UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"DESARROLLO DE UNA APLICACIÓN ANDROID PARA GESTIÓN DE RUTAS"

TRABAJO FIN DE GRADO

Diciembre – 2019

AUTOR: Darío Barberá Madrid

DIRECTOR: César Fernández Peris

UNIVERSIDAD MIGUEL HERNÁNDEZ DE ELCHE

ESCUELA POLITÉCNICA SUPERIOR DE ELCHE

GRADO EN INGENIERÍA MECÁNICA

"DESARROLLO DE UNA APLICACIÓN ANDROID PARA GESTIÓN DE RUTAS"

TRABAJO FIN DE GRADO

Diciembre - 2019

AUTOR: Darío Barberá Madrid

DIRECTOR: César Fernández Peris

AGRADECIMIENTOS

En primer lugar, debo nombrar sin ninguna duda a esas personas que nunca me han fallado, y que por supuesto nunca lo harán a lo largo de mi vida. Esa gran familia que tanto me ha apoyado en mis buenos y no tan buenos momentos, haciéndome ver que después de esas duras etapas de presión e incertidumbre, siempre habrá algo por lo que luchar y seguir adelante.

A destacar, mis padres y mi hermanito, a los que tanto adoro y tanto les debo, gracias a ellos todo a resultado ser un poco más llevadero en este largo camino.

Y a mis abuelos, que tan solo pueden recibirme siempre con una gran sonrisa, y a los cuales estaré tremendamente orgulloso de darles esta increíble noticia, como es haber finalizado mi carrera de Ingeniería.

Por otra parte, todo ese círculo de personas que me han acompañado, tanto dentro como fuera de la universidad, han sido realmente valiosas en mi vida de estudiante.

Por un lado, mi gente de siempre, amigos que me ayudaron y lo siguen haciendo a sentirme lleno de ganas para continuar dándolo todo por superar objetivos, pudiendo evadirme con ellos por unas horas, en aquellas tardes donde sentía que si no me tomaba un respiro, mi cabeza explotaría.

Y a la vez, gracias a la universidad, puedo agradecerle aún a más gente el hecho de que ahora mismo esté finalizando este proyecto de mi vida. Tantos y tantos alumnos hemos podido empatizar unos con otros, y ayudarnos mutuamente para superar cada asignatura, que se creó un vínculo enorme entre muchos de nosotros.

Y por supuesto, necesito dar las gracias a la gente que me ha rodeado en mis aventuras fuera de la UMH, en las Becas Destino y Erasmus. Personas que han conseguido que toda esta experiencia haya sido aún más satisfactoria, haciendo que mi trayectoria universitaria sea para siempre inolvidable.

Finalmente, debo agradecer infinitamente a mi tutor, César Fernández Peris, por haber tenido la paciencia y serenidad tan inmensas a la hora de ayudarme a elaborar este trabajo, ya que sin su ayuda no habría sido posible alcanzar nuestro propósito.

RESUMEN

El presente proyecto, *"Desarrollo de una Aplicación Android para Gestión de Rutas",* y al cual se le ha asignado el título de "WALK & RIDE" como nombre de la aplicación, concede la explicación detallada del proceso de programación, diseño y puesta a punto de una app para dispositivos móviles de Android.

Este trabajo ha sido llevado a cabo gracias a la propuesta de un curso de programación impartido por César Fernández Peris y Mª Asunción Vicente Ripoll, profesores de la Universidad Miguel Hernández de Elche en el área de Ingeniería de Sistemas y Automática, y todo ello con el objetivo de permitir al alumnado iniciar su propio desarrollo de una aplicación con el programa Android Studio.

Se espera con este trabajo adquirir conocimiento, soltura y familiarizarse con el uso del lenguaje informático JAVA a la hora de introducirse en el mundo de la programación.

ÍNDICE

TABLA DE ILUSTRACIONES

1. Introducción y Descripción de la Aplicación

En este documento se recoge el trabajo de investigación y desarrollo realizado por el alumno Darío Barberá Madrid para completar el Trabajo de Fin de Grado (TFG), finalizando con el mismo el Grado de Ingeniería Mecánica en la Universidad Miguel Hernández de Elche.

"WALK & RIDE" es una aplicación para dispositivos Android, la cual ha sido desarrollada con el objetivo de permitir en ella la búsqueda de diversos tipos de rutas de senderismo y también para ciclistas o motoristas, para así facilitar a sus usuarios distintas opciones a la hora de realizar sus excursiones.

Además de esto, también se les da la posibilidad de crear sus propios recorridos en la aplicación, para que con ello guarden caminos que no hayan obtenido por medio de la app, y así tengan estos a su disposición en su dispositivo por si quisieran repetirlo en otro momento, o además para mostrarlo a otra gente que también pudiera estar interesada en este tipo de actividades, compartiendo así las experiencias más gratificantes a la hora de emprender un recorrido caminando o sobre ruedas.

Por supuesto, en esta aplicación podremos obtener la información esencial de nuestra nueva ruta, siendo así mostrados los kilómetros a recorrer en cada una de ellas, y una variada lista de opciones para los distintos tipos de usuario junto con los niveles de dificultad.

2.Objetivos y Motivación

Actualmente vivimos en una sociedad que cada vez se encuentra más vinculada con la alta tecnología, y esta evoluciona constantemente ayudando al ser humano en su vida diaria y facilitando muchas actividades cotidianas ya en el presente. Con la aparición de los Smartphones ha cambiado radicalmente nuestra forma de vida, ya sea tanto para el ámbito de contacto como para el ocio y tiempo libre.

Debido a que en mi Grado Universitario de Ingeniería Mecánica no obtenemos ningún tipo de aprendizaje respecto al lenguaje informático de JAVA, la propuesta que nuestro tutor César nos facilitó para realizar nuestro trabajo de fin de grado me resultó de lo más interesante, ya que de esta manera podría adquirir ciertos conocimientos sobre la programación informática, y además podría resultarme de gran ayuda a la hora de obtener posibles ofertas de trabajo en el futuro.

Al mismo tiempo, el hecho de introducirse en este tipo de proyecto también implicaba el aprendizaje de una forma de organizar el tiempo y los recursos que no había sido precisada hasta ahora.

Y además de todo esto, también esperaba con ello poder familiarizarme con el uso de APIs y servicios externos como Google Maps API o el programa de desarrollo ANDROID STUDIO, habituales a la hora de crear aplicaciones para dispositivos Android.

Por lo tanto, no dudé en aceptar la creación de este proyecto tras la propuesta de nuestro tutor, y con ello tener la posibilidad de introducirme aún más en este mundo y conocer más profundamente como funciona esta ingeniería.

Y de este modo, nuestro objetivo desde un primer momento ha sido básicamente el poder crear una aplicación para dispositivos móviles de Android que permita la visualización de un número determinado de rutas de ciclismo/motociclismo (o también senderismo), donde además se obtenga la posibilidad de creación de rutas por los propios usuarios en sus dispositivos.

Esta idea de aplicación ha surgido por la existencia de otros ejemplos, los cuales han triunfado en la actualidad sirviendo de gran ayuda para aquellos fanáticos de este tipo de actividades.

La aplicación que más ha servido de referencia ha sido *“WIKILOC”,* la cual ha servido de gran motivación para animarnos a empezar este proyecto con el fin de obtener un desarrollo semejante en algunas funciones y porque realmente contribuye muchísimo a ejercer esta actividad que a mucha gente le apasiona.

Y por supuesto, señalaremos también que esta aplicación que hemos desarrollado en ningún momento ha sido planteada con ánimo lucrativo, por tanto, no se espera obtener ningún beneficio más allá del meramente formativo.

En el apartado de las *Aplicaciones Similares* se verán algunos de los ejemplos más descargados en el mercado para este tipo de posibilidades en una aplicación, como la ya mencionada *"WIKILOC"*.

2.1 Contexto de Desarrollo

2.1.1 Mundo Android

Como ya he mencionado anteriormente, la aplicación ha sido desarrollada para el sistema operativo Android.

Android es un sistema operativo desarrollado por Google y la Open Handset Alliance, basado en el núcleo Linux y diseñado principalmente para dispositivos móviles con pantalla táctil, como teléfonos inteligentes (smartphones), tabletas y también para relojes inteligentes, televisores y automóviles.

Inicialmente fue desarrollado por Android Inc., empresa que Google respaldó económicamente y más tarde, en 2005, compró.

Este sistema operativo fue presentado en 2007 junto a la fundación del Open Handset Alliance (consorcio de compañías de hardware, software y telecomunicaciones), para avanzar en los estándares abiertos de los dispositivos móviles. Además, ese mismo año se anunció la primera versión del sistema operativo: Android 1.0 Apple Pie.

Aun así, terminales con Android no estarían disponibles hasta el año 2008, y el primer dispositivo con el sistema operativo Android sería el HTC Dream, estrenado en octubre de 2008.

Desde ese mismo momento, la implantación del sistema ha sido progresiva desde unos inicios marcados por una gran fragmentación en la elección de cada fabricante en el sistema operativo escogido para sus dispositivos.

Actualmente, Android es el sistema operativo móvil más utilizado del mundo, con una cuota de mercado que en el 2019 ha superado el 90 %, muy por encima de IOS, el sistema operativo para sistemas móviles de la compañía Apple y segundo sistema operativo más extendido.

A continuación tenemos una representación de como ha evolucionado la cuota del año 2010 al 2017 mostrando cada uno de los sistemas operativos más importantes.

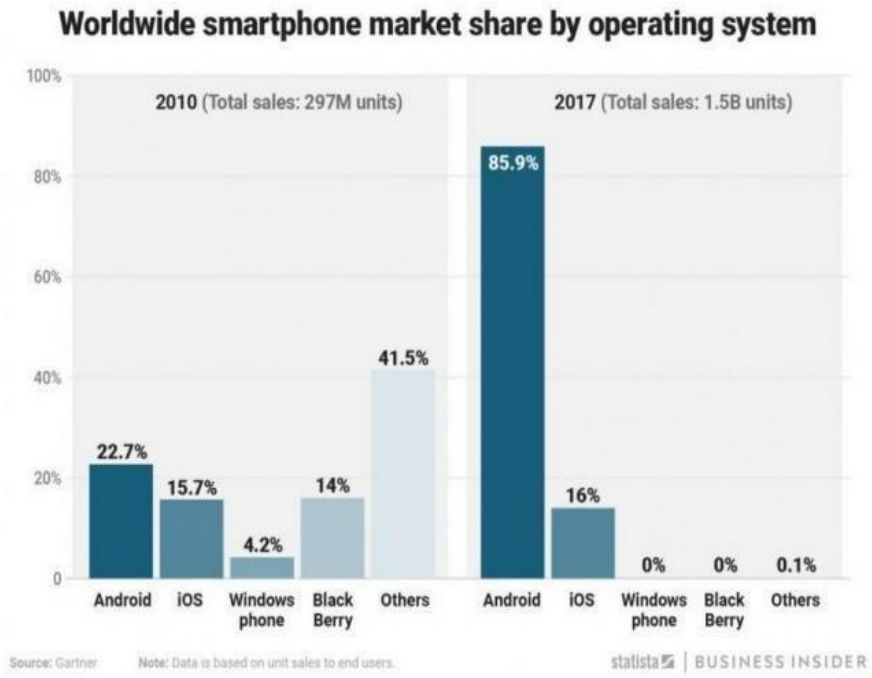

Ilustración 1: Evolución cuota de mercado Android

Cabe decir que el historial de versiones de este sistema operativo se inició en noviembre del 2007 con el lanzamiento de Android Beta.

La primera versión comercial (de prueba), Android 1.0, fue lanzada en septiembre de 2008, y el sistema operativo base desde su lanzamiento original, ha sufrido un constante actualizado, con el cual se pretenden corregir fallos de programa y agregan nuevas funcionalidades.

Desde abril de 2009, tras la versión de prueba y la primera estandarizada, las versiones de Android regularmente actualizadas han sido desarrolladas bajo un nombre en clave y sus nombres siguen un orden alfabético: Cupcake, Donut, Éclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwich, Jelly Bean, KitKat, Lollipop, Marshmallow, Nougat, Oreo, Android Pie, y el último en sumarse a la lista ha sido Android 10, la décima y última versión principal y la decimoséptima versión del sistema operativo Android.
“Android 10” fue lanzado oficialmente el 3 de septiembre de 2019.

Nombre código	Número de versión	Fecha de lanzamiento	Nivel de API
Apple Pie	1.0	23 de septiembre de 2008	1
Banana Bread	1.1	9 de febrero de 2009	2
Cupcake	1.5	25 de abril de 2009	3
Donut	1.6	15 de septiembre de 2009	4
Eclair	2.0 – 2.1	26 de octubre de 2009	5 – 7
Froyo	2.2 – 2.2.3	20 de mayo de 2010	8
Gingerbread	2.3 – 2.3.7	6 de diciembre de 2010	9 – 10
Honeycomb	3.0 – 3.2.6	22 de febrero de 2011	11 – 13
Ice Cream Sandwich	4.0 – 4.0.5	18 de octubre de 2011	14 – 15
Jelly Bean	4.1 – 4.3.1	9 de julio de 2012	16 – 18
KitKat	4.4 – 4.4.4	31 de octubre de 2013	19 – 20
Lollipop	5.0 – 5.1.1	12 de noviembre de 2014	21 – 22
Marshmallow	6.0 – 6.0.1	5 de octubre de 2015	23
Nougat	7.0 – 7.1.2	15 de junio de 2016	24 – 25
Oreo	8.0 – 8.1	21 de agosto de 2017	26 – 27
Pie	9.0	6 de agosto de 2018	28
Android 10	10.0	3 de septiembre de 2019	29

Ilustración 2: Versiones sistema operativo Android

El problema con las versiones de Android es que no todos los dispositivos se actualizan a la última versión, y esto, unido al poco espacio temporal entre unas versiones y otras, hacen que haya muchos dispositivos operativos con distintas versiones de Android.

Este factor se ha de tener en cuenta a la hora de desarrollar en Android, cada versión de Android tiene un nivel de API o Application Programming Interface (interfaz de programación de aplicaciones), esto es importante porque utilizar un nivel u otro de API puede hacer que una aplicación no pueda ser utilizada en versiones más antiguas de Android, pero por otro lado cuanto mayor sea el nivel de API mayor número de ventajas y recursos para el desarrollador.

Por este motivo hay que alcanzar un compromiso entre las ventajas que puedan ofrecer las nuevas versiones de Android y el número de dispositivos en el mercado con un sistema operativo capaz de ejecutar tu aplicación.

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	0.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5%
4.1.x	Jelly Bean	16	1.9%
4.2.x		17	2.9%
4.3		18	0.8%
4.4	KitKat	19	12.8%
5.0	Lollipop	21	5.7%
5.1		22	19.4%
6.0	Marshmallow	23	28.6%
7.0	Nougat	24	21.1%
7.1		25	5.2%
8.0	Oreo	26	0.5%
8.1		27	0.2%

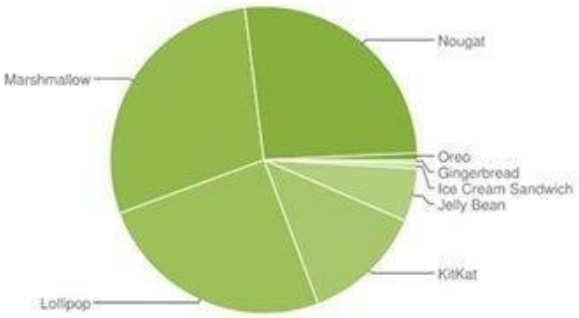

Ilustración 3: Fragmentación dispositivos Android 2018

3. Herramientas de Desarrollo en el Proyecto

En este apartado se definirán los programas de desarrollo dentro de la programación en Android, que han servido para la creación del TFG, por un lado el entorno de Mit App Inventor 2, y por otro el programa Android Studio.

3.1 Mit App Inventor 2

Este programa fue utilizado al inicio de nuestro curso de programación, con ayuda de nuestra profesora Mª Asunción Vicente Ripoll, y aunque con él no hemos desarrollado el trabajo, al tener unos controles muy intuitivos y sencillos a la hora de programar, nos sirvió, y mucho, para comprender poco a poco en que se basaba lo que más adelante sería el proceso de creación de nuestra aplicación.

App Inventor es un entorno de desarrollo de software creado por *Google Labs* para la elaboración de aplicaciones destinadas al sistema operativo *Android*. La plataforma se puso a disposición del público el 15 de diciembre de 2010 y está dirigida a personas que no están familiarizadas con la programación informática.

El usuario puede, de forma visual y a partir de un conjunto de herramientas básicas, ir enlazando una serie de bloques para crear la aplicación. El sistema es gratuito y se puede descargar fácilmente de la web.

Dentro de sus inconvenientes se encuentra que las aplicaciones creadas con App Inventor están limitadas por su simplicidad, aunque permiten cubrir un gran número de necesidades básicas en un dispositivo móvil.

Las características que tiene son:

- Está basado en JavaScript para crear un lenguaje visual.

- Permite crear una aplicación en menos tiempo que otros entornos de programación.

- La interfaz gráfica permite al usuario crear aplicaciones con muchas funcionalidades.

Las interfaces que podemos observar en Mitt App Inventor 2 son las siguientes:

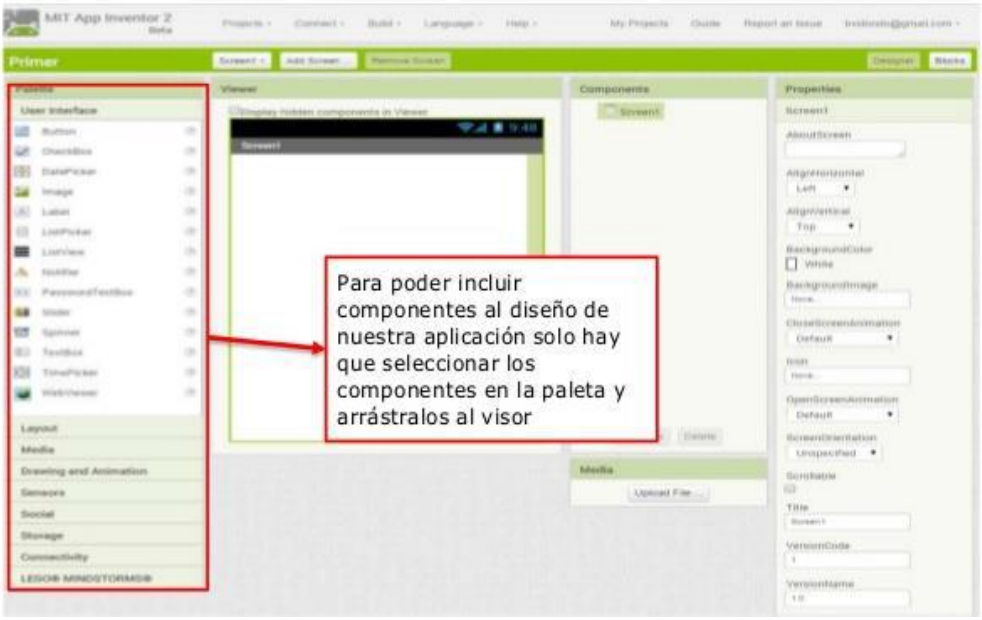

Ilustración 4: Controles de diseño

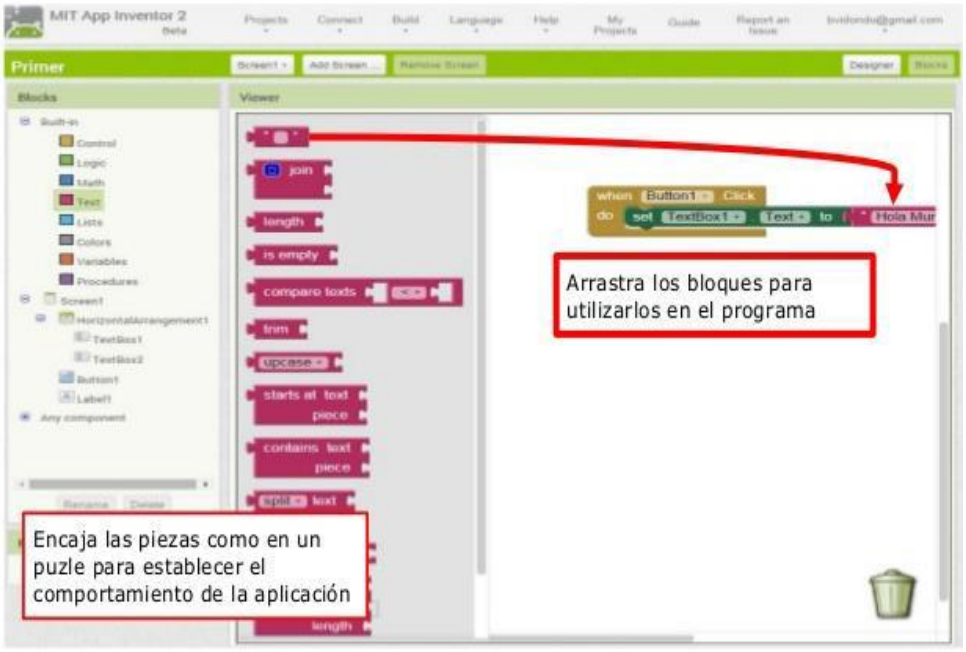

Ilustración 5: Editor de Bloques

3.2 Android Studio

Android Studio es el entorno de desarrollo integrado oficial para el desarrollo de apps para Android, comercializado por Google, y diseñado para ofrecer nuevas herramientas para la creación de las aplicaciones como alternativa al entorno Eclipse, hasta ahora el IDE más utilizado, y como ya dijimos, es el entorno empleado para la creación de nuestra app.

¿Qué ofrece Android Studio?

- Un entorno de desarrollo claro y robusto.
- Facilidad para probar el funcionamiento en diversos tipos de dispositivos.
- Asistentes y plantillas para los elementos comunes de programación en Android.
- Renderizado en tiempo real.
- Un dispositivo virtual de Android que se utiliza para ejecutar y probar aplicaciones.
- Un editor de diseño enriquecido que permite a los usuarios arrastrar y soltar componentes de la interfaz de usuario.
- Alertas en tiempo real de errores sintácticos, compatibilidad o rendimiento, antes de compilar la aplicación.

La interfaz de trabajo de este software es la siguiente:

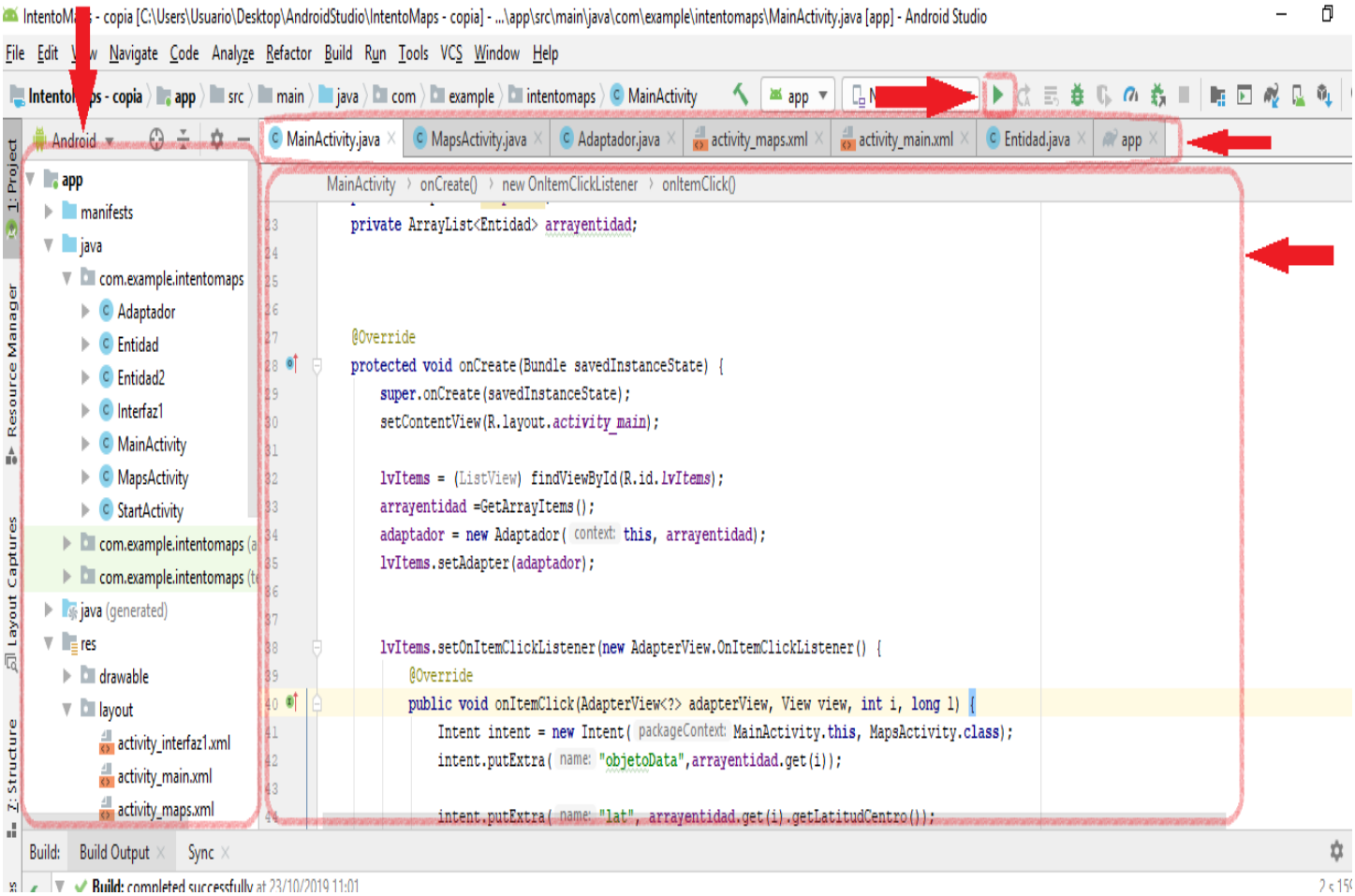

Ilustración 6: Interfaz de desarrollo Android Studio

A la izquierda se encuentra el panel con los diferentes archivos que estructuran el proyecto (Java, xml, etc.). En la parte central se sitúa el cuadro donde se programan los códigos, y en el panel de arriba los nombres de los archivos con los que se está trabajando, señalando de forma resaltada aquel en el que actualmente nos situamos para editar el código.

Este entorno de desarrollo integra emuladores para probar las aplicaciones. Sin embargo, también es posible conectar dispositivos móviles al ordenador de trabajo y, tras instalar los respectivos controladores o *drivers*, probar las

aplicaciones en nuestros dispositivos, pulsando en el símbolo “Play” indicado en la ilustración.

En cuanto a los requisitos del sistema para su instalación y uso:

	Windows	macOS	Linux
Versión SO	10/8/7 (32 o 64-bit)	Mac OS X 10.10 (Yosemite) o superior, hasta 10.13 (macOS High Sierra)	GNOME o KDE desktop
RAM	3 GB RAM mínimo, 8 GB RAM recomendado y 1 GB más para el emulador de Android		
Espacio en disco	2 GB de espacio en disco para Android Studio, 4 GB recomendados (500 para la IDE y al menos 1,5 GM para Android SDK, imágenes de sistema de emulador y cachés)		
Versión JAVA	Java Development Kit (JDK) 8		
Resolución de pantalla	1280x800 mínimo, 1440x900 recomendado		

Ilustración 7: Tabla de Requisitos para uso de Android Studio

3.1.1 Instalación

Para conseguir instalar Android Studio es necesario disponer del Java Development Kit (JDK).

Una vez completada la instalación del JDK, se procede a la descarga del programa, del SDK (Kit de Desarrollo de Software) de Android y de todos los paquetes del SDK necesarios para asegurar la compatibilidad con los dispositivos Android en los que se desee desarrollar.

3.1.2 Primeros Pasos en Android Studio

Para familiarizarse por primera vez con el programa, mi tutor César Fernández ofreció un curso de aprendizaje básico con el que al mismo tiempo de explicarnos las principales funciones de Android Studio, también nos ayudó a concretar posibles ideas de proyectos en las cuales basarnos para crear nuestro propio trabajo.

Una vez establecido un objetivo de desarrollo, ya comencé a seguir instrucciones más enfocadas en mi idea de proyecto mediante diversos tutoriales y explicaciones expuestas en páginas web dedicadas a la resolución de dudas con este programa.

4.Materiales y Métodos

4.1 Google Maps

La plataforma de mapas de *Google*, *Google Maps*, es la integrada en la aplicación. Se trata de un servidor de mapas que pertenece a *Alphabet Inc*. y ofrece imágenes de mapas desplazables y fotografías por satélite de la Tierra.

Características del Servidor

Este servidor permite acercar y alejar el mapa al usuario, y tanto si se trabaja desde un ordenador personal o desde un dispositivo móvil, podremos controlar el mapa y moverlo a la ubicación que queramos. Además, se podrá controlar el nivel de zoom que más nos convenga.

Una de las características más interesantes para nuestro tipo de aplicación que incluye *Google Maps* es la posibilidad de establecer las rutas más convenientes y breves posibles de un punto a otro que escojamos, además de facilitarnos varias opciones de tramos a la hora de realizar nuestro recorrido hasta el objetivo definido. En algunas zonas, también se nos permitirá visualizar los caminos en 3D, cosa que ayudará en caso de confusión. El problema de todo esto es que todas estas funciones serán habilitadas bajo pago al servidor de Google por derechos, cosa que ya se incluye por supuesto en todas las aplicaciones de la "Store" de Android.

Coordenadas

En cuanto al sistema de coordenadas utilizado en *Google Maps, este* es el WGS84, (del inglés *World Geodetic System 84*, que significa Sistema Geodésico Mundial 1984).

Este sistema de coordenadas geográficas mundial permite localizar cualquier punto en la Tierra sin necesitar otro de referencia, con un error de cálculo menor a 2 cm. Es en el que se basa el Sistema de Posicionamiento global (GPS) y está expresado en grados, minutos y segundos.

Para realizar los cálculos de distancias y superficies, es necesario transformar estas coordenadas geodésicas en cartesianas. Para ello se utiliza el sistema UTM.

El sistema de coordenadas universal transversal de Mercator (en inglés *Universal Transverse Mercator*, UTM) esta basado en la proyección cartográfica transversa de Gerardus Mercator, un importante geógrafo, matemático y cartógrafo flamenco del siglo XVI.

Este sistema divide la tierra en 60 husos de 6º de longitud. Cada huso se numera con un número entre el 1 y el 60 (1er huso: longitudes 180º-174º, meridiano 177ºW), y cada uno tiene asignado un meridiano central, donde se sitúa el origen de coordenadas. Además, estos se numeran en orden ascendente hacia el este.

En cuanto a la latitud, el sistema divide la Tierra en 20 bandas de 8º de latitud, denominadas con letras desde la C hasta la X, excluyendo algunas como la I y la O por evitar confusiones. Las letras menores de N se sitúan en el hemisferio sur, mientras que la N y superiores para el hemisferio norte.

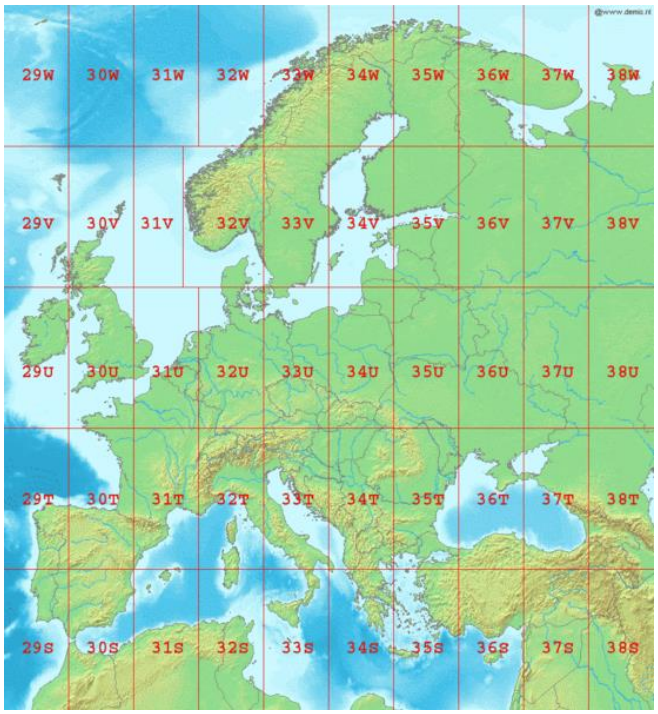

Ilustración 8: Lars Aronsson Coordenadas (2006)

La figura anterior muestra un mapa que, con las zonas de latitud y longitud del sistema de coordenadas universal transversal de Mercator de Europa, se puede observar que van desde la zona 29S a la 38W.

Se trata de coordenadas latitud y longitud siendo positivas para Norte y Este, y negativas para Sur y Oeste.

4.2 Lenguaje de Programación

- *Android Studio* trabaja principalmente con dos tipos de archivo: **XML** y **JAVA**.

Archivo XML

Tiene como misión crear las interfaces (o *layouts*) que el usuario visualiza y con las que interactúa. Es el encargado de la parte estética de las aplicaciones, aunque también es posible modificar elementos desde la clase *java*.

Archivo JAVA

En estos archivos es donde se programan las instrucciones para el comportamiento de los elementos declarados en los archivos xml. Es necesario identificar los elementos de los archivos xml y declararlos en el java. Esto es posible mediante la instrucción *findViewById*. En el apartado de "*Archivos XML y JAVA*", serán enumerados los archivos xml y java que contiene el proyecto de la aplicación.

4.3 Diseño y Programación

4.3.1 Estructura del Proyecto

Cada proyecto en Android Studio contiene uno o más módulos con archivos de código fuente y archivos de recursos. Entre los tipos de módulos se incluyen los siguientes:

-Módulos de apps para Android

-Módulos de bibliotecas

-Módulos de Google App Engine

De manera predeterminada, Android Studio muestra los archivos de tu proyecto en la vista de proyectos de Android, como se muestra en la ilustración 7. Esta vista se organiza en módulos para proporcionar un rápido acceso a los archivos de origen clave de tu proyecto.

Todos los archivos de compilación son visibles en el nivel superior de Secuencias de comando de Gradle y cada módulo de la aplicación contiene las siguientes carpetas:

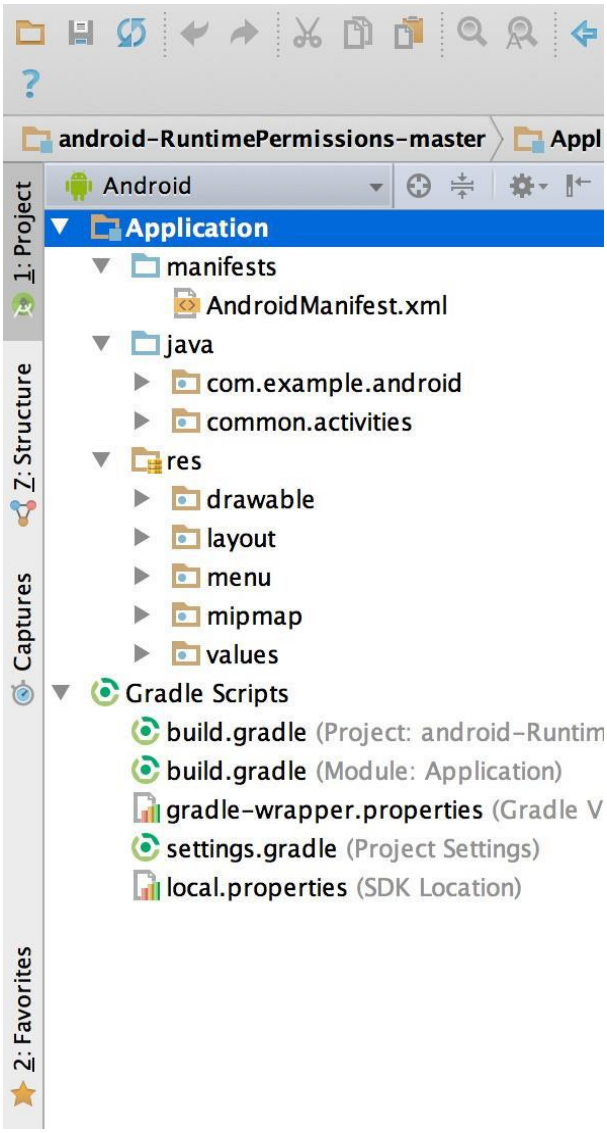

Ilustración 9: Archivos del Proyecto

- Manifest: Contiene el archivo AndroidManifest.xml.

- Java: Contiene los archivos de código fuente de java, incluido el código de prueba JUnit.

- Res: Contiene todos los recursos, como los diseños XML, cadenas de IU e imágenes de mapa de bits.

La estructura del proyecto para Android en el disco difiere de esta representación plana. Para ver lo que sería el esquema de archivos real del proyecto, debemos seleccionar *Project* en la lista desplegable arriba del todo, en la ventana la cual se muestra como Android en la ilustración anterior.

También puedes personalizar la vista de los archivos del proyecto para enfocarte en aspectos específicos del desarrollo de tu app. Por ejemplo, al seleccionar la vista *Problems* de tu proyecto, aparecerán enlaces a los archivos de origen que contengan errores conocidos de codificación y sintaxis, como un punto y coma o una etiqueta de cierre faltante para un elemento XML en un archivo de diseño.

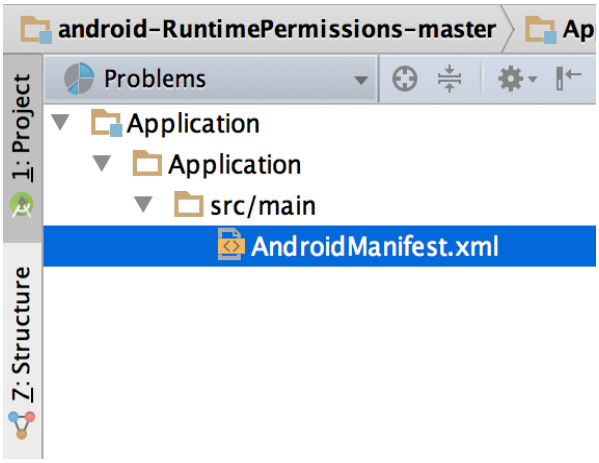

Ilustración 10: Archivos del proyecto en la vista Problems, mostrando archivo de diseño con un problema.

Códigos Java de Actividades Principales

En este proyecto contaremos con varias actividades principales, cada una de ellas con su respectiva interfaz o “layout”, y enlazadas con otras actividades que conectarán estas para escoger la función que el usuario desee obtener en la aplicación.

Actividad Inicial (StartActivity.java)

Con esta interfaz dará comienzo nuestra aplicación cuando ésta sea lanzada en el dispositivo. Aquí tendremos acceso a dos botones, uno con la función de entrar en el menú principal, y otro botón para obtener información detallada sobre la app.

Ilustración 11: Actividad de Inicio

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_start);

}

//Pulsar COMENZAR

public void Comenzar (View view) {
    Intent comenzar = new Intent( packageContext: this, Interfaz1.class);
    startActivity(comenzar);

}

//Pulsar INFO
public void info(View view) {
    final AlertDialog.Builder dialog = new AlertDialog.Builder( context: this);
    dialog.setTitle("¡Bienvenidos a WALK & RIDE!")
            .setMessage("Con esta aplicación podrás disfrutar de tu afición por ...")
            .setNegativeButton("Salir", new DialogInterface.OnClickListener() {
                @Override
                public void onClick(DialogInterface paramDialogInterface, int paramInt) {
                }
            });
    dialog.show();

}
```

Ilustración 12: StartActivity.java

Y aquí podemos ver en el lenguaje JAVA como han sido creadas estas funciones de cada uno de los botones para que puedan cumplir con lo que se pretende.

Al tener dos botones en el layout, toca declarar ambos en este archivo con sus respectivas funciones a la hora de lanzar la app al móvil.

Por una parte, en el botón *START*, gracias al uso del código *Intent*, el cual nos sirve como objeto para solicitar una acción determinada, en nuestro caso queremos que nos lleve a una nueva *Activity* que será la del menú de la app. Con ello, indicaremos cual es la nueva interfaz a la que queremos acceder (*Interfaz1*) desde la actual:

```
new Intent (StartActivity.this, Interfaz1.class);
startActivity(comenzar);
```

Tras ello, tan solo añadiendo el "*startActivity",* activaremos el funcionamiento de este botón.

En cuanto al botón *Información,* gracias al código "*Alert.Builder dialog",* el cual hemos empleado varias veces en nuestro desarrollo, en lugar de tener que crear una nueva actividad solo para introducir un breve texto, esta orden nos permite añadir una ventana reducida al pulsar el botón, donde aparecerá aquel texto que nosotros hayamos vinculado con el código mencionado.

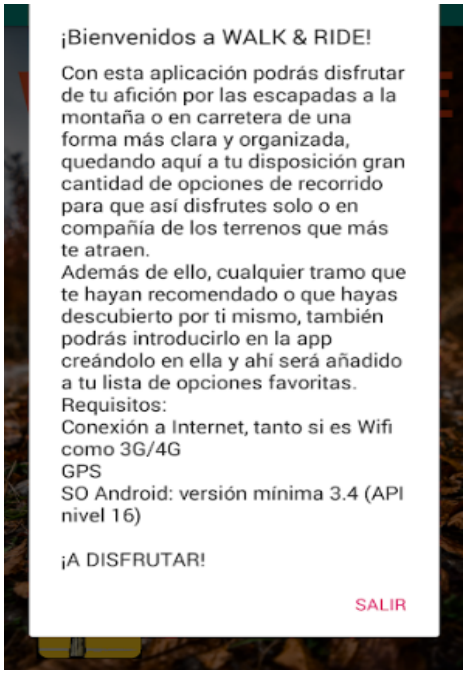

Ilustración 13: Alert.Builder dialog

Actividad Menú

Actividad a la que el usuario accede tras el botón de inicio de la actividad que se abre al lanzar la aplicación. En ella se da la opción de pasar a la lista de rutas por defecto que la aplicación contiene, y también a la actividad de creación de una propia ruta o al conjunto de rutas que hayan sido creadas por el usuario.

Ilustración 14: Actividad Menú

```java
public class Interfaz1 extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_interfaz1);
    }

    // Pulsar Rutas

    public void Destinos (View view) {
        Intent destinos = new Intent ( packageContext: this, MainActivity.class);
        startActivity(destinos);
    }
    public void Create (View view) {
        Intent create = new Intent ( packageContext: this, CreateActivity.class);
        startActivity(create);
    }
    public void Misrutas (View view) {
        Intent misrutas = new Intent( packageContext: this, MyRoutesActivity.class);
        startActivity(misrutas);
    }
}
```

Ilustración 15: Interfaz1Activity.java

Y aquí, vemos como se establecen también las funciones de los botones de la Actividad Menú, con los cuales tendremos acceso a las actividades más importantes de nuestra aplicación.

Actividad Rutas de la Propia App (MainActivity.java)

En esta *Activity* tendremos el catálogo de rutas que la app contiene por defecto, para distintos tipos de usuario.

Aquí, será añadido un ListView, una subclase pública de la clase *AdapterView*, que nos servirá para mostrar datos en lista, tal como puede ser una lista de contactos, lista de productos, o de nuestras rutas en este caso. La ventaja de utilizar esta clase es que puedes añadir, modificar o eliminar elementos de la lista cuando quieras de forma sencilla, y además con esto facilitamos el hecho de que todos los componentes de la lista den acceso a una actividad específica, sin necesidad de crear tantas "*Activities"* como elementos tengamos en nuestra lista.

Los *AdapterView* requieren crear un archivo "*adaptador"*, objetos de programación que sirven como puente entre los *AdapterView* y la información que se desea mostrar.

El *adaptador* dará acceso a los archivos de información y es responsable de mostrar una vista (o *view*) para cada ítem de información. En nuestro proyecto, el archivo "Adaptador.java" es el *adapter* que relaciona el *ListView* con el *"ArrayList<Entidad>"* de las rutas.

Además, necesitaremos un archivo *Entidad,* el cual hará la función de ordenar y establecer que datos queremos que se nos muestren en cada una de las rutas al acceder a una específica, es decir, sus propias coordenadas, su nombre y distancia, etc.

ScrollView

La clase *ScrollView* ofrece un contenedor para una jerarquía vertical de vistas que se pueda desplazar por el usuario, introduciéndolo en nuestra actividad para que pudiéramos acceder a todas las rutas que teníamos en nuestro ArrayList, incluidas aquellas que no cupieran en la pantalla, pudiendo así con el dedo subir y bajar la lista. Un ejemplo de ScrollView sería el siguiente:

```
<ScrollView
    android:layout_width="match_parent"
    android:layout_height="90dp"
    android:layout_weight="5">
    <TextView
        android:id="@+id/listado1"
        android:layout_margin="0dp"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="" />
</ScrollView>
```

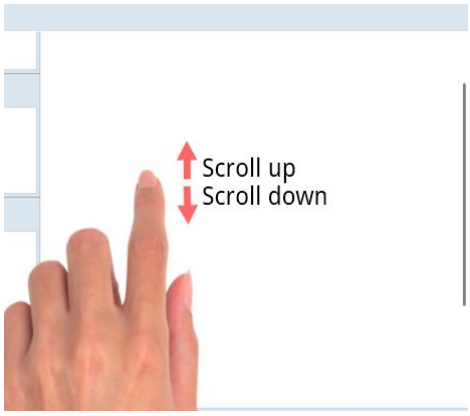

Ilustración 16: Función ScrollView

Sin embargo, no se ha incluido finalmente en el proyecto, ya que, al emplear la clase ListView en nuestro trabajo, junto con el ArrayList, este método implementará automáticamente el *"Scrolling",* para que el usuario pueda desplegar sin ningún problema todos los elementos ocultos.

```java
public class MainActivity extends AppCompatActivity {
    private ListView lvItems;
    private Adaptador adaptador;
    private ArrayList<Entidad> arrayentidad;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        lvItems = (ListView) findViewById(R.id.lvItems);
        arrayentidad =GetArrayItems();
        adaptador = new Adaptador( context: this, arrayentidad);
        lvItems.setAdapter(adaptador);


        lvItems.setOnItemClickListener((adapterView, view, i, l) -> {
                Intent intent = new Intent( packageContext: MainActivity.this, MapsActivity.class);
                intent.putExtra( name: "objetoData",arrayentidad.get(i));

                intent.putExtra( name: "lat", arrayentidad.get(i).getLatitudCentro());
                intent.putExtra( name: "lng", arrayentidad.get(i).getLongitudCentro());
                intent.putExtra( name: "zoom", arrayentidad.get(i).getZoomMapa());
                startActivity(intent);
```

Ilustración 17: MainActivity.java

```java
puntosPirineoLat.add(new Float( value: -4.410748));
puntosPirineoLat.add(new Float( value: -5.409531));
puntosPirineoLat.add(new Float( value: -5.732059));
puntosPirineoLat.add(new Float( value: -6.243071));
puntosPirineoLat.add(new Float( value: -6.516268));
puntosPirineoLat.add(new Float( value: -7.000616));
puntosPirineoLat.add(new Float( value: -7.368137));
puntosPirineoLat.add(new Float( value: -7.723003));
puntosPirineoLat.add(new Float( value: -8.009946));
puntosPirineoLat.add(new Float( value: -8.27525));
puntosPirineoLat.add(new Float( value: -8.545846));
listItems.add(new Entidad(R.drawable.ic_pirineo,  titulo: "Ruta Pirineos",


listItems.add(new Entidad(R.drawable.ic_castellon,  titulo: "Ruta Castellón"
listItems.add(new Entidad(R.drawable.ic_granada,  titulo: "Ruta Granada",
```

Ilustración 18: Construyendo el ArrayList

Aquí podemos observar todo el *Activity* del *ArrayList* para nuestra lista de rutas variada y por defecto en nuestra aplicación, donde añadimos un conjunto de coordenadas específicas para cada recorrido, junto con la imagen y el nombre de cada uno de ellos y sus respectivos detalles.

A continuación vemos cómo quedaría nuestra interfaz de rutas:

Ilustración 19: Lista de rutas de la app

Actividades Google Maps

Son las actividades que en nuestra aplicación mostrarán el mapa con la ruta indicada, o bien con el mapa donde nosotros crearemos nuestra propia ruta punto por punto, siempre con el servidor de Google Maps introducido en ella, para así poder hacer uso de sus características. Este mapa de Google Maps se ha podido implementar mediante la clase SupportMapFragment.

SupportMapFragment

En *Android Studio*, un *fragment* representa un comportamiento o una parte de la interfaz de usuario en una actividad. Puedes combinar múltiples fragmentos en una sola actividad para crear una IU (Interfaz de Usuario) multipanel y volver a usar un fragmento en múltiples actividades. Puedes pensar en un fragmento como una sección modular de una actividad que tiene su ciclo de vida propio (en Android este ciclo será controlado por el sistema, y no por la aplicación, como en otros SO), recibe sus propios eventos de entrada y que puedes agregar o quitar mientras la actividad se esté ejecutando (algo así como una "subactividad" que puedes volver a usar en diferentes actividades).

Un fragmento siempre debe estar integrado a una actividad y el ciclo de vida del fragmento se ve directamente afectado por el ciclo de vida de la actividad anfitriona. Por ejemplo, cuando la actividad está pausada, también lo están todos sus fragmentos, y cuando la actividad se destruye, lo mismo ocurre con todos los fragmentos.

Sin embargo, mientras una actividad se está ejecutando (está en el estado del ciclo de vida *reanudada*), puedes manipular cada fragmento de forma independiente; por ejemplo, para agregarlos o quitarlos. Cuando realizas una transacción de fragmentos como esta, también puedes agregarlos a una pila de actividades administrada por la actividad; cada entrada de la pila de actividades en la actividad es un registro de la transacción de fragmentos realizada. La pila de actividades le permite al usuario invertir una transacción de fragmentos (navegar hacia atrás) al presionar el botón *Atrás*.

Cuando agregas un fragmento como parte del diseño de tu actividad, este se ubica en ViewGroup, dentro de la jerarquía de vistas de la actividad, y el fragmento define su propio diseño de vista. Puedes insertar un fragmento en el diseño de tu actividad declarando el fragmento en el archivo de diseño de la actividad (XML) como elemento <*fragment*> o agregándolo a un *Viewgroup* existente desde el código de tu aplicación. Sin embargo, no es necesario que un fragmento forme parte del diseño de la actividad; también puedes usar un fragmento con su IU propia, como un trabajador invisible para la actividad.

En nuestro caso, utilizamos ***SupportMapFragment***, una clase pública extendida de *fragment* que permite de manera sencilla implementar un mapa en la actividad. Además, esta clase es compatible con versiones de *Android* y niveles de API anteriores.

Mediante la clase getMapAsync (OnMapReadyCallback*),* automáticamente se inicializará el sistema de mapas y la vista.

Un *fragment* puede acceder a la instancia *Activity* mediante *getActivity()* y realizar tareas de manera sencilla. De la misma forma, para llamar a métodos del fragmento usamos *findFragmentById()*.

Un método, en programación, es una subrutina cuyo código es definido en una clase y puede pertenecer tanto a una clase como a un objeto. Existen dos tipos de métodos: de instancia (relacionados con un objeto en particular), y estáticos o de clase (asociados a una clase en particular).

Un ejemplo de este tipo de tareas en *fragment* sería el de la posibilidad de tomar un punto específico del mapa haciendo click sobre él. Para ello, utilizaríamos el siguiente código, que ya pertenece a la actividad de creación de ruta:

```
    // Marcar Puntos
puntosRuta = new ArrayList<>();
    mMap.setOnMapClickListener(new GoogleMap.OnMapClickListener() {
    @Override
    public void onMapClick(LatLng latLng) {
        puntosRuta.add(latLng);
        mMap.addMarker(new MarkerOptions().position(latLng));
    }
});
```

Ilustración 20: puntosRuta.add(latLng)

Tras ello, decidimos añadir un AlertDialog.Builder dialog, ya mencionado y empleado anteriormente, con lo que se desarrolló finalmente el siguiente código para el añadido de puntos en el mapa:

CreateActivity.java

```
puntosRuta = new ArrayList<>();
mMap.setOnMapClickListener((latLng) → {
        final AlertDialog.Builder dialog =new AlertDialog.Builder( context: CreateActivity.this);
        dialog.setTitle(R.string.addpoint)
                .setPositiveButton( text: "Sí", (dialogInterface, i) → {
                        puntosRuta.add(latLng);
                        Marker marker = mMap.addMarker(new MarkerOptions().position(latLng));
                        allMarkers.add(marker);
                    })
                    .setNegativeButton ( text: "No", new DialogInterface.OnClickListener() {
                        @Override
                        public void onClick(DialogInterface dialogInterface, int i) {
                        }
                    });
                        dialog.show();

                });

    }
```

Ilustración 21: DialogBuilder Add Points

Donde *mMap* es un objeto de la clase *googleMap* que hemos creado en la actividad. El método *onMapClickListener* es llamado cuando el usuario hace un click en cualquier parte del mapa. Este método proporciona la variable *LatLng*, compuesta por la latitud y longitud del punto del mapa sobre el cual se ha hecho el click para marcarlo.

Una vez llamado al método, se abrirá una ventana mediante el código *AlertDialog.Builder,* en el que aquí se le dará la opción al usuario de si quiere marcar el punto elegido o si desea anularlo. Si el usuario pulsa el botón positivo, implementado mediante *setPositiveButton*, se llama al método *OnClick* del mismo y se procede a enviar a la actividad principal la variable *LatLng,* recogida anteriormente, mediante el código ***puntosRuta*.add(latLng)**.

Este código aquí usado es el método cuya función es la de poder marcar un punto específico del mapa con un icono que nos proporciona el propio servidor de Google Maps, para así, uniendo varios de ellos una vez hayan sido marcados, obtener una ruta nueva a la cual el propio usuario podrá ponerle un nombre, y después, guardarla en su aplicación para darle un posterior uso.

Ilustración 22: CreateActivity.java

Ilustración 23: Añadir punto de ruta

Ahora se mostrará el código que permite el guardado de la ruta una vez la hallamos marcado punto por punto y después de darle un nombre:

```
//Guardar Ruta
public void saveRoute(View view) {
    String fileName = "ruta_1";
    String datos = "";
    for (LatLng punto : puntosRuta) {
        datos = datos + punto.longitude + "" + punto.latitude + "\n";
    }
    FileOutputStream fos;
    try {
        fos = openFileOutput(fileName, Context.MODE_PRIVATE);
        fos.write(datos.getBytes());
        fos.close();
    } catch (Exception e) {
        Log.e( tag: "MAPAS", e.getMessage());
    }
}
```

Ilustración 24: Guardar las Rutas

Aquí obtenemos la posibilidad de guardar en un archivo nuestra nueva ruta marcada en el mapa empleando la clase *FileOutputStream*, la cual nos sirve para escribir en un fichero los bytes de dicha ruta creada. También vamos a utilizar el modo Context.MODE_PRIVATE, que hará que el fichero sea privado a la aplicación, y con ello, se crearía el fichero si éste no existiera todavía.

Una vez guardada la ruta creada, haría falta conseguir que esta fuera leída dentro del dispositivo, y que de esta manera sea representada en el mapa.

Pero nuestro objetivo además ha sido el poder agrupar todas las rutas creadas en un ListView nuevo, una nueva Activity donde sean detectadas todas ellas y podamos escoger la que queramos visualizando los distintos tramos según el archivo que escojamos:

MyRoutesActivity.java

```
private ArrayList<String> GetArrayRutas() {
    ArrayList<String> names = new ArrayList<>();
    File[] files = getFilesDir().listFiles();
    int num = files.length;
    for (int i = 0; i < num; i++) {
        String name = files[i].getName();
        if (name.startsWith("RUTA ")) {
            names.add(name);
        }
    }
    return names;
```

Ilustración 25: ArrayList<String>

Así pues, con este código, crearemos un nuevo ArrayList, esta vez sin definir ninguno de los componentes del nuevo ListView, sino que con la función “listFiles” llamaremos a todos aquellos archivos de coordenadas que hayamos creado y guardado en la actividad anterior:

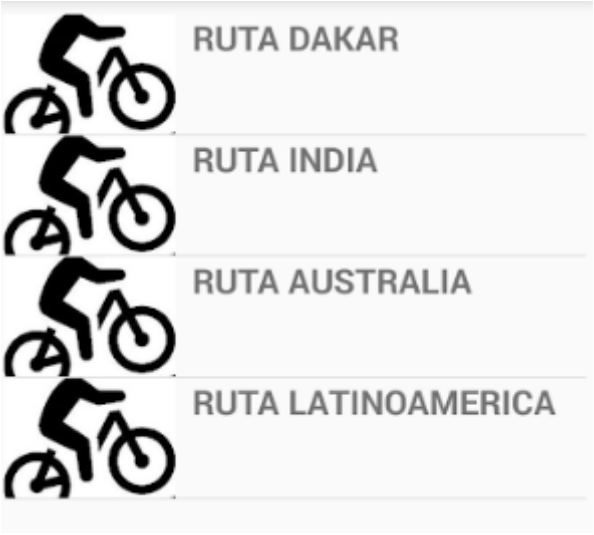

Ilustración 26: Lista Rutas Creadas

Ilustración 27: Ruta Creada Abierta

```
//Leer Ruta
public void readRoute () {

    String filename = getIntent().getStringExtra( name: "filename");

    ArrayList<LatLng> puntos = new ArrayList<>();
    try {
        FileInputStream fis = openFileInput(filename);
        InputStreamReader isr = new InputStreamReader(fis);
        BufferedReader br = new BufferedReader(isr);
        StringBuilder sb = new StringBuilder();
        String line;
        while ((line = br.readLine()) != null) {
            sb.append(line);
            String[] split = line.split( regex: " ");
            float longitud = Float.parseFloat(split[0]);
            float latitud = Float.parseFloat(split[1]);
            LatLng punto = new LatLng(longitud, latitud);
            puntos.add(punto);
        }

        isr.close();

    } catch (Exception e) {
        Log.e( tag: "MAPAS", e.getMessage());
```

Ilustración 28: Lectura de Rutas Creadas

Aquí arriba vemos el código que nos permitirá acceder a los archivos ya creados y guardados anteriormente, los que hemos podido ver en las imágenes anteriores de la aplicación, gracias a la clase *FileInputStream*, con la que se leerán los bytes ya escritos con el método *Output* anterior.

También cabe decir que, para no crear una nueva actividad *"fragment"* en la que añadir un nuevo mapa, hemos podido hacer uso del mapa ya empleado en la actividad *MapsActivity* gracias a un código de condiciones en el que según el archivo que abriéramos, la app nos daría acceso a una interfaz o a otra:

```
String filename = getIntent().getStringExtra("filename");
if (filename.equalsIgnoreCase("XXX")) {
    Item = (Entidad) getIntent().getSerializableExtra("objetoData");
    rutaPropia = false;
```

Y por último, también se ha conseguido implementar un conjunto de pasos que harán que, mediante una media aritmética, el enfoque del zoom y la colocación de la pantalla estén ajustados según la posición en la que esté situada nuestra ruta creada:

```
rutaPropia = true;
ArrayList<LatLng> puntos = readRoute();
ArrayList<Float> longs = new ArrayList<>();
ArrayList<Float> lats = new ArrayList<>();
double minLat = 1000;
double maxLat = -1000;
double minLong = 1000;
double maxLong = -1000;
for (LatLng punto : puntos) {
    lats.add(new Float(punto.longitude));
    longs.add(new Float(punto.latitude));
    if (punto.latitude>maxLong) maxLong = punto.latitude;
    if (punto.longitude>maxLat) maxLat = punto.longitude;
    if (punto.latitude<minLong) minLong = punto.latitude;
    if (punto.longitude<minLat) minLat = punto.longitude;
}
double longitudCentro = (maxLong+minLong)/2;
double latitudCentro = (maxLat+minLat)/2;

double longTot = maxLong-minLong;
double latTot = maxLat-minLat;
double distMax = max(longTot, latTot);
float zoom = 1;
if (distMax>10) zoom = 0.5f;
else if (distMax>5) zoom = 0.4f;
```

Ilustración 29: Ajuste Zoom y Coordenadas en Pantalla

3.3.2 Android Manifest Y Permisos

Todas las aplicaciones deben tener un archivo ***AndroidManifest.xml*** (con ese nombre exacto) en el directorio raíz.

El archivo de manifiesto proporciona información esencial sobre tu aplicación al sistema Android, información que el sistema debe de tener para poder ejecutar el código de la app.

Entre otras cosas, el archivo de manifiesto hace lo siguiente:

- Nombra el paquete de Java para la aplicación. El nombre del paquete sirve como un identificador único para la aplicación.

- Contiene los componentes de la aplicación, que incluyen todas las actividades, servicios, receptores de emisiones y proveedores de contenido. También nombra las clases que implementa cada uno de los componentes y publica sus capacidades, como los mensajes Intent con los que pueden funcionar. Estas declaraciones notifican al sistema Android los componentes y las condiciones para el lanzamiento.

- Describe los permisos que necesita la aplicación para acceder a las partes protegidas del sistema u otras aplicaciones. También declara cualquier permiso que otras aplicaciones deben tener si quieren acceder al contenido de esta aplicación.

- Determina los procesos que alojan a los componentes de la aplicación.

- Enumera las clases *Instrumentation* que proporcionan un perfil y otra información mientras la aplicación se ejecuta. Estas declaraciones están en el manifiesto solo mientras la aplicación se desarrolla y se quitan antes de la publicación de esta.

- Declara el nivel mínimo de API que requiere la aplicación.

- Enumera las bibliotecas con las que debe de estar vinculada la aplicación.

AndroidManifest.xml

manifest

```
        android:icon="@mipmap/ic_launcher"
        android:label="IntentoMaps"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".StartActivity"></activity>

        <activity
            android:name=".MapsActivity"
            android:label="Map">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="AIzaSyBlta7oBe2QwxzO1cbYlD-JwmErwF68W1s" />
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="12451000" />
    </application>
    <uses-permission android:name="com.google.android.providers.gsf.permission.READ_GSERVICES" />
    <uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.MAPS_RECEIVE" />
```

Ilustración 30: Archivo *Android Manifest*

Aquí tenemos el archivo ***AndroidManifest.xml,*** donde podemos ver los componentes de nuestra propia aplicación, incluyendo los permisos necesarios para su funcionamiento, puesto que requiere algunos indispensables para acceder a los servicios de ubicación y conexión a Internet. Los servicios se solicitan con los siguientes códigos del Manifest:

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

Este permiso es necesario para las aplicaciones de navegación, de envío de posición a otro contacto, etc. La app podrá localizarnos mediante GPS y conocer nuestra posición exacta.

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

Con esta línea damos permiso a la aplicación a tener acceso a Internet, muy importante en nuestra app para poder utilizar el servidor de Google Maps a la hora de escoger nuestras rutas.

3. `<uses-permission android:name="android.permission.MAPS_RECEIVE" />`

Este permiso está ligado al servidor de Google Maps e indicará donde se usará la API de Google. Así, permitirá que nuestro proyecto reciba el mapa.

Aquí incluyo algunos otros permisos también específicos para nuestra aplicación:

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

Al abrir la aplicación por primera vez, se solicitará el añadido de los permisos de ubicación al usuario, el cual debe aceptarlos para permitir a la aplicación acceder a esos servicios.

3.3.3 Mis Archivos Layout y JAVA

El proyecto de Android Studio contiene archivos de tipo XML y JAVA, tal cual hemos ya indicado anteriormente. A continuación se mostrarán todos los archivos creados y editados de nuestra propia aplicación junto con su correspondiente función:

XML

Con el uso de los archivos de tipo XML (*eXtensible Markup Language*), podemos diseñar rápidamente los elementos de cada actividad de pantalla, con una serie de elementos anidados.

Cada archivo de diseño debe contener exactamente un elemento raíz, que debe ser un objeto *View* o *ViewGroup*. Una vez que hayas definido el elemento raíz, puedes agregar widgets u objetos de diseño adicionales como elementos secundarios para crear gradualmente una jerarquía de vistas que defina tu diseño.

Función	Archivo
Diseño de la interfaz (*layout*) del Inicio.	activity_start.xml
Diseño de la interfaz menú	activity_interfaz1.xml
Diseño de la interfaz de los elementos de la lista de rutas por defecto	item.xml
Interfaz de elementos de lista de rutas creadas por el usuario	Item2.xml
Diseño interfaz de *fragment* con rutas fijas	activity_maps.xml
Diseño interfaz con *fragment* para creación de rutas	activity_create.xml
Diseño de interfaz de la Listview de rutas por defecto y guardadas	activity_main.xml
	activity_myroutes.xml

JAVA

En la carpeta Java encontramos el código fuente de la aplicación, y cada uno de los archivos que encontramos en esta carpeta se corresponde con una de las interfaces de usuario que hemos creado. Su función dentro de la aplicación es dotar de funcionalidades a todos los elementos que hayamos creado anteriormente, como botones, visores de texto, ventanas editables de texto, etc.

Función	Archivo
Actividades principales con clase *fragment* para el mapa y sus utilidades	MapsActivity.java
	CreateActivity.java
Actividad menú	Interfaz1.java
Clase *adapter* para relacionar el *ListView* con el *array* "Items" y "Puntos"	Adaptador.java
	Adaptador_2.java

3.3.4 Librerías y API

En informática, una librería o biblioteca es una colección de recursos externos los cuales el programador puede añadir a su programa para que estos le faciliten la ejecución de una serie de funciones entre sí, como por ejemplo la utilizada en nuestro caso que nos permite emplear los servicios de navegación GPS por medio de Google Maps, o también, si el programador necesita realizar un cálculo matemático complejo y difícil de programar.

Las librerías de software funcionan de la misma manera que una biblioteca tradicional: pides información sobre un tema concreto y después la utilizas dentro de tu proyecto según te convenga o la necesites.

Para introducir estas librerías en nuestro propio proyecto, hemos empleado la opción de *Project Structure*, en la cual, dentro de la pestaña *Dependencies*, podemos ver y modificar las librerías que ya están añadidas por defecto en el nuevo proyecto, y aquí mismo en el buscador, encontraremos aquellas que necesitemos introducir.

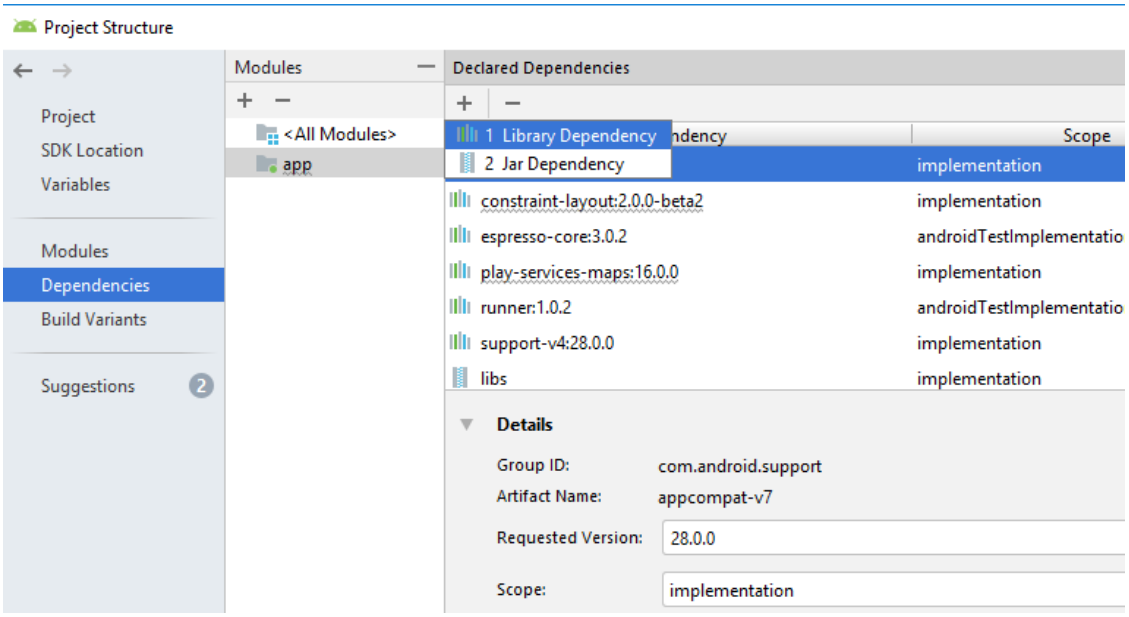

Ilustración 31: Proceso de búsqueda e introducción de nueva librería/biblioteca

Una vez hayamos añadido la nueva librería, también debemos declararla en las *Dependencies* del archivo *build.gradle (Module: app)* para que se actualice nuestro proyecto completamente.

Librería de Android SDK

Un SDK, o Kit de Desarrollo de Software, es un conjunto de herramientas de desarrollo que ayudan a la programación de aplicaciones para un entorno tecnológico particular.

El SDK de Android incluye, además de otras herramientas, un simulador de teléfono basado en QEMU, un depurador de código, una biblioteca y tutoriales. En este proyecto son empleadas diversas clases de esta librería.

Librería de Java

Esta librería contiene, por ejemplo, la clase ArrayList, cuyo uso será primordial en nuestro proyecto a la hora de desarrollar diversas listas de datos.

Google Maps API

¿Qué es una API? Traducida del inglés, esta es una “Interfaz de Programación de Aplicaciones” (Application Programming Interface), es un conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta librería para ser utilizado por otro software como una capa de abstracción. Solo que la implementación interna de esas funciones está oculta al público y los programadores sólo pueden ingresar datos de entrada y obtener así un resultado, sin saber lo que sucedió en el medio.

Google Maps Android API es un conjunto de APIs que permiten superponer datos en un mapa de Google personalizado, crear aplicaciones móviles con la poderosa plataforma de cartografía de Google, incluyendo imágenes de satélite, vista de la calle, etc.

Con la cobertura global más precisa del mundo y una comunidad activa llevando a cabo actualizaciones diarias, los usuarios se benefician de un servicio de mejora contínua.

Y para poder utilizar sus mapas, es necesario instalar en nuestro proyecto esta librería, solicitando unas credenciales desde nuestra cuenta de Google, indicando el nombre y el tipo de proyecto para el que la queremos utilizar.

Para la implementación de esta API, en el archivo *build.gradle* del proyecto se añade, en el apartado de *dependencies*:

```
implementation 'com.google.android.gms:play-services-maps: 16.0.0'
```

Este servicio de mapas de Google que ofrece diversas funcionalidades como, por ejemplo, marcadores, asignación de rutas, trazar tramos dentro de un mapa, etc, era, por tanto, lo prioritario a incluir en nuestro proyecto si queríamos hacer uso de un mapa para así señalar coordenadas y crear nuestras propias rutas.

3.3.5 Elementos Importantes

Aquí señalamos algunas de las clases más importantes que nos pueden servir específicamente a la hora de desarrollar nuestro trabajo.

- *GoogleMaps*: para incorporar el mapa.
- *ZoomMapa*: para poder hacer zoom.
- *Marker*: para añadir marcadores.
- *MarkerOptions*: para diseñar los marcadores.
- *Polyline*: para dibujar el conjunto de líneas conectadas.
- *PolylineOptions*: para hacer posible la creación de esa polilínea.
- *OnClickListener*: para llamar a una acción en el momento en que el usuario pulse el elemento asignado de la pantalla.

ArrayList de Java

Ya nombrada anteriormente, y de las clases más importantes en el proyecto, esta construye una lista de elementos de un tipo determinado según nos convenga, como por ejemplo una lista de títulos de películas, permitiendo almacenar dichos datos de forma dinámica, sin necesidad de declarar el número de datos añadidos.

En nuestra aplicación se emplean por ejemplo ArrayList del tipo <LatLng>, <Entidad>, <String> o <Float> (para guardar números que tengan tanto parte entera como decimal).

4.5 Requisitos

La aplicación exige unos requisitos mínimos de hardware y software para su correcto funcionamiento:

Requisitos de hardware:

- Dispositivo móvil con el SO *Android*, ya sea tableta o *Smartphone*.
- Conexión a internet, ya sea mediante Wifi o datos móviles (3G/4G).
- Conexión al sistema GPS.
- Memora RAM mínima de 512MB.
- Procesador CPU de 1Ghz mínimo.
- Memoria interna libre de 10MB.

Requisitos de software:

- *Android* versión 3.1 (Nivel de API 12) o superiores.
- *Google Play Service*.
- *Google Maps*.

4.6 Aplicaciones Similares

En este apartado, serán comentadas varias de las aplicaciones más exitosas del mercado a la hora de realizar actividades al aire libre que proporcionen funciones de ubicación y recorrido, y en las que más nos hemos fijado a la hora de realizar nuestro diseño de interfaces y utilidades para nuestra aplicación.

WIKILOC

Sin duda, esta aplicación ha sido nuestro referente ya que de ella surgió además la idea de poder crear algo similar, gracias a las grandes posibilidades que nos ofrece y la sencillez con la que ha sido diseñada para poderse así manejar sin ningún problema para todos aquellos que la utilicen.

Wikiloc es un servicio web gratuito para visualizar y compartir rutas y puntos de interés GPS para nuestros viajes de aventura en moto, rutas para bicicleta en montaña, o si nos gusta el senderismo y buscamos nuevos caminos (incluso, para actividades como kayak, parapente o esquí).

Utilizando software libre y la API de Google Maps, Wikiloc hace la función de base de datos personal de localizaciones GPS. Desde cualquier acceso a Internet un usuario de GPS puede cargar sus datos GPS y al momento visualizar la ruta y puntos de paso o “waypoints” con distinta cartografía de fondo, incluidos servidores de mapas externos WMS (Web Map Service) o descargarlo a Google Earth para verlo en 3D. Al mimo tiempo se muestra el perfil de altura, distancia, desniveles acumulados y las fotos o comentario que el usuario quiera añadir.

El valor añadido de esta app es la posibilidad de compartir al instante estas localizaciones con los demás usuarios de Wikiloc, que pueden visualizar, valorar y descargar la ruta desde cualquier punto con conexión a Internet. En la aplicación para Android (Ilustración 1) se implementan las funcionalidades del servicio web y además, como servicio de pago,
se ofrece la opción de navegación sobre la ruta.

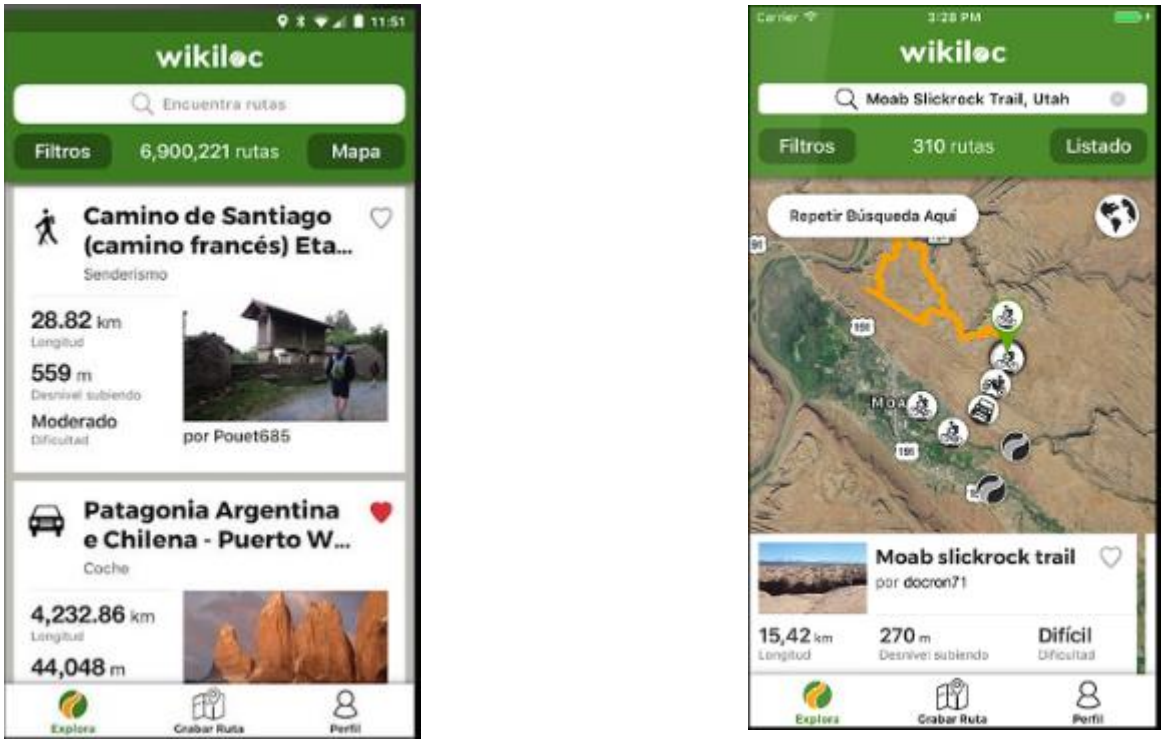

Ilustración 32: Pantallas de Elección de Rutas y Mapa de Wikiloc

GPX VIEWER

En el caso de esta app, también tendremos acceso a todo tipo de rutas, pero no está vinculada tanto al uso de ella para la actividad deportiva. La ventaja de esta, es que por medio de su compatibilidad con varios servicios online, podremos visualizar mapas y rutas registrados con esta misma app, con otra o incluso descargados desde Internet. Y también será capaz de leer una gran cantidad de ficheros, como *KML, GPX, KMZ O LOC*.

Y con todo ello, podremos combinar varias de esas rutas sobre un mismo mapa, diferenciarlas con colores, realizar anotaciones y comparar información entre ellas: altitud, velocidad, ritmo cardíaco o cualquier otro dato que se haya guardado mientras se registraba la ruta.

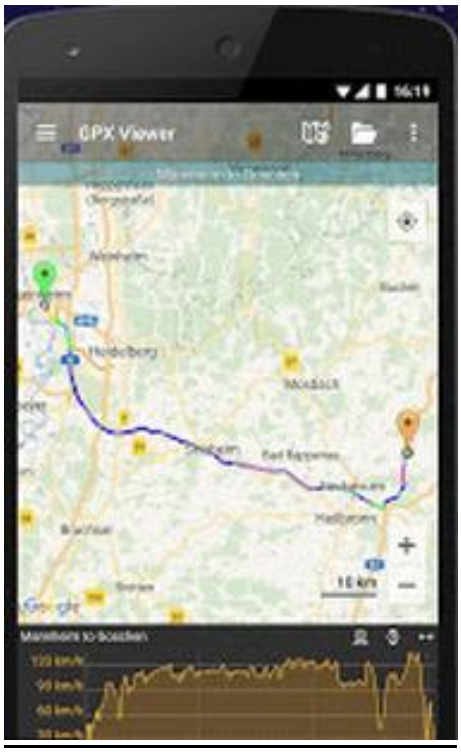

Ilustración 33: Mapa con ruta y sus detalles (GPX Viewer)

ORUXMAPS

Aquí tenemos otra completísima aplicación que nos sirvió de referente para establecer posibles objetivos de diseño en nuestra app, que por el contrario no fueron finalmente añadidos debido a su complejidad pero si ofrecen amplio margen de mejora y perfeccionamiento.

Con *ORUXMAPS* los amantes de las actividades al aire libre se verán completamente satisfechos. En ella, se podrán ver mapas, importarlos, guardar rutas y ver las estadísticas de las rutas realizadas por el usuario de las mismas.

ORUXMAPS está pensado ante todo para los aficionados al senderismo o a la BTT (bicicleta todo terreno); ello se demuestra por lo especializado de sus opciones y características, que cubren las necesidades que pueden surgir a la hora de trazar recorridos usando el GPS: añadir waypoints, tomar fotos, etc.

Permite registrar recorridos y guardarlos en los populares formatos GPX y KML nombrados anteriormente, los cuales hemos empleado en nuestra propia aplicación para la creación de las rutas fijas.

Asimismo, se pueden importar los mapas de otros usuarios y seguirlos. A ello se suman las prácticas estadísticas de esta app y la posibilidad de compartir los recorridos en otros servicios web como Everytrail, Gpsies o mapmytracks.

Ilustración 34: Pantalla de Inicio de Oruxmaps

5. Resultados y Discusión

En este apartado, se tratará de analizar aquello que hemos conseguido desarrollar, y discutir los resultados obtenidos, además de compararlos con otros proyectos similares.

Como ya se ha mencionado anteriormente, a la hora de realizar esta aplicación, hemos podido basarnos en otras apps de funciones similares, creadas para el uso comercial, y que son de gran ayuda para todas aquellas personas que aprovechan su tiempo libre disfrutando de este tipo de actividades.

Con todo ello, desde un primer momento sabíamos que, de la misma forma que se hizo con estos ejemplos encontrados, nosotros también deberíamos trabajar empleando la API del servidor de Google Maps, para poder utilizar su mapa a la hora de visualizar las rutas y crear las nuestras propias. Este puede considerarse uno de los puntos más valiosos de aprendizaje en programación durante mi proceso de desarrollo de este proyecto.

Además de ello, fue necesario aprender a introducir en diversas actividades las ya nombradas ListView y Arraylist, que sin duda, son métodos de desarrollo indispensables para la creación de una aplicación que pueda considerarse más avanzada.

El hecho de tener que adquirir el conocimiento de cómo emplear este tipo de métodos para la programación, me resultó realmente satisfactorio. Esto reflejaba que de verdad estaba siendo de gran ayuda el proyecto elegido, para poder empezar a entender, poquito a poco, el funcionamiento de esta ciencia tan extensa como es la programación.

Haciendo una ligera comparación con otros ejemplos de aplicaciones, creadas para poder obtener rutas de senderismo, o bien para ciclismo y motociclismo, desde el punto positivo de nuestra app, podemos decir que nuestra interfaz de mapeado, que mostrará las rutas, y nos permitirá señalar punto por punto las coordenadas de nuestro recorrido creado, cubrirá prácticamente el 100% de nuestra pantalla, a diferencia de otras apps, donde la amplitud de la zona del mapa se verá bastante reducida, lo que dificultará el manejo del mapa. Y esto no será debido a la falta de información de la ruta en la interfaz, ya que también podemos encontrar detalles del tramo elegido en nuestra aplicación. Con ello, tendremos una mayor visión de la situación geográfica de todo nuestro recorrido.

A su vez, el guardado de rutas en nuestra propia aplicación, gracias a la clase *FileInputStream,* tras la particular opción de poder crear los tramos que el usuario desee, es un complemento añadido al proyecto que se puede catalogar como el punto más fuerte de nuestro trabajo, y que no todas las apps de hoy en día ofrecen.

Por supuesto, ante todo tendremos propuestas de mejora, que en el siguiente apartado mencionaremos, ya que este sencillo prototipo deja mucho que desear respecto a las increíbles opciones que ofrece el mercado de apps, con las que tendremos plena confianza en nuestro dispositivo móvil a la hora de disfrutar de una agradable aventura, ya sea caminando o sobre ruedas.

6. Propuestas de Mejora

Añadir elementos de utilidad en la interfaz de ruta

Algunos detalles como la altitud, presión o grados/clima de la zona recorrida serían realmente importantes a la hora de decidir cual sería la elección de ruta más conveniente. Las aplicaciones Top ya disponen de este tipo de información, e incluso permiten indicar el pulso del usuario de forma constante si se le aplica un dispositivo medidor conectado al móvil.

Características de Google Maps Service

Tal y como ya sabemos, después de que Google Maps nos sacara de algún apuro, las funciones que este servidor nos ofrece, como obtener el camino perfecto, la duración exacta hasta alcanzar el lugar indicado, o incluso poder seguir paso a paso sus indicaciones curva tras curva, son claramente ideales. Y la mayoría de las apps disponibles actualmente hacen uso de estos recursos que, por supuesto, serán habilitados tras realizar el pago indispensable. En nuestro caso, ese paso evolutivo de la app que conlleva el registro con el servidor no ha sido establecido, lo cual refleja una sencillez importante en las características de la aplicación.

Posibilidad de compartir las rutas

Tanto la opción de compartir nuestras rutas creadas, como el poder enviar aquellas que sean de nuestro interés en la lista de rutas ya formadas (por ejemplo vía Whatsapp o Gmail), para que algún conocido pueda recibir la información del tramo desde fuera de la app, es una idea de lo más suculenta a la hora de obtener la aplicación perfecta en este ámbito.

Conseguir una app interactiva

Por supuesto, para alcanzar la perfección en este tipo de herramientas, que nos ayudarán a realizar planes sin fallo alguno, de lo más importante será siempre el poder recibir consejos y opiniones sobre cada una de las rutas. De esta manera, introducir la posibilidad de dar nuestra opinión, descripción y puntuaciones de nuestra experiencia en cada viaje, será clave para que así podamos estar seguros de que nuestra próxima aventura valdrá la pena.

7. Bibliografía

Recopilación de las referencias sobre publicaciones utilizadas en este proyecto:

Wikipedia: La Enciclopedia Libre.

[1] URL: https://es.wikipedia.org/wiki/Android_Studio

Academia Android

[2] URL: https://academiaandroid.com/android-studio-v1-caracteristicas-comparativa-eclipse/

Andr4all: Web Noticias de Android.

[3]URL: https://andro4all.com/2018/01/distribucion-android-2018

Wikipedia: La Enciclopedia Libre.

[4] URL: https://es.wikipedia.org/wiki/Android_Studio

Android Developer: Android Studio

[5] URL: https://developer.android.com/

Solicitud de API con Google Maps

[6] URL: https://console.developers.google.com/apis/

Deetz, Charles H (1944). Elementos de proyección de mapas y su aplicación a la construcción de mapas y cartas. Washington. Desde Wikipedia.

Aronsson, Lars. Wikipedia: La Enciclopedia Libre. Imagen del Mapa de Europa en Coordenadas UTM.

[7] URL: https://es.wikipedia.org/wiki/Sistema de coordenadas universal transversal de Mercator

Universo Android. Resolviendo dudas para el uso de métodos en Android Studio.

[8] URL: https://es.universoandroidhn.com

JC Mouse. Código Colectivo.

[9] URL: https://www.jc-mouse.net/android/introduccion-a-fragment

Universidad Poltécnica de Valencia. Máster en desarrollo de Aplicaciones Android. *Definición de Ciclo de vida de una actividad.*

4rsoluciones. (2013). Diferencias entre SDK, API, Biblioteca

[10] URL: http://www.4rsoluciones.com/blog/framework-sdk-biblioteca-api-cuales-son-las-diferencias-2/

Android Developers. Web oficial de Android para desarrolladores.

[11] URL: https://developer.android.com/guide/topics/manifest/manifest-intro?hl=es-419

La Vanguardia: Periódico diario

[12] URL: https:// www.lavanguardia.com/tecnologia/20170730/43155740258/aplicaciones-gps-viajes-para-no-perderse-apps-appszoom.html

Instinto Binario

[13] URL: https://instintobinario.com/manejar-ficheros-java/

StackOverFlow, Android Developer, Proyecto Simio, Android Curso. Youtube Tutorials. *Soporte para el desarrollo del trabajo.*

[14] URLs: https://es.stackoverflow.com

https://developer.android.com

http://www.proyectosimio.com

http://www.androidcurso.com