router.

Ahora abriremos el menú → Advanced Setup → NAT → Virtual Servers y nos aparecerá la pantalla siguiente



Broadband Router - Mozilla Firefox

192.168.1.1/main.html

Broadband Router

ADB

Device Info  
Advanced Setup  
Layer2 Interface  
WAN Service  
LAN  
NAT  
Virtual Servers  
Port Triggering  
DMZ Host  
Security  
Parental Control  
Quality of Service  
Routing  
DNS  
DSL  
3G Key  
UPnP  
DNS Proxy  
Print Server  
Storage Service  
Interface Grouping  
Certificate  
Multicast  
Wireless  
Diagnostics  
Management

NAT -- Virtual Servers

Select the service name, and enter the server IP address and click "Apply/Save" to forward IP packets for this service to the specified server. **NOTE: The "Internal Port End" cannot be modified directly. Normally, it is set to the same value as "External Port End". However, if you modify "Internal Port Start", then "Internal Port End" will be set to the same value as "Internal Port Start".**  
Remaining number of entries that can be configured: 24

Use Interface:

Service Name:  
☐ Select a Service:   
☒ Custom Service:

Server IP Address:

External Port Start	External Port End	Protocol	Internal Port Start	Internal Port End
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		
		TCP		

Figura 50. Página configuración puertos.

Tendremos que abrir el puerto 80 y redirigirlo a la IP de nuestro servidor web.

Poniendo los datos que nos hacen falta: IP del servidor, external port 80 e internal port 80

Protocol TCP/UDP nos debería quedar de la siguiente manera:



Broadband Router - Mozilla Firefox

Archivo Editar Ver Historial Marcadores Herramientas Ayuda

192.168.1.1/main.html

Broadband Router

**ADB**

**NAT -- Virtual Servers Setup**

Virtual Server allows you to direct incoming traffic from WAN side (identified by Protocol and External port) to the Internal server with private IP address on the LAN side. The Internal port is required only if the external port needs to be converted to a different port number used by the server on the LAN side. A maximum 32 entries can be configured.

Add Remove

Server Name	External Port Start	External Port End	Protocol	Internal Port Start	Internal Port End	Server IP Address	WAN Interface	Remove
ll	3306	3306	TCP/UDP	3306	3306	192.168.1.44	ppp0	<input type="checkbox"/>
uTorrent (TCP)	15329	15329	TCP	15328	15328	192.168.1.64	ppp0	<input type="checkbox"/>
uTorrent (UDP)	15329	15329	UDP	15328	15328	192.168.1.64	ppp0	<input type="checkbox"/>
none	80	80	TCP/UDP	80	80	192.168.1.44	ppp0	<input type="checkbox"/>
Teredo	49898	49898	UDP	49898	49898	192.168.1.44	ppp0	<input type="checkbox"/>
Teredo	51584	51584	UDP	51584	51584	192.168.1.44	ppp0	<input type="checkbox"/>
uTorrent (TCP)	15328	15328	TCP	15328	15328	192.168.1.44	ppp0	<input type="checkbox"/>
uTorrent (UDP)	15328	15328	UDP	15328	15328	192.168.1.44	ppp0	<input type="checkbox"/>

Add Remove

Device Info  
Advanced Setup  
Layer2 Interface  
WAN Service  
LAN  
NAT  
Virtual Servers  
Port Triggering  
DMZ Host  
Security  
Parental Control  
Quality of Service  
Routing  
DNS  
DSL  
3G Key  
UPnP  
DNS Proxy  
Print Server  
Storage Service  
Interface Grouping  
Certificate  
Multicast  
Wireless  
Diagnostics  
Management

Figura 51. Página información puertos.

Ahora ya tendremos acceso desde internet a nuestro servidor web a través del puerto 80 y podremos utilizar nuestra aplicación Android sin problemas de comunicación con nuestro servidor web.



### 5.3 Implementación.

Tenemos dos partes muy bien diferenciadas en la parte de implementación en nuestro proyecto. Por un lado la implementación de la parte de la aplicación en Android y por la otra, la implementación de la parte del servidor web.

La base de datos se montará en mysql y las comunicaciones con ella se realizará sobre peticiones a páginas php a través de paso de parámetros por POST y retornadas en formato string / JSON.

El siguiente esquema lo expone gráficamente:

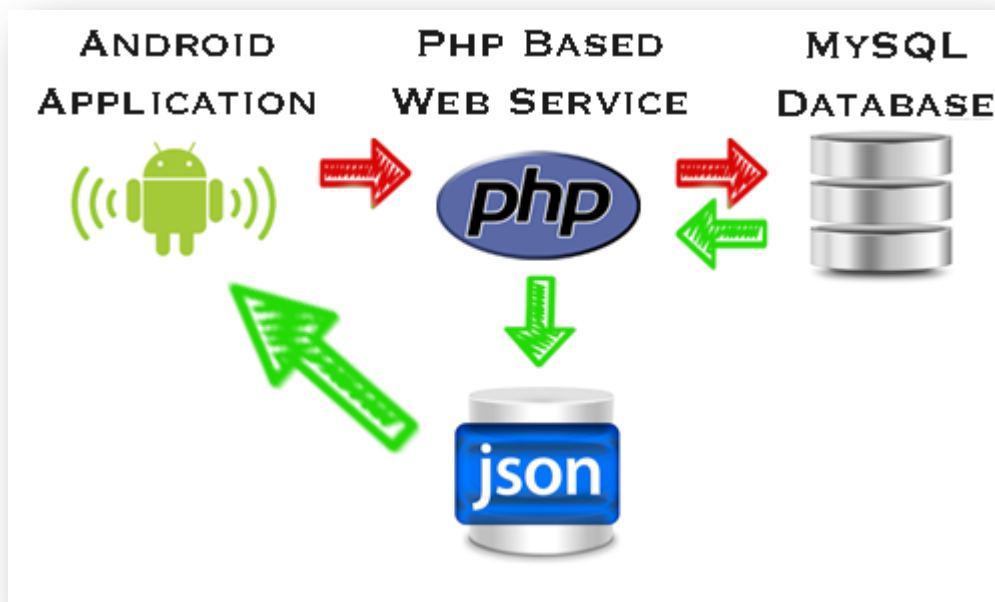


Figura 52. Diagrama flujo datos de la aplicación

#### 5.3.1 Implementación de la aplicación Android.

Empezaremos por comentar la como es la estructura de nuestra aplicación Android:

##### 5.3.1.2 Estructura de nuestro proyecto Android

Todas las aplicaciones Android tienen un esquema básico predefinido. Nuestra aplicación tiene la siguiente estructura:



- Carpeta **src** : donde están los archivos java del código de las *activities* y de las *clases*. En nuestra aplicación lo tenemos separado en dos paquetes:
  - 1.- **project.vcanof.biketracking**: donde están las todas las actividades (pantallas)
  - 2.- **project.vcanof.biketracking.clases**: donde se encuentran las clases.

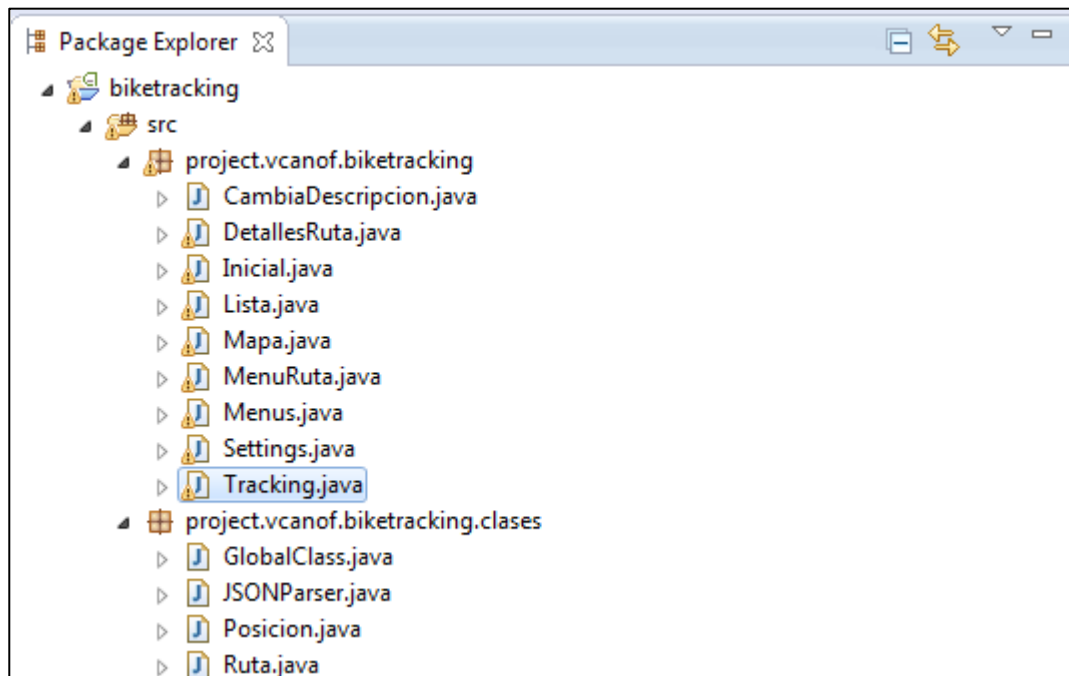


Figura 53. Estructura de la carpeta src

- Carpeta **gen**: contiene una serie de elementos de código **generados automáticamente** cuando compilamos el proyecto. Cabe comentar el fichero **R.java**, es importante ya que almacena una serie de constantes con los identificadores de todos los recursos de la aplicación incluidos en la carpeta **res**, de forma que podamos acceder fácilmente a estos recursos desde nuestro código a través de este dato.

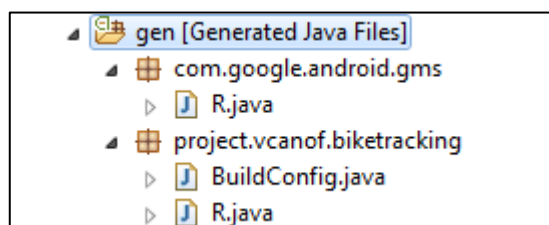


Figura 54. Estructura de la carpeta gen





- Carpeta **bin**: contiene los elementos compilados de la aplicación y otros ficheros auxiliares ( jars del google play services ) . Cabe destacar el fichero **biketracking.apk**, que es el ejecutable de la aplicación que se podrá instalar en cualquier dispositivo.

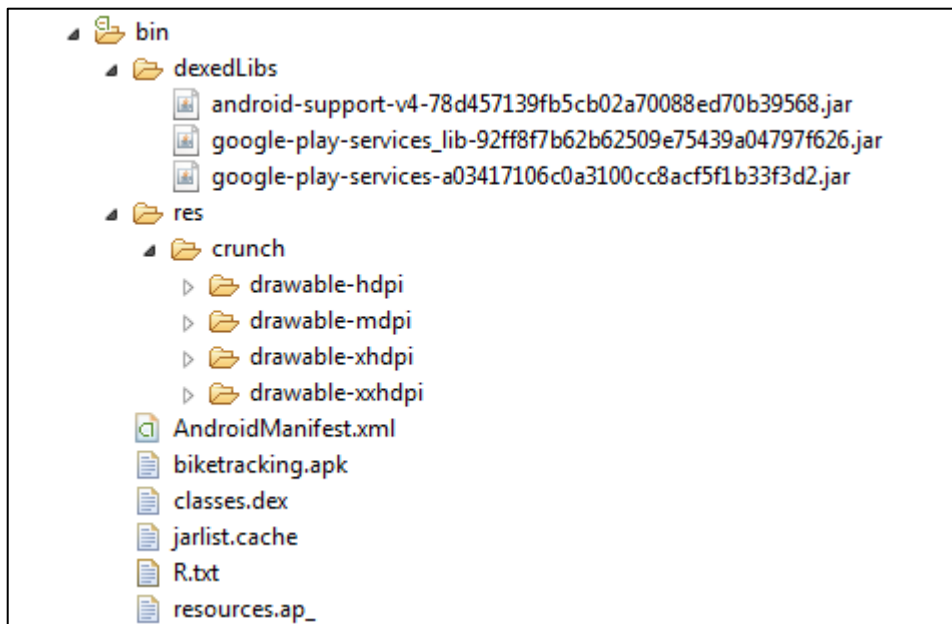


Figura 55. Estructura de la carpeta bin

- Carpeta **libs**: Contiene las librerías auxiliares:

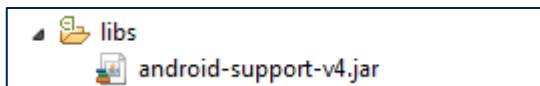


Figura 56. Estructura de la carpeta bin

- Carpeta **res**: contiene todos los ficheros de recursos necesarios para el proyecto: imágenes, vídeos, cadenas de texto, etc... En la subcarpteta layout están todos los ficheros XML de la interfaz gráfica de las activities como se muestra:



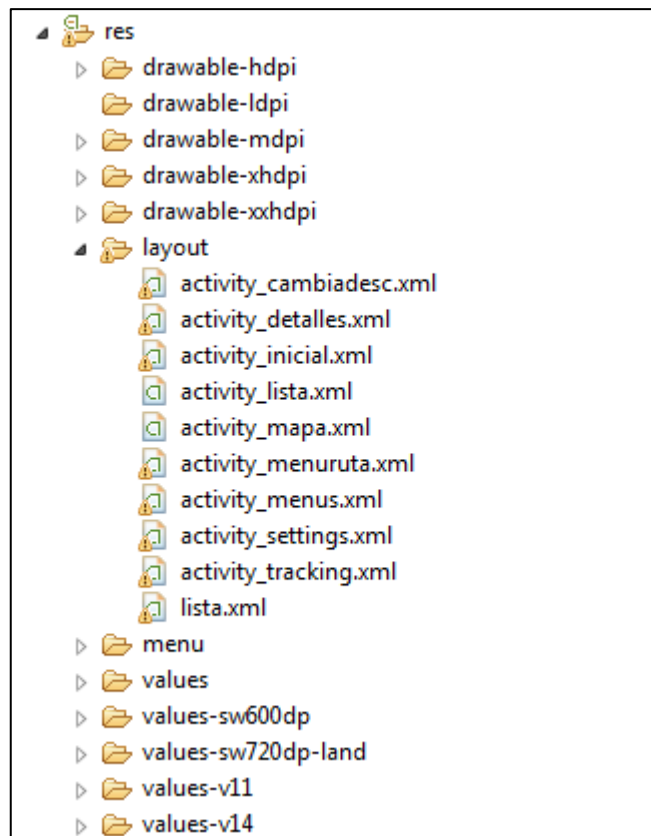


Figura 57. Estructura de la carpeta res

En la raíz de toda la estructura nos encontramos con un fichero muy importante en la aplicación: **androidmanifest.xml**. Este fichero contiene la definición en XML de los aspectos principales de la aplicación, como por ejemplo su identificación (nombre, versión, icono, ...), sus componentes (pantallas, mensajes, ...), las librerías auxiliares utilizadas, o los permisos necesarios para su ejecución. Durante el proceso de creación de código se hizo destacable la necesidad de tener bien controlado que contiene este archivo ya que al tener algún despiste al manipular este fichero provoca el no funcionamiento de nuestra aplicación, a continuación se detalla que contiene este archivo:

#### 5.3.1.3 Fichero *androidmanifest.xml*

En este fichero, en primer lugar contiene como se llama nuestro paquete de actividades:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="project.vcanof.biketracking"
    android:versionCode="1"
    android:versionName="1.0" >
```

A continuación especificamos que versiones de api Android mínima podrá ser usada en el dispositivo cliente y la versión con la que está pensada ser usada.



```
<uses-sdk
    android:minSdkVersion="9"
    android:targetSdkVersion="18" />
```

Ahora viene una parte fundamental de la existencia de este archivo, que permisos tiene que ser aceptados por el usuario al utilizar la aplicación:

```
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.USE_CREDENTIALS" />
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"
/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION" />

<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission
android:name="com.google.android.providers.gsf.permission.READ_GSERVICES"/>
```

Para usar el GPS pediremos permisos de access\_coarse\_location y access\_fine\_location, para la acceder a internet permission.INTERNET, la API de google maps requiere de permiso WRITE\_EXTERNAL\_STORAGE. La API de google+ requerirá de .GET\_ACCOUNTS y USE\_CREDENTIALS para su buen funcionamiento.

En el resto de fichero dentro de la etiqueta **application** especificaremos que actividades tiene nuestro proyecto y cuál de ellas es la que se ejecuta primero:

```
<activity
    android:name="project.vcanof.biketracking.Inicial"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Así sería como definiríamos a nuestra actividad principal y a continuación estarán declaradas las demás actividades:

```
<activity
    android:name="project.vcanof.biketracking.Menu"
    android:label="@string/app_name" >
</activity>
```

Y así sucesivamente con todas ( 7 más, 9 en total ). Al final de este archivo también nos hace falta declarar cierto tipo de metadatos requeridos para el funcionamiento de la API google maps ( la key ) y para el uso de los google play services:



```
<!-- Google API Key -->

<meta-data android:name="com.google.android.maps.v2.API_KEY"
    android:value="AIzaSyCbXj4M1e-lzJvn63nUmXwrejGp0dGowk"/>

<meta-data
    android:name="com.google.android.gms.version"
    android:value="@integer/google_play_services_version" />
```

A continuación se irán explicando cada una de las actividades y las 4 clases utilizadas:

Empezaremos primero con las clases ya que estas son las que se irán utilizando en la actividades.

#### 5.3.1.4 Clases

Hay cuatro y las tenemos en el paquete **project.vcanof.biketracking.clases**:

- GlobalClass.java
- JSONParser.java
- Posicion.java
- Ruta.java

##### 5.3.1.4.1 GlobalClass.java

Esta clase fue creada ante la necesidad de tener disponibles ciertos valores durante la ejecución de la aplicación desde cualquier actividad como son:

- Identificador del usuario
- Nombre del usuario ( obtenido de la API de g+ )
- El tiempo de actualización del GPS
- IP del servidor web

Todos los valores están definidos como String.

Para hacer que pueda ser accesible en toda la aplicación esta clase hace un extend de la clase Application: **public class** GlobalClass **extends** Application

La clase tiene getters y setters para los 4 valores.



#### 5.3.1.4.2 Ruta.java

Esta es la clase que utilizamos para cada una de las rutas. Incluye todos los valores asociadas a ella:

- Identificador de la ruta
- Descripción de la ruta
- Fecha de creación
- Hora de inicio
- Hora de finalización
- Identificador del usuario que la ha hecho
- Nombre del usuario autor
- Si es publica o no
- Velocidad máxima alcanzada
- Altitud ganada
- Tiempo total de la ruta

Todos estos atributos son String menos los del *identificador de ruta* que es un entero y el de *si es publica* o no, que es un booleano.

Se han implementado getters y setters para todas las variables.

#### 5.3.1.4.3 Posicion.java

Esta es la clase que define a cada una de la posiciones de nuestra aplicación, tiene las variables:

- Latitud
- Longitud
- Altitud
- Tiempo

Siendo todas de tipo **double** menos el tiempo que es de tipo **long**.

La clase tiene sus getters y setters para todas las variables.



#### 5.3.1.4.4 JSONParser.java

Esta clase es muy importante en nuestra aplicación, es la que se encarga de hacer las peticiones al servidor web y devolvernos el resultado en un objeto de tipo JSON. Es el pilar de nuestras comunicaciones con el servidor.

La cabecera del método que realiza la petición y nos devuelve un resultado es la siguiente:

```
public JSONObject makeHttpRequest(String url, String metodo, List<NameValuePair> params)
```

Pasamos los siguientes parámetros:

1. url → es el String que contiene la dirección del webservice que nos interesa hacer la petición. Por ejemplo: [http://77.231.164.32/biketracking/lista\\_rutas.php](http://77.231.164.32/biketracking/lista_rutas.php)
2. método → es el String que contendrá de que tipo es nuestra petición: POST / GET
3. params → pasamos los parámetros que nos interesa pasar a través de la petición para que lleguen a nuestro script en el servidor mediante una List de NameValuePair.

El código utilizado:

```
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(url);
httpPost.setEntity(new UrlEncodedFormEntity(params));
HttpResponse httpResponse = httpClient.execute(httpPost);
HttpEntity httpEntity = httpResponse.getEntity();
salida = httpEntity.getContent();
```

Nos guardaremos el retorno en un InputStream con el nombre de salida que posteriormente procesaremos para obtener un JSON como sigue:

```
BufferedReader reader = new BufferedReader(new
InputStreamReader(salida, "iso-8859-1"), 8);
StringBuilder sb = new StringBuilder();
String line = null;
while ((line = reader.readLine()) != null)
{
    sb.append(line + "\n"); // nueva linea
}
salida.close();
json = sb.toString();
```



A través de un `BufferedReader` con el `InputStream` del retorno de la petición iremos obteniendo los datos y los iremos almacenando en un `StringBuilder` para luego obtener un `String` y así poderlo convertir finalmente a un objeto de tipo `JSON`:

```
jObj = new JSONObject(json);  
  
return jObj;
```

Ya tenemos el retorno en un objeto `JSONObject` con el resultado de nuestra petición a la dirección del servidor.

### 5.3.1.5 Actividades

Tenemos 9 actividades en nuestro proyecto Android, las iremos comentando una a una:

#### 5.3.1.5.1 Activity Inicial.java

En esta pantalla, que es la principal e inicial, se ha utilizado la API de google+ para autenticar a la usuarios. Está asociada a la interfaz “activity\_inicial.xml” donde están definidos los botones para entrar como registrado o no registrado, el de iniciar sesión con google+, desconectar y revoke. Estos últimos botones de desconectar y revocar la autenticación están puestos para cumplir con la directrices que marca los términos de políticas de google developer [5] ( eliminar cualquier información del usuario cuando se desconecte el usuario y poder tener acceso a desconectarse )

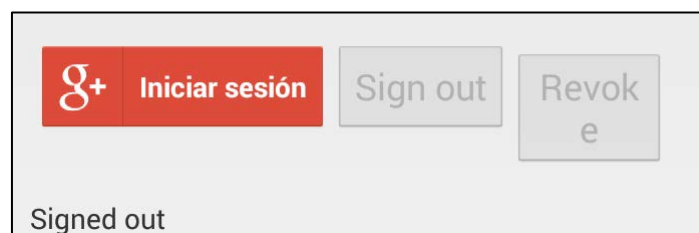


Figura 58. Login G+.

Para implementar esta actividad hemos utilizado como base un ejemplo que pone la web de google developers a disposición de los desarrolladores:

<https://github.com/googleplus/gplus-quickstart-android>

Se ha implementado una instancia de la clase `GoogleApiClient` para crear un enlace con los google play services y poder así poder establecer conexión con las APIS de google y el

[5] <https://developers.google.com/+/policias>



estado del autenticado.

```
private GoogleApiClient mGoogleApiClient;
```

Al inicio de la aplicación se llama a un método para montar nuestro GoogleApiClient denominado buildGoogleApiClient():

```
mGoogleApiClient = buildGoogleApiClient();
```

con el código siguiente:

```
private GoogleApiClient buildGoogleApiClient()
{
    return new GoogleApiClient.Builder(this)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .addApi(Plus.API, null)
        .addScope(Plus.SCOPE_PLUS_LOGIN)
        .build();
}
```

Donde indicamos en caso de conexión fallida donde tiene que retornar, y que tipo de alcance tiene nuestro autenticado (OAuth 2.0 )

Se definen diferentes códigos dependiendo del estado del ciclo de vida esté la actividad:

Al arrancar la activity:

```
protected void onStart()
{
    super.onStart();

    mGoogleApiClient.connect();
}
```

Se llama a conectar nuestro GoogleApiClient.

Al parar la actividad:

```
protected void onStop() {
    super.onStop();

    if (mGoogleApiClient.isConnected())
    {
        mGoogleApiClient.disconnect();
    }
}
```

Si vemos que seguimos conectados desconectaremos.

Se llamara a desconectar nuestro enlace GoogleApiClient.



Cuando tengamos establecida la conexión con la Api de google:

```
public void onConnected(Bundle connectionHint)
{
    mSignInButton.setEnabled(false);
    mSignOutButton.setEnabled(true);
    mRevokeButton.setEnabled(true);

    // Recuperamos informacion del usuario

    Person currentUser = Plus.PeopleApi.getCurrentPerson(mGoogleApiClient);

    mStatus.setText(String.format(getResources().getString(R.string.signed_in_as), c
currentUser.getDisplayName()));

    Toast.makeText(getApplicationContext(), "Bienvenido " +
currentUser.getDisplayName() + " !!", Toast.LENGTH_LONG).show();

    usuario = currentUser.getId();
    nombreusuario = currentUser.getDisplayName();

    entrang.setVisibility(View.VISIBLE);

    // Indicate that the sign in process is complete.
    mSignInProgress = STATE_DEFAULT;
}
```

Haremos no disponible el botón de sign-in y disponible el botón de entrar como usuario registrado, desconexión y revocar. Y a continuación tiraremos un mensaje de bienvenida con el nombre del usuario que hemos obtenido de la API de google+ mediante la clase Person → Plus.PeopleApi.getCurrentPerson(mGoogleApiClient)

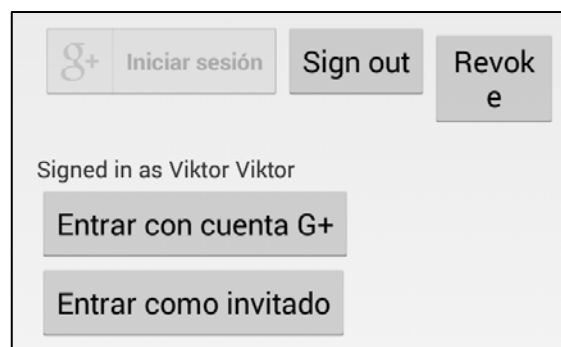


Figura 59. Login realizado G+.

Cuando el usuario ya este autenticado y apriete el botón de “entrar con cuenta G+” tenemos el siguiente código:





```
final GlobalClass globalVariable = (GlobalClass) getApplicationContext();
    globalVariable.setUsuario(usuario);
    globalVariable.setNombreUsuario(nombreusuario);

    Intent intent = new Intent(Inicial.this, Menus.class);

    //Creamos la información a pasar entre actividades

    Bundle b = new Bundle();
    b.putString("NOMBRE", usuario);

    //Añadimos la información al intent
    intent.putExtras(b);

    //Iniciamos la nueva actividad
    startActivity(intent);
```

Donde le enviaremos a la siguiente actividad ( la del menú principal ) el nombre de usuario para que la pueda mostrar en la pantalla y se lanzará la actividad del menú principal.

#### 5.3.1.5.2 Activity Menus.java

Esta actividad es en la que podremos acceder al resto de pantallas de la aplicación. Si se es un usuario registrado tendremos acceso al botón “**ver tus rutas**” y en caso contrario aparecerá no accesible ( modo disabled ). La interfaz la tenemos implementada en el fichero XML: “ activity\_menus.xml”y cuenta con el botón de “empezar nuevo tracking” que nos lanzaría el activity Tracking, el botón de “settings” para la pantalla de ajustes Settings.java y el botón de “ver rutas públicas” el cual nos enviaría al activity Lista al igual que el botón “ver tus rutas” del caso de usuario registrado.

Todos estos botones tienen sus respectivos *setOnClickListener* para que cuando sean pulsados nos lancen sus correspondientes pantallas pero con varias diferencias entre ellos:

- Botón de “empezar nuevo tracking”: implementa una verificación por si tiene o no conectado el GPS, tal que:

```
LocationManager locationManager;
locationManager=(LocationManager)(getApplicationContext().getSystemService(LOCATI
ON_SERVICE));
// miramos si esta habilitado
isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

if (!isGPSEnabled )
{
    conectaGPS (); // lanzamos dialogo aviso
```



```
// no hay GPS habilitado
}
else
{
// tenemos el GPS habilitado
Intent intent = new Intent(Menus.this, Tracking.class);
startActivity(intent);
}
}
```

Si está habilitado lanzamos la activity de tracking.java y en caso contrario mostramos un dialogo de aviso mediante un **AlertDialog** con un OK:

```
public void conectaGPS ()
```

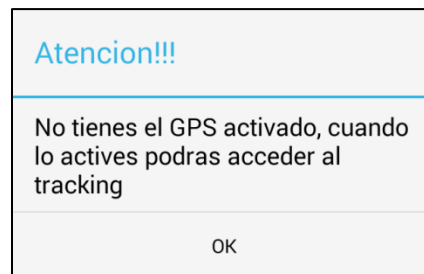


Figura 60. Dialogo aviso de GPS deshabilitado.

- Botón de “Ver tus rutas”: Lanzaremos una actividad Lista pasando como parámetro dentro de un Bundle el modo al que queremos entrar, en este caso modo privado.

```
Intent intent = new Intent(Menus.this, Lista.class);
Bundle b = new Bundle();
b.putString("modo", "privado");
intent.putExtras(b);
startActivity(intent);
```

- Botón de “Ver rutas públicas”: Lanzaremos una actividad Lista como la anterior pero en vez de un parámetro modo privado ahora público.
- Botón de settings: solamente lanzara la actividad de Settings.

### 5.3.1.5.3 Activity Settings.java

Esta es la pantalla en la que podremos cambiar un par de valores importantes de la aplicación: la IP del servidor web y el tiempo de actualización y registro de la posición



GPS ( en ms ). La interfaz la tenemos definida en el fichero XML `activity_menus` y consta de 4 `TextView` para la visualización de los datos actuales, dos `EditText` para introducir los nuevos valores y con sus dos correspondientes botones “Cambiar” para llevar a cabo la actualización. Los dos valores son guardados en dos variables de la clase `GlobalClass` por lo que son accesibles durante toda la aplicación por las diferentes actividades que requieren conocer estos valores para su ejecución. Son:

- `tiempoActualizacion`: cuando es presionado el botón cambiar del tiempo de actualización se realiza el cambio del valor  
`globalVariable.setServidor(String.valueOf(camp.getText()));`  
donde “camp” es el `editText` en el que se ha introducido el nuevo valor.
- `servidor`: como en el caso anterior cuando se lanza el `onClickListener` llamaremos al setter de la `globalVariable` para llevar a cabo la actualización.

En ambos casos acabamos la ejecución de la actividad con un `finish ()`; y lanzando un `Toast` ( visualizador de mensajes ) indicando que se ha realizado el cambio al valor que le hemos introducido:

```
Toast.makeText(getApplicationContext(), "Cambiado servidor a: " +  
globalVariable.getServidor(), Toast.LENGTH_LONG).show();
```

#### 5.3.1.5.4 Activity Tracking.java

Esta es la actividad que permitirá registrar y guardar todas las posiciones que se irán generando durante el transcurso de una ruta. La interfaz la tenemos definida en `activity_tracking.xml` en el que tenemos:

- `TextViews` para visualizar la velocidad, latitud, longitud y altura en el momento.
- Un cronometro con el tiempo que se lleva de ruta, lo creamos con un widget `Chronometer`.
- Un botón “stop” para parar el registro de posiciones y si se tiene conectividad proceder al envío de los datos al servidor.

Al iniciar la activity se procede a inicializar todas las variables, las más destacadas:

```
final GlobalClass globalVariable = (GlobalClass) getApplicationContext();
```



```
url_inserir="http://" + globalVariable.getServeridor() + "/biketracking/inserir_posiciones.php"
url_max="http://" + globalVariable.getServeridor() + "/biketracking/max_rutas.php";
url_nueva_ruta="http://" + globalVariable.getServeridor() + "/biketracking/nueva_ruta.php";
```

que recuperando la IP de nuestro servidor web generamos la dirección a donde hacer las peticiones.

```
private ArrayList <Posicion> valores ;
```

Este será nuestro ArrayList de objetos tipo posición donde guardaremos todas las posiciones que vayamos recogiendo durante la ruta.

Contamos con un método que se inicia después de inicializar todas las variables:

```
empezar_localizacion ()
```

Este método nos crea un LocationManager que junto con un LocationListener podremos iniciar nuestro registro de posiciones. El LocationManager lo inicializamos con nuestros parámetros de tiempo de actualización mínimo entre registros de posición con esta llamada:

```
locManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
tiempoactualizacion, 20, locListener);
```

Cada vez que tengamos un cambio de posición que detecte nuestro GPS se ejecutara nuestro código dentro del método:

```
public void onLocationChanged(Location location) { nuestro codigo }
```

Que será cuando guardamos la posición actual en nuestro ArrayList de posiciones:

```
Posicion a_guardar = new Posicion
(location.getLatitude(),location.getLongitude(),tiempo,
location.getAltitude());
        valores.add(a_guardar);
```

y la de actualizar los valores de la pantalla al último valor obtenido.

```
mostrarPosicion (location);
```

En el que lo que haremos es setText con su valor correspondiente de todos los TextView.



Y de paso comprobaremos si la velocidad es la máxima hasta el momento y la guardaremos si fuera así: `if (1.getSpeed()>vmax ) {vmax= 1.getSpeed();}`

En cuanto se presione el botón de STOP, se procederá a empezar a intentar guardar los datos, primeramente comprobaremos si tenemos conexión a internet con el método: `isNetworkAvailable()` que nos devuelve con un booleano si hay o no conectividad.

Si no la hay tiraremos un dialogo de aviso con el método: `mostrarVentanaAlerta();` avisando de que no hay conexión a internet y cambiando el texto del botón de parar a Recargar datos para que cuando el usuario quiera probar de volver a enviar los datos puede hacerlo las veces necesarias.

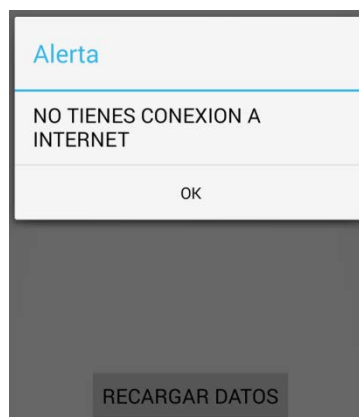


Figura 61. Dialogo aviso de internet deshabilitado.

En caso de tener conexión a internet iniciaremos la tarea asíncrona dedicada a enviar los datos: `new inserir().execute();`

Este método lo tenemos declarado con la siguiente cabecera:

```
class inserir extends AsyncTask <Void, Integer, Boolean>
```

Donde tendremos diferentes partes dentro de ella:

- **protected void** `onPreExecute()` → es la parte que se ejecuta primero donde lanzaremos un dialogo comunicando que se están enviando los datos al servidor.
- **protected** `Boolean doInBackground(Void... params)` → Aquí va nuestro código a ejecutar:

Crearemos primero una ruta nueva:

```
JSONObject json2 = jsonParser.makeHttpRequest(url_nueva_ruta, "POST",  
param);
```



Pasando el parámetro param todos los valores de la ruta a través de parejas de valores

```
List<NameValuePair> param = new ArrayList<NameValuePair>();  
param.add(new BasicNameValuePair("fecha", currentDate) ) etc...
```

Y luego crearemos el JSON de todas las posiciones que tenemos:

```
JSONArray array = new JSONArray();  
// recorreremos todo el array para crear el json con lo  
datos  
for (Posicion x:valores)  
{  
    JSONObject js = new JSONObject();  
    try {  
        js.put("longitud", x.getLongitud());  
        js.put("latitud", x.getLatitud());  
        js.put("tiempo", x.getTiempo());  
    }  
    catch (JSONException e)  
    {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
    // Añadimos el json al json array  
    array.put(js);  
}
```

Y lo enviaremos como con la creación de una nueva ruta a través de parejas de

BasicNameValuePair con la etiqueta de json como se muestra:

```
paramms.add(new BasicNameValuePair("json", array.toString() ) );  
  
JSONObject json = jsonParser.makeHttpRequest(url_inserir, "POST",  
paramms);
```

La petición la realizamos a través del método makeHttpRequest que implementa la clase JSONParser de nuestro objeto jsonParser.

Comprobaremos al acabar que todo ha salido correctamente recogiendo la variable success de retorno.

- **protected void** onPostExecute(Boolean result): Este ya es el código cuando ha acabado la inserción de las posiciones. Cerraremos el dialogo iniciado al postexecute y comprobaremos al acabar que todo ha salido correctamente recogiendo la variable success de retorno e informando del éxito a través de un Toast ( mensaje informativo ).



En la clase tracking tenemos también un método **public double** calculaAltitud () que nos sirve para calcular la altitud ganada durante toda la ruta. Lo que hace este método es recorrer todas las posiciones y acumulando la diferencia de las altitudes de cada par mediante un foreach. Este método se llamara cada vez que se cree una nueva ruta.

#### 5.3.1.5.5 Activity Lista.java

Esta es la activity que nos mostrara las rutas tanto privadas como privadas de un usuario. Como ya se comentó en la activity de menus se pasa un parámetro para saber si hay que hacer un listado de rutas privada o publicas dependiendo del botón del que vengamos ( modo privado/publico ). El fichero XML que tiene definido la interfaz de esta actividad es el activity\_lista, el cual contiene un ListView que es el que utilizaremos para listar y poder seleccionar las rutas.

Las rutas las guardaremos en un ArrayList de objetos ruta: `ArrayList <Ruta> rutasList;`

Como ya se ha comentado en la actividad tracking también se recupera la IP de la clase GlobalClass para generar la dirección para hacer las peticiones.

Para recuperar la rutas que necesitamos utilizaremos una tarea asíncrona: **new** cargaRuta().execute(); En esta tarea asíncrona: **class** cargaRuta **extends** AsyncTask <String, String, String> tenemos las siguientes partes:

- **protected void** onPreExecute()→ crearemos un dialogo informando de que se va a contactar con el servidor.

```
pDialog = new ProgressDialog(Lista.this);  
pDialog.setMessage("Cargando datos del servidor ....");
```

- **protected** String doInBackground(String... args)→ En esta parte es donde haremos la petición al servidor con el siguiente código:

```
List<NameValuePair> params = new ArrayList<NameValuePair>();  
params.add(new BasicNameValuePair("usuario", usuario ));  
params.add(new BasicNameValuePair("modo", modo ));  
  
// recogemos el resultado en JSON de la pagina php  
  
JSONObject json = jParser.makeHttpRequest(url_Lista_rutas,  
"POST", params);
```



Este es el mismo método que hemos utilizado en la actividad de tracking y que utilizaremos en todas las peticiones dentro de una tarea asíncrona. Nos valemos de un objeto de la clase JSONParser para enviar la petición y recoger los datos en un JSON. Comprobamos que hay resultados: si no hubiese se informa al usuario y se retorna a la pantalla de menus, y en caso de haber resultados se recorrerá todos los resultados y se guardaran en el ArrayList de rutas.

- **protected void** onPostExecute(String file\_url) → En esta parte será donde recorramos el ArrayList de rutas que hemos obtenido en la anterior parte y crearemos un vector de rutas para introducirlas en un AdapterView de ViewHolders de cada una de las rutas para poderlas listar en el ListView y hacer que se pueda seleccionar cada una de ellas y lanzar la activity de MenuRuta.

Para ello se ha creado una clase AdaptadorRuta **extends** ArrayAdapter<Ruta> que nos permite utilizar un fichero xml (lista.xml) para definir la interfaz cada uno de los ViewHolder de las rutas que muestra los datos de la ruta.

En **Lista.xml** definimos 6 TextView para mostrar de cada ruta: fecha, descripción, hora inicial y final, si es publica y el autor.

#### 5.3.1.5.6 Activity MenuRuta.java

Esta es la actividad a la que llegamos cuando hemos seleccionado alguna ruta del ListView de la activity Lista. El fichero que define la interfaz gráfica es el “activity\_menuruta.xml”

Tendremos los botones con sus respectivas acciones :

- **Ver mapa:** en caso de clicar nos enviaría la actividad Mapa para mostrar la ruta.
- **Detalles de la ruta:** al acceder nos enviaría a la actividad DetallesRuta.
- **Eliminar Ruta:** ejecutaremos una tarea asíncrona para eliminar la ruta.
- **Cambiar descripción:** nos enviará a la actividad que nos permite cambiar la descripción CambiaDescripcion.
- **Cambiar privacidad:** ejecutaremos también una tarea asíncrona para cambiar el atributo de privacidad.





Siempre comprobaremos si venimos de un usuario no registrado o registrado para permitir la ejecución de las tres acciones de modificación o eliminación.

Tendremos dos tareas asíncronas implementadas en dos métodos diferentes:

1.- borrarRuta → **class** borraRuta **extends** AsyncTask <String, String, String>

Nos permitirá borrar la ruta en cuestión, tendremos su identificador porque nos lo ha pasado la actividad anterior (Lista) a través de parámetro, la petición hecha a través del JSONParser : JSONObject json = **jParser**.makeHttpRequest(*url\_borrar*, "POST", params); Donde el parámetro es el identificador de la ruta.

Hemos implementado una medida de seguridad ante un borrado accidental. Un dialogo nos pedirá confirmar el borrado, lo llamaremos con el método **public void** seguroBorrar () que con dos botones de SI o NO confirmaremos la acción.

2.- cambiaPrivacidad→ **class** cambiaPrivacidad **extends** AsyncTask <String, String,

String> En esta tarea asíncrona haremos la petición a través de nuestra ya conocida clase JSONParser. Mostraremos un dialogo indicando que se está contactando con el servidor en el PreExecute (), y luego haremos la petición con el método httpRequest de nuestro JSONParser. Al final cerraremos el dialogo y indicaremos que todo ha ido bien si tenemos la confirmación con la variable *success* vía retorno.

#### 5.3.1.5.7 DetallesRuta.java

En esta actividad mostraremos los datos de la ruta, en el fichero activity\_detalle.xml está definida la interfaz visual de la actividad, que consta de 6 TextView para mostrar:

- Distancia total
- Velocidad media
- Tiempo total
- Si es publica
- Velocidad máxima
- Altitud ganada



Actualizaremos todos los campos de texto con los valores correspondientes menos uno.

El valor que nos falta ( distancia total ) lo obtendremos ejecutando una tarea

asíncrona para hacer la petición al servidor web. La cabecera es: **class** cargaValores

**extends** AsyncTask <String, String, String>

Como en todas las anteriores actividades mostraremos un dialogo indicando que está contactando con el servidor en la tarea asíncrona. También usaremos un JSONParser para hacer la petición mediante un makeHttpRequest a la dirección del servidor web para el cálculo de distancia (calculo\_distancia.php).

Hemos implementado un método para convertir el tiempo en milisegundos a un formato de hora:minuto:segundo : **public** String convierteTiempo (String e )

```
long segundos = (Long.parseLong(e))/1000;
long h = segundos / 3600;
long m = (segundos - h * 3600)/60;
long s = segundos - 3600 * h - 60 * m;
return new String
(String.valueOf(h)+":"+String.valueOf(m)+":"+String.valueOf(s));
```

#### 5.3.1.5.8 CambiaDescripcion.java

Esta es la actividad que utilizaremos para cambiar la descripción de una ruta. Usa la interfaz guardada en fichero “activity\_cambiadesc.xml” que consta de:

- Descripción actual: mostrada con un TextView
- Descripción a cambiar: a insertar por el usuario en un EditText
- Botón de aceptar: ejecutara una tarea asíncrona para cambiar la descripción.
- Botón de cancelar: volveremos a la actividad anterior con un finish ();

La tarea asíncrona encargada de cambiar la descripción de la ruta es: **class** cambiaDescripcion **extends** AsyncTask <String, String, String>

Seguimos el mismo patrón que en las peticiones anteriores al servidor web: utilizando un JSONParser. Pasaremos como parámetros el identificador de la ruta y la descripción nueva a cambiar. Controlaremos la buena realización de la modificación en la parte de onPostExecute informando de ello o no en un Toast informativo:



```
Toast.makeText(getApplicationContext(), "Cambio de descripcion exitosa",  
Toast.LENGTH_LONG).show();
```

### 5.3.1.5.9 Mapa.java

Esta actividad implementa la API de google maps para mostrar gráficamente el recorrido de una ruta. El archivo “activity\_mapa.xml” contiene la interfaz de la actividad ( un fragment ). Sabremos el identificador de la ruta porque nos la ha pasado la actividad anterior.

Hemos implementado diferentes métodos para llevar a cabo las diferentes acciones dentro de la actividad:

1.- **private void** inicializaMapa() → En este método inicializamos el mapa a mostrar, que lo haremos al inicio y en el estado de ciclo de vida onResume ().

```
if (googleMap == null)  
{  
    googleMap = ((MapFragment)  
getFragmentManager().findFragmentById(R.id.map)).getMap();  
    // Miramos si todo ha ido bien  
  
    if (googleMap == null)  
    {  
        Toast.makeText(getApplicationContext(), "No ha sido posible crear el  
mapa!!", Toast.LENGTH_SHORT).show();  
    }  
}
```

Y en caso que no se pueda mostrar lanzaremos un mensaje informando de ello.

2.- Tarea asíncrona para obtener las posiciones cargaDatos () con la siguiente cabecera:

```
class cargaDatosGps extends AsyncTask <String, String, String>
```

Dentro de este método obtendremos las posiciones de la ruta a mostrar, como en las demás actividades lo haremos a través de una petición de JSONParser pasando el identificador de ruta como parámetro. Se recorrerán todos los resultados y se guardaran en un ArrayList de posiciones: ArrayList <Posicion> **datosgps**;

En la sección de la tarea asíncrona **onPostExecute** llamaremos al método **marcalaruta()** para crear la visualización gráfica del recorrido.

3.- **private void** marcaruta() → En este método cargaremos las posiciones a un



ArrayList de objetos LatLng para crear el recorrido grafico de la siguiente manera:

```
ArrayList <LatLng> track = new ArrayList <LatLng> ();  
// recorremos todo el array de posiciones  
for (Posicion x:datosgps)  
{  
    track.add(new LatLng(x.getLatitud(),x.getLongitud()));  
}  
// creamos el camino con el array de las posiciones gps  
googleMap.addPolyline(new PolylineOptions().geodesic(true).addAll(track)) ;
```

Esto nos creara una línea del recorrido de nuestra ruta pasando por todas las posiciones GPS.

Luego añadiremos dos marcadores para indicar el inicio y fin de la ruta:

```
googleMap.addMarker(inicio);  
googleMap.addMarker(finall);
```

### 5.3.2 Implementación en servidor web

La implementación en el servidor web consta de **11** ficheros PHP guardados en una carpeta llamada **biketracking**. Hemos seguido las pautas de lo que en su momento valoramos en la etapa de diseño. Los ficheros son los siguientes:

- 1 db\_config.php
- 2 db\_connect.php
- 3 borra\_ruta.php
- 4 calculo\_distancia.php
- 5 cambia\_descripcion.php
- 6 cambia\_privacidad.php
- 7 get\_posiciones.php
- 8 insertar\_posiciones.php
- 9 lista\_rutas.php
- 10 max\_rutas.php
- 11 nueva\_ruta.php

Se controlara el correcto paso de parámetros de entrada por POST en cada de uno de ellos utilizando la función de PHP **isset**: `isset($_POST['variable_de_entrada'])` y sacando su carencia por mensaje de salida a través de una cadena de texto llamada **mensaje** y un variable que tomara valores 0 o 1 si se ha realizado la operación bien llamada **success** también en el String de retorno. Se comentara más adelante lo casos posibles en cada uno de los scripts.



Cabe mencionar también que se han usado los métodos que ya nos proporciona PHP:

`json_decode ($cadena)` y `json_encode ($array)` para poder trabajar con las cadenas de texto y arrays para manipularlos a nuestra conveniencia.

A continuación iremos comentando cada uno de ellos:

### 5.3.2.1 *db\_config.php*

Este archivo PHP nos aporta toda la información referente a los datos de conexión a nuestra base de datos del servidor mysql. Es un archivo que está separado porque si en cualquier momento queremos cambiar algún valor de conexión/servidor lo podamos hacer de una manera rápida y limpia sin tener que modificar nada más.

```
<?php

/*
 *-----
 * DATOS DE LA CONEXION A LA BD
 *-----
 */

define('DB_USER', "root"); // usuario
define('DB_PASSWORD', "*****"); // contraseña
define('DB_DATABASE', "biketracking"); // nombre de la base de datos
define('DB_SERVER', "localhost"); // nombre servidor

?>
```

Se definen los siguientes parámetros.

- Nombre del usuario
- Contraseña del usuario
- Nombre de la base de datos
- Nombre del servidor donde está la base de datos

**db\_connect.php** recogerá estos parámetros con un **require\_once ("db\_config.php")**



### 5.3.2.2 db\_connect.php

Este script en php contiene la clase DB\_CONNECT que nos servirá para conectarse a una base de datos con los parámetros que le facilita el anterior script php **db\_config.php**.

Tiene las siguientes funciones:

- function \_\_construct() → función del constructor
- function \_\_destruct() → función del destructor
- function connect() → función que nos facilita la conexión retornando un cursor
- function close() → función para cerrar la conexión.

Como ya se comentó en el apartado del fichero **db\_config.php**, este es requerido para recuperar los parámetros de conexión que se utilizan en la clase DB\_CONNECT:

DB\_USER, DB\_PASSWORD, DB\_DATABASE, DB\_SERVER.

Este script será utilizado en cada uno de los demás scripts para poder establecer la conexión a la base de datos utilizando un **require\_once ('db\_connect.php')**.

### 5.3.2.3 borra\_ruta.php

Este el script PHP que nos permitirá eliminar una ruta. Como ya se especificó en el diseño este script recibirá por parámetro POST sólo el identificador de la ruta a borrar. Al borrar la ruta se borran automáticamente todas las posiciones que están asociada a ella gracias a que en su momento se especificó en diseño un *constraint* del tipo **ON DELETE CASCADE** en la tabla de las posiciones referenciado a la clave ajena **id\_ruta**.

Hay **tres** salidas posibles de mensajes de retorno:

```
{"success":0,"mensaje":"Falta el parámetro de entrada"}
```

En un primer momento verificamos que se no está llegando un parámetro y en caso contrario indicáramos que nos falta el parámetro. **if** (isset(\$\_POST['id']))



```
{"success":1,"mensaje":"Borrado con éxito"}
```

Con este retorno sabremos que la operación se ha realizado sin ningún problema.

Sabremos que se ha realizado con éxito porque al ejecutar la sentencia SQL nos retornara un resource y en caso contrario no.

```
{"success":0,"mensaje":"Falta el parámetro de entrada"}
```

En este caso nos indicaría que nos ha llegado el parámetro pero algo no ha salido bien al intentar borrar la ruta.

La sentencia SQL utilizada para la eliminación de la ruta es:

```
DELETE FROM rutas WHERE id_ruta = $id"
```

#### 5.3.2.4 *calculo\_distancia.php*

Este es el script en PHP que se encargara de calcular la distancia total de nuestra ruta.

Le llega un parámetro que es el identificador de la ruta de la que queremos saber la distancia y en caso de que no lo tuviéramos directamente nos indicaría a través del retorno que nos falta de este valor:

```
{"success":0,"mensaje":"Falta el parametro de entrada"}
```

Y en caso de no haber problemas nos sacara un mensaje de salida con el valor de la distancia **en kms** de la ruta del tipo:

```
{"success":1,"distancia":"valordistancia_en_kms"}
```

El proceso que seguimos es el siguiente:

- Recuperamos todas las posiciones de la ruta.
- Recorremos el array de posiciones y en cada iteración del bucle del recorrido llamamos a la función `VincentyDistance($lat1,$lat2,$lon1,$lon2)` con las coordenadas de GPS de los dos posiciones implicadas. Este retorno lo vamos acumulando en la variable **distancia** que es la retornaremos al final.



El algoritmo de cálculo de la distancia entre dos posiciones GPS es bastante complejo.

#### 5.3.2.5 cambia\_descripcion.php

Este es el script PHP encargado de la modificación de la descripción de una ruta. Se le pasan por parámetro la nueva descripción y el identificador de ruta. También seguiremos con el mismo modelo de retorno mensaje y variable success informando de cómo ha ido la operación, teniendo estos 3 casos posibles:

- Cuando falte algún parámetro de entrada:

```
{"success":0,"mensaje":"Faltan datos de entrada"}
```

- Cuando no se ha efectuado la operación correctamente:

```
{"success":0,"mensaje":"No se ha modificado correctamente"}
```

- O cuando la operación ha sido realizada sin ningún problema:

```
{"success":1,"mensaje":"Descripcion cambiada con exito"}
```

La sentencia SQL utilizada para hacer este cambio es:

```
UPDATE rutas SET descripcion = '$descripcion' WHERE id_ruta = $id
```

#### 5.3.2.6 cambia\_privacidad.php

Este es el script PHP encargado de la modificación de la privacidad de una ruta. Se le pasa por parámetro el nuevo estado y el identificador de ruta afectada. Seguimos con el mismo modelo de retorno de mensaje reportando cómo ha ido la operación, teniendo estos 3 casos posibles:

- Cuando falte algún parámetro de entrada:

```
{"success":0,"mensaje":"Faltan datos de entrada"}
```

- Cuando no se ha efectuado la operación correctamente:

```
{"success":0,"mensaje":"No se ha modificado correctamente"}
```

- O cuando la operación ha sido realizada sin ningún problema:





```
{"success":1,"mensaje":"Modificacion con exito"}
```

La sentencia SQL utilizada para hacer este cambio es:

```
UPDATE rutas SET espublica = '$estado' WHERE id_ruta = $id
```

### 5.3.2.7 get\_posiciones.php

En este código PHP accederemos a la base datos para recuperar todas las posiciones de una ruta en un Array pasando por parámetro el identificador de la ruta en cuestión. Como en los anteriores scripts sacamos a través del retorno junto a las posiciones como ha ido el proceso:

- Cuando falte algún parámetro de entrada:

```
{"success":0,"mensaje":"Falta el parámetro de entrada"}
```

- Cuando no se ha efectuado la operación correctamente:

```
{"success":0,"mensaje":"No se han encontrado posiciones"}
```

- O cuando la operación ha sido realizada sin ningún problema:

```
{"success":1,"mensaje":"Todo OK!" .... array de posiciones [...]}
```

La sentencia SQL utilizada:

```
SELECT * FROM datosgps WHERE id_ruta = $id
```

Cuando tenemos los registros de las posiciones de la tabla datosgps los recorremos todos mediante un **while** y vamos guardando los valores de cada posición en un array de posiciones. Al final se retorna este array con todas las posiciones de la ruta.

### 5.3.2.8 insertar\_posiciones.php

En este código PHP nos proporciona la tarea de introducir todas las posiciones de una ruta. Recibe por parámetros POST el identificador de la ruta , un JSON que contiene todas las posiciones de la misma a guardar y el número de posiciones a guardar. Este número de posiciones a guardar nos servirá para comprobar si realmente se han introducido todas las



posiciones que nos han llegado. Tendremos como en los anteriores scripts diferentes mensajes de salida y una variable para controlar si todo ha ido bien (success):

- Si todas las posiciones se han introducido correctamente:

```
{"success":1,"mensaje":"Todo OK!!"}
```

- Si ha habido alguna posición que no se ha introducido bien:

```
{"success":0,"mensaje":"Error en la inserción"}
```

- Cuando hay algún parámetro que nos falta:

```
{"success":0,"mensaje":"Faltan datos de entrada"}
```

Al parámetro de entrada JSON que contiene todas las posiciones le aplicamos la función **json\_decode** para poder acceder a los datos que contiene el json de posiciones mediante un array. Este array lo iremos recorriendo para recoger las variables de la posición a guardar y la introduciremos con la siguiente sentencia mysql:

```
INSERT INTO datosgps (latitud, longitud, tiempo, id_ruta) VALUES('$latitud',  
'$longitud', '$tiempo', '$id')
```

Tenemos una variable que nos controla las posiciones que hemos introducido correctamente que es la que comprobaremos al final del recorrido para saber si todas las posiciones han sido introducidas con éxito.

### 5.3.2.9 lista\_rutas.php

Este script en PHP nos facilitara todas las rutas privadas de un usuario o todas las rutas públicas de la base de datos de la ruta. Dos parámetros nos llegaran por POST:

- Identificador del usuario
- Modo de consulta: privado o público.

En modo público se devolverán todas las rutas públicas y en modo privado se devolverán todas las rutas del usuario que se nos pida.



Tendremos 3 diferentes casos de salida:

- La operación ha sido realizada con éxito:

```
{"success":1,"rutas": array de rutas [] ..... }
```

- En caso de faltar algún parámetro de entrada:

```
{"success":0,"mensaje":"Faltan datos de entrada"}
```

- No tener ningún resultado:

```
{"success":0,"mensaje":"No se encuentra ninguna ruta"}
```

Tenemos **dos** tipos de sentencias mysql de consulta a la base de datos.

- Modo privado:

```
SELECT * FROM rutas WHERE usuario = $usuario
```

- Modo público:

```
SELECT * FROM rutas WHERE espublica = 1
```

Mediante un **while** recorreremos los resultados de las rutas y nos guardamos los datos cada una de ellas en un array de rutas para ser devuelto al finalizar junto con la cadena de texto **mensaje** y la variable **success**.

### 5.3.2.10 max\_rutas.php

Este script no forma parte del diseño ya que se ha implementado durante la implementación. Al crear una ruta nueva nos hace falta saber que id\_ruta ha cogido la nueva ruta para después insertar todas las posiciones a esta nueva ruta; es por ello que se hizo este código que lo que hace es simplemente devolver el ultimo id\_ruta creado. Esto se realizara después de crear una ruta nueva. No hay paso de parámetros vía POST .

El código es el siguiente:

La sentencia SQL utilizada: 

```
SELECT max(id_ruta) from rutas
```



El retorno es del tipo:

```
{"numero":"xxx","success":1}
```

### 5.3.2.11 nueva\_ruta.php

Este script es el que introduce una nueva ruta en la base de datos. Como se comentó ya en la parte de diseño se le pasaran por parámetro todos los datos de la ruta:

- Usuario que ha hecho la ruta.
- Nombre del usuario
- Fecha de la ruta.
- Hora de inicio
- Hora final
- Tiempo total.
- Velocidad máximo alcanzada en la ruta.
- Ganancia de altitud

Compartimos la misma manera de mensajes de salida que en los anteriores scripts.

- Si la ruta se ha introducido correctamente:

```
{"success":1,"mensaje":" Todo OK! Ruta creada"}
```

- Si ha habido algún problema:

```
{"success":0,"mensaje":" Error: no se ha introducido la nueva ruta"}
```

- Cuando hay algún parámetro que nos falta:

```
{"success":0,"mensaje":"Faltan datos de entrada"}
```

La sentencia SQL para introducir la ruta es:

```
INSERT INTO `rutas` (`id_ruta`, `descripcion`, `fecha`, `hora_inicio`,  
`hora_final`, `usuario`, `espublica`, `nombreautor`, `tiempo`, `velocidadmax`, `altitudganada` ) VALUES ('', 'Aun sin descripcion', '$fecha', '$horainicio',  
'$horafinal', '$usuario', 0, '$nombre', '$tiempo', '$velocidadmax', '$ganancia'  
)
```



## 6.- Evaluación

Este proyecto se ha ido haciendo de manera modular, por lo que todas las partes han podido ser probadas, evaluadas y rectificadas por separado y paso por paso.

Al principio se utilizó mucho la máquina virtual que nos ofrece el Android SDK, pero más adelante con la introducción de las APIS de google se tuvo que pasar a utilizar dispositivos físicos para una buena evaluación de la aplicación. El dispositivo que se ha utilizado para la mayoría de las pruebas es un **Samsung Galaxy S3** GT-I9300 con Android 4.3 Jelly Bean. ( un dispositivo muy popular hoy en día ).

Se han probado y solucionado los siguientes supuestos:

- Quedarse sin conexión a internet al enviar datos de una ruta recién hecha. ( creación de un diálogo avisando y dando la posibilidad de volver a enviar los datos las veces necesarias hasta que se tenga conectividad)
- Listado de rutas sin ningún resultado. ( se avisa al usuario de esto )
- Uso de varios usuarios y compartición de rutas entre ellos.
- Apertura del mapa de google maps sin ninguna posición en la ruta. ( se verifica y se avisa al usuario de ello ).
- Cambio del tiempo de actualización del GPS a un valor mínimo de 1000 ms para comprobar el correcto funcionamiento cuando hay alta carga de datos a manipular.



## 7.- Costes

El coste de nuestro proyecto es bastante bajo, ya que hemos buscado un servidor muy barato y eficiente. Los gastos son los siguientes:

- **Raspberry PI:** El precio es de 36.95€ que si le añadimos un cargador para la alimentación ( unos 3.5 €) y la tarjeta SD de memoria donde alojaremos el SO y los datos ( 20 €) nos sale un total de **60.45 €** para costear el servidor web.
- Registrarse como desarrollador en **Google Play**: para tener nuestra aplicación disponible en el google play y poder cobrar de la descarga de esta nos hará falta este registro. El precio es de unos 25 \$, que pasados a euros son unos **18,20 €** Es un pago único ya que no es una cuota periódica por lo que puede ser considerado un gasto inicial.
- Conexión a internet: el gasto de conexión a internet de nuestro servidor lo consideramos **0 €** ya que podemos aprovechar cualquier conexión ADSL / fibra que ya tengamos disponible.
- Aplicativos: el gasto de las aplicaciones a utilizar los consideraremos también cero ya que hemos utilizado un IDE de programación con licencias de software libre.

En conclusión hemos tenido un **gasto inicial** de unos **78.25 €** Y como gastos variables tenemos unos 1,10 €/mes aproximadamente ( **13.2 € anuales** ) que corresponden al gasto de electricidad del Raspberry PI que estará 24 horas en servicio los 365 días del año. Vemos que este coste en el servidor web es bastante reducido ya que cualquier **hosting web** nos saldría bastante más caro que nuestro Raspberry ( en 1&1 en la modalidad más barata unos 3 €/mes ).



## 8.- Planificación temporal

### 8.1 Desglose por fases del proyecto

En la siguiente tabla se muestra el tiempo dedicado a cada tarea, junto con la fecha inicial y final.

Tarea	Duración	Fecha inicial	Fecha final
Preparación aplicativos	1 día	03/1/2014	04/1/2014
Leer documentación previa	5 días	03/1/2014	08/1/2014
Pruebas iniciales	6 días	09/1/2014	15/1/2014
Planificación y análisis inicial	2 días	16/1/2014	18/1/2014
Diseño aplicación Android	29 días	20/1/2014	16/2/2014
Diseño webservices	12 días	06/2/2014	18/2/2014
Diseño de la base de datos	14 días	04/2/2014	18/2/2014
Implementación aplicación Android	37 días	06/2/2014	18/2/2014
Implementación webservices	37 días	26/2/2014	04/4/2014
Preparación servidor web	2 días	06/3/2014	08/3/2014
Migración datos a nuevo servidor	1 día	10/3/2014	11/3/2014
Puesta en marcha nuevo servidor	1 día	12/3/2014	13/3/2014
Pruebas de la aplicación	21 días	27/3/2014	17/4/2014
Documentación del desarrollo	12 días	14/4/2014	26/4/2014

A continuación se mostrará el diagrama de Gantt visual con todos los datos.

## 8.2 Diagrama de Gantt de la planificación

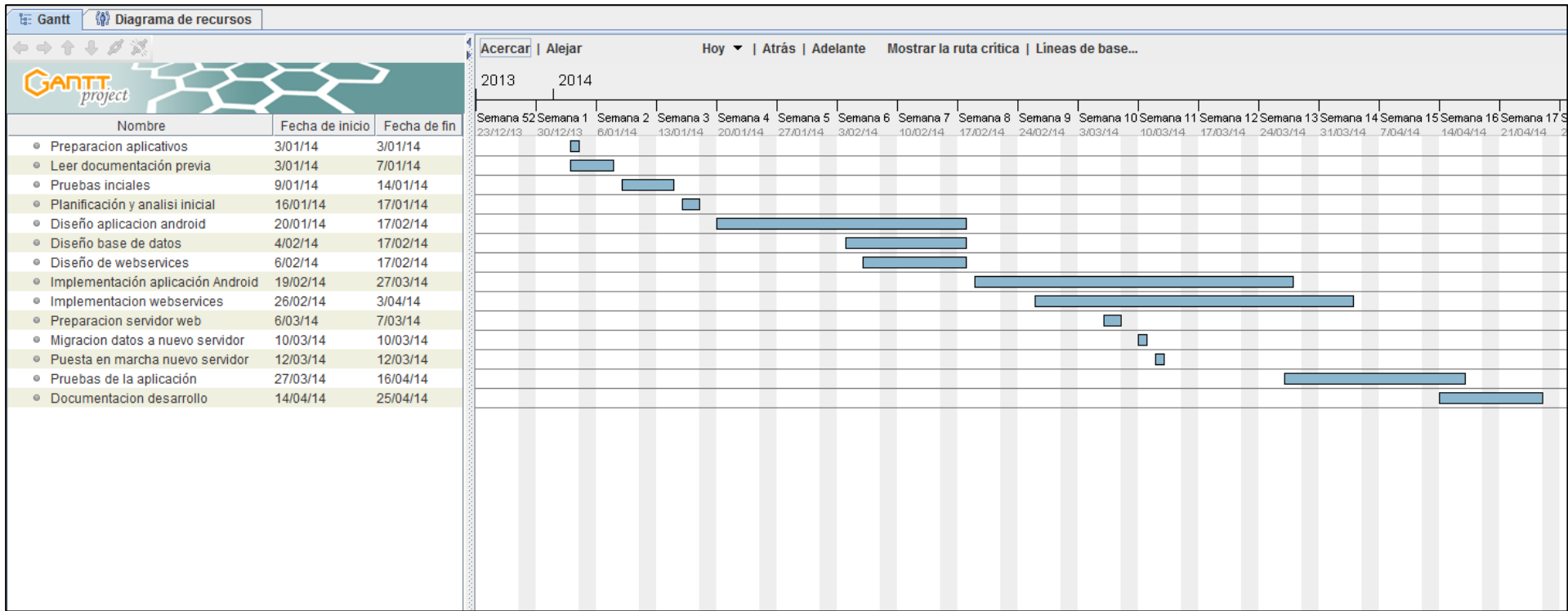


Figura 63. Diagrama Gantt planificación.



## 9.- Conclusiones

Partiendo de la base de que se comenzaba con unos conocimientos de programación en Android desde cero ha resultado muy satisfactorio llegar a hacer una aplicación totalmente funcional.

Al principio fue muy duro ya que tuve que aprender cosas muy básicas pero que eran imprescindibles para poder avanzar. Con el tiempo y mil pruebas después se pudo entender bastantes conceptos de la programación Android. Gracias a mucha gente que escriben en foros y a las múltiples páginas web dedicadas a Android se puede llegar a muchas soluciones a los varios problemas que surgen durante la programación de una aplicación.

Ha sido muy interesante porque durante todo el proceso han surgido varios tipos de problemas y retos. Uno de ellos fue el de buscar un servidor web eficiente y barato, y que se solucionó configurando un servidor web en un Raspberry pi 2. Fue una solución que se hizo sobre la marcha y que se ha comprobado que es una solución más que válida en términos de eficiencia. ( sólo gasta unos 3.5 w)

También quisiera comentar que ha sido un proyecto muy variado donde se han tocado bastantes temas diferentes en informática ya desde montar un servidor web, configurar un router para la conexión, programación ya tanto en php y java, como configuraciones en SO Linux ( raspbian ).

En conclusión: se ha sacado muy buen provecho de todo el proceso que ha conllevado hacer el proyecto: conocimientos de programación en Android, diseño y funcionamiento de base datos ...



## **10.- Recursos utilizados**

### **10.1 Bibliografía**

- Mark L. Murphy - The Busy Coder's Guide to Android Development, Version 4.2, Sep 2012
- Professional Android 4 Application Development by Reto Meier
- Learning PHP, MySQL, JavaScript, and CSS, Second Edition by Robin Nixon.
- Manual Programación Android ver 3.0, Salvador Gómez Oliver  
([WWW.SGOLIVER.NET](http://WWW.SGOLIVER.NET))

### **10.2 Páginas web**

<http://www.sgoliver.net>

<http://stackoverflow.com>

<http://www.raspberrypi.org>

<http://www.androidhive.info>

<https://developers.google.com/android/>

<http://geekytheory.com/json-i-que-es-y-para-que-sirve-json/>

<http://jsperf.com/vincenty-vs-haversine-distance-calculations>

<http://www.movable-type.co.uk/scripts/latlong-vincenty.html>

<http://www.programacion.com.py/moviles/android/utilizar-mysql-en-android-con-jdbc>

<http://ieeesb-uniovi.es/talleres-charlas/linux/raspberrypi/>

[http://www.unavco.org/edu\\_outreach/tutorial/geoidcorr.html](http://www.unavco.org/edu_outreach/tutorial/geoidcorr.html)

<https://developers.google.com/maps/documentation/android/>



<https://developers.google.com/+/mobile/android/getting-started>

[http://www.tutorialspoint.com/json/json\\_php\\_example.htm](http://www.tutorialspoint.com/json/json_php_example.htm)

<http://www.deif.org/blog/calcular-distancia-a-partir-de-datos-gps/>

<http://www.latamreview.com/noticias/ver/6686/las-seis-mejores-aplicaciones-para-disfrutar-en-bicicleta>

<http://www.javacodegeeks.com/2013/09/android-viewholder-pattern-example.html>

### **10.3 Software**

- Eclipse IDE SDK Version: 4.2.2 Build id: M20130204-1200.
- ADT de Android para Eclipse
- Android SDK
- Notepad++ version 6.5.5.
- XAMPP para windows.
- Adobe Photoshop CS5 Extended.
- Microsoft Office 2010 y Visio 2010.
- MySQL Workbench 5.2.47 CE.

### **10.3 Hardware**

- Samsung galaxy S3 GT-I9300 con Android 4.3 Jelly Bean
- Samsung galaxy S5 GT-I9500 con Android 4.4.2 Kit Kat
- HTC wildfire S

