INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

INGENIERÍA EN CONTROL Y AUTOMATIZACIÓN

DESARROLLO DE UNA APLICACIÓN DE SMARTPHONE PARA LA UBICACIÓN DE TROLEBUSES Y AVISO DE ACERCAMIENTO

TESIS

QUE PARA OBTENER ELTÍTULO DE:

INGENIERO EN CONTROL Y AUTOMATIZACIÓN

PRESENTAN:

Almeida Montiel Saúl

Montaño Sánchez Orlando Jair

Vera Serrano Luis

DIRECTORES DE TESIS

M. en C. Mauricio Aarón Pérez Romero

M. en C. Antonio Obregón Tenorio

MÉXICO, D.F. DICIEMBRE 2015

INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

UNIDAD PROFESIONAL "ADOLFO LÓPEZ MATEOS"

T E M A D E T E S I S

QUE PARA OBTENER EL TÍTULO DE **INGENIERO EN CONTROL Y AUTOMATIZACIÓN**

POR LA OPCIÓN DE TITULACIÓN **TESIS COLECTIVA Y EXAMEN ORAL INDIVIDUAL**

DEBERA (N) DESARROLLAR

C. SAUL ALMEIDA MONTIEL

C. ORLANDO JAIR MONTAÑO SANCHEZ

C. LUIS VERA SERRANO

"DESARROLLO DE UNA APLICACIÓN DE SMARTPHONE PARA LA UBICACIÓN DE TROLEBUSES Y AVISO DE ACERCAMIENTO".

DESARROLLAR UNA APLICACIÓN PARA SMARTPHONE EN SISTEMA OPERATIVO ANDROID, EN LA CUAL SE VISUALICE EN TIEMPO REAL, LA UBICACIÓN DEL TROLEBÚS (EN LA RED DE TRANSPORTE DE LA UNIDAD PROFESIONAL ADOLFO LÓPEZ MATEOS), ASÍ COMO EL TIEMPO DE ARRIBO A LA UBICACIÓN DEL USUARIO. SE PLANTEA UTILIZAR ANDROID STUDIO PARA EL DESARROLLO DE LA APLICACIÓN, CON LA FINALIDAD DE OFRECER AL USUARIO DE DICHO SOFTWARE, UNA ALTERNATIVA DE SELECCIÓN EN CUANTO A LA FORMA DE LLEGAR O RETIRARSE DE LA UNIDAD ACADÉMICA.

- **CREAR UNA APLICACIÓN PARA OBTENER LAS COORDENADAS DEL TROLEBÚS.**
- **DESARROLLAR LA COMUNICACIÓN ENTRE EL SERVIDOR Y EL USUARIO PARA EL ENVIÓ DE LAS COORDENADAS DEL SISTEMA DE TRANSPORTE.**
- **GENERAR LA APLICACIÓN DE USUARIO MEDIANTE LA PLATAFORMA ANDROID STUDIO.**
- **IMPLEMENTACIÓN DE UN SISTEMA MÓVIL PARA ENVIAR DATOS DESDE EL SISTEMA DE TRANSPORTE AL USUARIO.**

CIUDAD DE MÉXICO, A 05 DE JULIO DE 2016.

A S E S O R E S

M. EN C. ANTONIO OBREGÓN TENORIO

M. EN C. MAURICIO AARÓN PÉREZ ROMERO

M. EN C. MIRIAM GÓMEZ ALVAREZ
JEFA DEL DEPARTAMENTO ACADÉMICO DE INGENIERÍA EN CONTROL Y AUTOMATIZACIÓN

Resumen

El siguiente prototipo tiene como finalidad señalar la posición tanto de un usuario del Servicio de Transportes Eléctricos (Trolebús) como cada unidad dentro del servicio, sobre un mapa (Vía *Google Maps*) que muestra el circuito recorrido por los trolebuses. Todo con el objetivo de proveer al usuario la información necesaria sobre la ubicación actual del transporte y mediante ello poder tomar la decisión de si hacer uso de este medio de transporte u optar por otra opción.

El prototipo basó su desarrollo en el sistema operativo Android, la razón de ello es por su gran cobertura dentro de la telefonía móvil, además de ser un entorno en el cual cualquier desarrollador tiene la facilidad de publicar las aplicaciones que fueron creadas por él mismo.
Para ello se hace uso de un software provisto por la misma compañía denominado Android Studio que permite el desarrollo de aplicaciones móviles haciendo uso del lenguaje de programación orientado a objetos (java).

Este prototipo basa su funcionamiento en la implementación del GPS, o Sistema de Posicionamiento Global (*Global Position System*) de cada Teléfono inteligente (*Smartphone*), colocado en cada unidad perteneciente al servicio de transporte, cuya posición es transmitida por medio de una aplicación (desarrollada en Android Studio) a otra aplicación que interprete la información recibida y la aplique para desplegar dentro del mapa todas las ubicaciones.

Como resultado de la ya antes descrito, el prototipo logró cumplir con los objetivos establecidos. El seguimiento en tiempo real de los trolebuses se visualizó desde cualquier teléfono celular con sistema operativo Android que tuviera la aplicación previamente instalada. Es importante destacar que pese a circunstancias ajenas al prototipo este se desempeñó correctamente haciéndolo confiable dentro del área para el que fue destinado.

Índice de contenido

Índice de Figuras

Índice de tablas.

Objetivo general

Desarrollar una aplicación para smartphone en sistema operativo Android, en la cual se visualice en tiempo real, la ubicación del trolebús (en la red de transporte de la Unidad Profesional Adolfo López Mateos), así como el tiempo de arribo a la ubicación del usuario. Se plantea utilizar Android Studio para el desarrollo de la aplicación, con la finalidad de ofrecer al usuario de dicho software, una alternativa de selección en cuanto a la forma de llegar o retirarse de la unidad académica.

Objetivo particular

- Crear una aplicación para obtener las coordenadas del trolebús.
- Desarrollar la comunicación entre el servidor y el usuario para el envío de las coordenadas del sistema de transporte.
- Generar la aplicación de usuario mediante la plataforma Andrioid Studio**.**
- Implementación de un sistema móvil para enviar datos desde el sistema de transporte al usuario.

Introducción

Se ha observado un problema dentro de la institución, que es la gran demanda que existe por parte de los estudiantes hacia el transporte a la hora de transportase a las diferentes unidades académicas del Instituto Politécnico Nacional ubicadas en Zacatenco. Uno de estos servicios es el Servicio de Transportes Eléctricos (Trolebús) que, de acuerdo con la comunidad estudiantil, la cantidad de autobuses que existen dentro del circuito no va acorde con la cantidad de usuarios que este tiene, provocando retrasos innecesarios a los docentes y estudiantes. Mediante la adaptación de dispositivos que permitan informar a los usuarios acerca del estado actual de dicho vehículo, el transeúnte podrá tomar decisiones de cómo se trasladará a su destino dentro de la unidad académica.

Tener un monitoreo del transporte trolebús mediante dispositivos GPS, de tal forma que el conocer y predecir la hora de llegada de dicho transporte sea una herramienta útil para el usuario. Todo esto por medio de la implementación de una aplicación cargada en un Teléfono Inteligente (Smartphone).

El objetivo de este prototipo es implementar una aplicación para detectar la ubicación de los trolebuses haciendo uso de dispositivos GPS, que permitan visualizar en tiempo real la ubicación del trolebús.

La mala administración en cuanto al transporte (Trolebús) ha creado una serie de problemáticas que se ven directamente afectadas en el control de tráfico, ejemplo de ello es la hora de llegada de los trolebuses la cual ha sido una constante problemática, debido a esto es imposible para el usuario tener certeza de la hora de arribo del Trolebús con el prototipo se quiere brindar la opción de elegir si el trolebús es la mejor forma de transportarse a la unidad académica.

Capítulo I. Antecedentes

1.1 Internet de las cosas.

Internet de las cosas (IdC), algunas veces denominado "Internet de los objetos", lo cambiará todo [Dave Evans, 2011, pág. 2]. Si bien puede parecer una declaración arriesgada, hay que tener en cuenta el impacto que Internet ha tenido sobre la educación, la comunicación, las empresas, la ciencia, el gobierno y la humanidad. Claramente Internet es una de las creaciones más importantes y poderosas de toda la historia de la humanidad.

Ahora se debe tener en cuenta que IdC representa la próxima evolución de Internet, que será un enorme salto en su capacidad para reunir, analizar y distribuir datos que puedan ser convertidos en información, conocimiento y en última instancia, sabiduría. En este contexto, IdC se vuelve inmensamente importante.

Ya están en marcha proyectos de IdC que prometen cerrar la brecha entre ricos y pobres, mejorar la distribución de los recursos del mundo para quienes más los necesitan y ayudar a comprender el planeta para que se pueda ser más proactivos y menos reactivos.

1.2 Evolución de la telefonía celular.

Actualmente las comunicaciones juegan un papel muy importante en la sociedad, ya que permite la constante interrelación y comunicación entre personas, sociedades, empresas y los demás actores del mundo moderno. Se puede decir, con total seguridad, que, sin comunicaciones, la vida como se conoce no podría existir. Con el paso del tiempo, la necesidad de estar cada vez más comunicados se hizo mayor, y es por ello que empresas y fabricantes relacionados al mundo de la telefonía se encuentran en una constante búsqueda por evolucionar y ofrecer cada vez mejores equipos y servicios.

Desde que, en 1854, año que podría definirse como la fecha de creación del mercado de las telecomunicaciones gracias a la invención del teléfono, no se ha dejado en ningún momento de innovar. Y si bien la invención del teléfono fue un acontecimiento histórico y que tuvo gran impacto a nivel mundial, que permitió reducir las grandes distancias levando a las personas la posibilidad de estar

comunicadas con mayor frecuencia y facilidad, no fue hasta la llegada de los teléfonos celulares en 1972, creación de la mente de Martin Cooper, que el teléfono se convirtió en un artefacto del cual es imposible desprenderse, alterando, además de la manera de comunicarse, la forma en que se comporta socialmente el individuo que lo utiliza.

Figura 1.2-1 Evolución de los teléfonos celulares.

Con el paso de los años, los *teléfonos celulares evolucionaron* (y lo siguen haciendo) de una manera drástica. Comenzaron siendo *"ladrillos"* (llamados de esta manera por su gran tamaño) y analógicos, para terminar, siendo pequeños y digitales con incontables e increíbles funciones.

1.3 La historia del teléfono celular.

El teléfono celular tiene sus inicios a principio de la Segunda Guerra Mundial, donde era una verdadera necesidad la comunicación a distancia, es por eso que Motorola creó un equipo llamado *Handie Talkie H12-16*. Es un equipo que permitía la comunicación a través de ondas de radio que en ese momento no superaban los 600 Khz.

Fue sólo cuestión de tiempo para que las dos tecnologías de Tesla y Marconi se unieran para crear la comunicación a través de radio-teléfonos: Martín Cooper, considerado como el padre de la telefonía

celular, fabricó el primer radio teléfono entre 1970 y 1973, en Estados Unidos, y en 1979 surgieron los primeros sistemas en el mercado de Tokio (Japón), fabricados por *NTT*. Europa no podía quedarse atrás y en 1981 se introdujo un sistema similar a AMPS (*Advanced Mobile Phone System*).

Ya en 1985 comenzaron a perfeccionar y a amoldar las características de este nuevo sistema. De esta forma en la década de los 80 se logró crear un equipo que utilizaba recursos similares a los de *Handie Talkie*, destinado a personas del sector empresarial que necesitaban una constante comunicación, es entonces donde se crea el teléfono móvil marcando un hito en la historia de los componentes inalámbricos, ya que con este equipo se podía hablar a cualquier hora y en cualquier lugar.

1.3.1 El primer teléfono celular.

El primer teléfono celular de la historia fue el *Motorola DynaTAC 8000X (Motorola, 2016)*, visto por primera vez en 1983. Tenía un peso de 780 gr y medía aproximadamente 33 cm x 9 cm x 4.5cm. Obviamente era analógico, y tenía un display pequeño. La batería tenía una durabilidad de no más de una hora hablando u 8 horas en stand-by. La calidad de sonido era muy mala, era pesado y anti estético, pero igualmente, determinadas personas pagaban su valor de USD $3,995 lo que lo convertía en un objeto de lujo al cual solamente podían acceder determinados grupos sociales.

Las primeras personas en utilizarlos fueron los hombres de negocios, ejecutivos y personal de alto poder adquisitivo, principalmente porque el desarrollo socioeconómico de una empresa necesita una comunicación eficaz, comunicación con proveedores, clientes, empleados, gobiernos y organismos reguladores. El uso de este servicio tenía un costo elevado ya que al haber falta de competencia los precios no bajaban y no había mejoras técnicas.

En 1984, se vendieron alrededor de 900.000 teléfonos, sobrepasando considerablemente la cantidad estimada.

1.3.2 Evolución de los sistemas de comunicación celular.

Primera Generación:

- NMT 450 (1981) Nordiska Mobil Telephongruppen, (Noruega, Suecia, Finlandia y Dinamarca)
- AMPS 800 (1983), USA, Chicago
- TACS 900 (1985) Gran Bretaña

- NMT 900 (1986)

Segunda Generación:

- GSM 900 (1991) ETSI
- DAMPS 800 (1991)
- Mejora IS-54
- IS-95 CDMA
- DCS 1800 (1992) ETSI
- GPRS Generación 2.5

Tercera Generación

- UMTS/IMT 2000

1.3.3 Sistemas de comunicación celular de primera generación.

El Sistema de Comunicaciones de Acceso Total (TACS, *Total Access Communications System*) es un sistema de comunicaciones para telefonía móvil celular dúplex en la banda de 900 MHz. El precursor del sistema TACS es el sistema AMPS (*American Mobile Phone System*), desarrollado en los EE.UU. por los laboratorios Bell en la década de los 70, y puesto en servicio en la primera mitad de la década de los 80. El sistema TACS fue desarrollado por el Reino Unido, adaptando el sistema AMPS a los requisitos europeos (especialmente en los aspectos de banda de frecuencia y canalización), y puesto en servicio en 1985.

En el Reino Unido se concedieron dos licencias para operar cada una con su red propia. Para ello, la banda original (890-915 MHz y 935-960 MHz) de 1000 canales se dividió en dos segmentos de 300 canales cada uno, dejando la subbanda 905-915 MHz y 950-960 MHz para la introducción posterior del sistema GSM. Posteriormente, se amplió la banda añadiendo los rangos 872-890 MHz y 917-935 MHz para otorgar la capacidad requerida. Esta nueva banda toma la denominación de E-TACS (Extended TACS).

A principio de esta década de los 90, otros países como Austria, Italia y España adoptaron también este sistema. Algo importante que se debe tener en cuenta es que el estándar TACS define tan sólo el protocolo de acceso radio entre una estación móvil y su correspondiente estación base. La gestión de la movilidad o lo que es igual, las facilidades de "*handover*" y "*roaming*" soportadas por el sistema, así

como la estructura y comunicaciones entre los distintos elementos de la red quedan a criterio del fabricante.

1.3.4 Sistemas de comunicación celular de segunda generación.

Desde principios de los 80's, después de que el NMT comenzase su operación comercial, se hizo evidente para algunos países europeos que los sistemas analógicos existentes tenían limitaciones. En primer lugar, la demanda potencial para los servicios móviles, aunque estaba siendo sistemáticamente subestimada, era mayor que la capacidad de las redes analógicas existentes. En segundo lugar, los diferentes sistemas existentes no ofrecían compatibilidad para sus usuarios: un terminal TACS no puede acceder a una red NMT ni viceversa. Lo que, es más, el diseño de un sistema celular nuevo requiere tal inversión que ningún país europeo puede acometer tal inversión de forma independiente si el único retorno esperado está sólo en su propio mercado nacional. Todas estas circunstancias apuntaban hacia el diseño de un sistema nuevo, desarrollado en común entre varios países.

El mayor requisito para un sistema de radio común es un ancho de banda común. Esta condición se cumplía unos años antes, en 1978, cuando se decidió reservar una banda de frecuencia de dos veces 25 MHz en torno a los 900 MHz para comunicaciones móviles en Europa.

La necesidad estaba clara y el mayor obstáculo había sido eliminado. Sólo quedaba organizar el trabajo. El mundo de las telecomunicaciones en Europa siempre estuvo dominado por la estandarización. La CEPT (*Conférence Européenne des Postes et Télécommunications*) es un foro de estandarización que, en los primeros V80, incluía a las Administraciones europeas de Correos y Telecomunicaciones de más de 20 países. Todas estas circunstancias llevaron a la creación en 1982 de un nuevo organismo de estandarización en la CEPT, cuya labor consistía en especificar un sistema único de telecomunicaciones para Europa, en 900 MHz. El recién creado Groupe Spécial Mobile (GSM) tuvo su primera reunión en diciembre de 1982, en Estocolmo.

En 1990, bajo petición del Reino Unido, se añadió a los objetivos del grupo de estandarización la especificación de una versión de GSM adaptada a la banda de frecuencias de 1800 MHz, con una asignación de 2 veces 75 MHz. Esta variante que se conoció con el nombre de DCS1800 Digital Cellular System 1800) tiene como objetivo proporcionar mayor capacidad en áreas urbanas. La elaboración del estándar GSM llevó casi una década.

1.3.5 General Packet Radio Service (GPRS)

La red GSM prevé unos servicios de transmisión de datos desde la fase inicial (fase 1). Sin embargo, se trata de servicios con modalidad de transferencia por conmutación del circuito, es decir, donde la red, una vez establecida la conexión física de cabo a rabo entre dos usuarios, dedica los recursos propios hasta que no es solicitado expresamente el establecimiento de la conexión, independientemente del hecho de que los dos usuarios se intercambien datos Página69 durante todo el tiempo de conexión. Esta modalidad de transferencia es óptima sólo en el caso en que los dos usuarios tengan que intercambiarse una cantidad significativa de datos (transferencia de ficheros o archivos); resulta ineficiente en cuanto los datos a intercambiarse son de pequeña entidad o bien, en el caso más frecuente, el tráfico de datos es de tipo interactivo o transitorio, es decir, el tiempo de uso efectivo de los recursos de la red supone sólo una parte con respecto al tiempo total de conexión (como, por ejemplo, la navegación en Internet a través de la *World Wide Web*).

1.3.6 Sistemas de comunicación celular de tercera generación.

El área de las comunicaciones móviles, junto con Internet, es la de crecimiento más rápido dentro del sector de las telecomunicaciones. En todo el mundo, a finales de 1999, había 450 millones de usuarios de telefonía móvil celular y la previsión es alcanzar los mil millones en el año 2004, una cifra similar a la de usuarios de Internet, de los cuales se espera que, al menos, unos 400 millones compartirán el uso de ambas redes, utilizando el teléfono móvil como el medio preferido de acceso a la Red. Esta tendencia es lógica si se tiene en cuenta que el número de móviles con capacidad multimedia y de navegación será muy superior al de ordenadores personales, superando incluso a las líneas de telefonía fija que existen en la actualidad (Figura 1.2:1). La explicación a este crecimiento del mercado se encuentra en el rápido avance de la tecnología, a las oportunidades comerciales que se asocian con la movilidad personal, y a la bajada del precio de los terminales y de las tarifas de conexión y por tráfico.

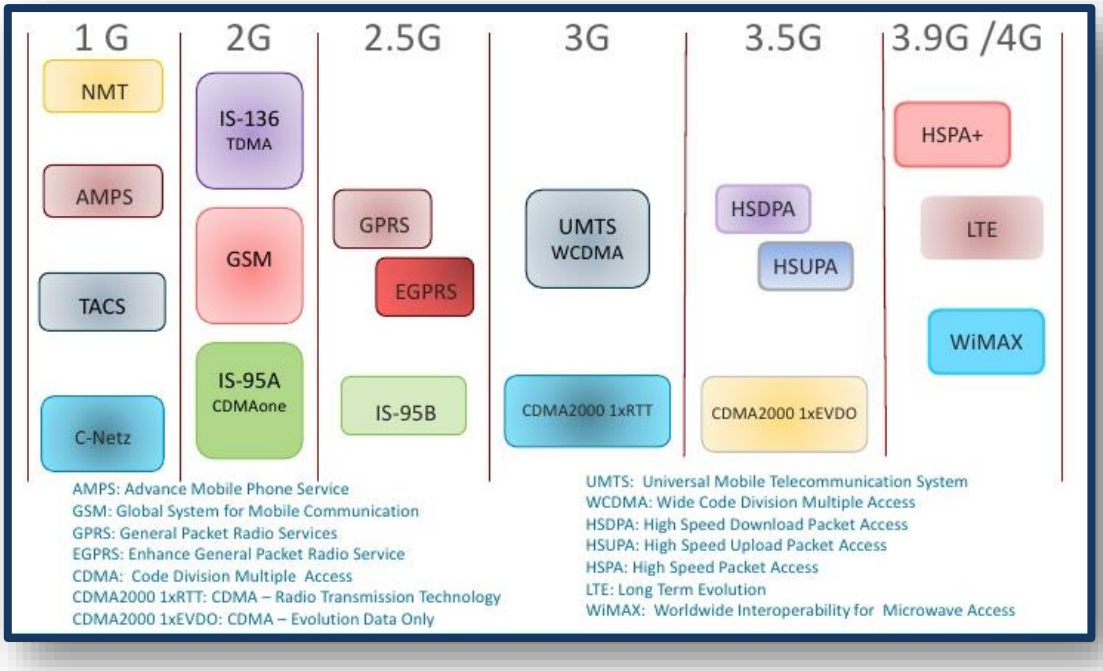


Figura 1.3-1 Tecnología sin hilos

Este crecimiento tan espectacular y rápido lleva aparejado el desarrollo e implantación de diferentes tecnologías -analógicas como FDMA y digitales como TDMA y CDMA- y estándares - AMPS, TDM, NMT, TACS, GSM, DECT o PHS-, muchas veces coexistiendo en el mismo país, lo que hace que resulte complicado, además de costoso, dotar de movilidad universal a los usuarios en sus desplazamientos. Es por ello que dentro de la UIT (Unión Internacional de Telecomunicaciones), un organismo perteneciente a las Naciones Unidas en el que organizaciones públicas y privadas coordinan las redes de telecomunicaciones y la creación de servicios en todo el mundo, se ha venido desarrollando una nueva solución denominada, en el año 1996, IMT-2000. El 2000 corresponde al año en que deberían estar definidos los estándares, entre los que se incluye UMTS (*Universal Mobile Telecommunications System*), que estará plenamente operativo antes del 2005, aunque algunas fases se pondrán en marcha mucho antes, como ha sucedido con GSM.

1.3.7 Mayor velocidad y mejores servicios

La tercera generación de móviles, denominada 3G, evoluciona para integrar todos los servicios ofrecidos por las distintas tecnologías y redes actuales, como GSM, TACS, DECT, RDSI e Internet, utilizando cualquier tipo de terminal, sea un teléfono fijo, inalámbrico o celular, tanto en un ámbito

profesional como doméstico, ofreciendo una mayor calidad de los servicios y soportando la personalización por el usuario y los servicios multimedia móviles en tiempo real. La velocidad de transferencia de datos que la UIT requiere en su solución IMT-2000 va desde los 144 kbit/s sobre vehículos a gran velocidad hasta los 2 Mbit/s sobre terminales en interiores de edificios (cifra al menos 60 veces superior a la que se tenía hasta hace poco utilizando un módem y la RTC), pasando por los 384 kbit/s para usuarios móviles en el extrarradio, o vehículos a baja velocidad.

1.4 Sistemas Operativos

Una computadora moderna consta de uno o más procesadores, una memoria principal, discos, impresoras, un teclado, un ratón, una pantalla o monitor, interfaces de red y otros dispositivos de entrada/salida. En general es un sistema complejo. Si todos los programadores de aplicaciones tuvieran que comprender el funcionamiento de todas estas partes, no escribirían código alguno. Es más: el trabajo de administrar todos estos componentes y utilizarlos de manera óptima es una tarea muy desafiante. Por esta razón, las computadoras están equipadas con una capa de software llamada sistema operativo, cuyo trabajo es proporcionar a los programas de usuario un modelo de computadora mejor, más simple y pulcro, así como encargarse de la administración de todos los recursos antes mencionados.

El programa con el que los usuarios generalmente interactúan se denomina *shell*, cuando está basado en texto, y GUI (*Graphical User Interface; Interfaz gráfica de usuario*) cuando utiliza elementos gráficos o iconos. En realidad, no forma parte del sistema operativo, aunque lo utiliza para llevar a cabo su trabajo.

El programa de interfaz de usuario, *shell* o GUI, es el nivel más bajo del software en modo usuario y permite la ejecución de otros programas, como un navegador Web, lector de correo electrónico o reproductor de música. Estos programas también utilizan en forma intensiva el sistema operativo.

Se ejecuta directamente sobre el *hardware* y proporciona la base para las demás aplicaciones de *software*.

Los sistemas operativos se pueden ver desde dos puntos de vista: como administradores de recursos y como máquinas extendidas. En el punto de vista correspondiente al administrador de recursos, la

función del sistema operativo es administrar las distintas partes del sistema en forma eficiente. En el punto de vista correspondiente a la máquina extendida, la función del sistema operativo es proveer a los usuarios abstracciones que sean más convenientes de usar que la máquina actual. Estas abstracciones incluyen los procesos, espacios de direcciones y archivos.

Los sistemas operativos tienen una larga historia, empezando desde los días en que reemplazaron al operador, hasta los sistemas modernos de multiprogramación. Entre los puntos importantes se tienen a los primeros sistemas de procesamiento por lotes, los sistemas de multiprogramación y los sistemas de computadora personal. Como los sistemas operativos interactúan de cerca con el hardware, es útil tener cierto conocimiento del hardware de computadora para comprenderlos. Las computadoras están compuestas de procesadores, memorias y dispositivos de E/S. Estas partes se conectan mediante buses. Los conceptos básicos en los que se basan todos los sistemas operativos son los procesos, la administración de memoria, la administración de E/S, el sistema de archivos y la seguridad. El corazón de cualquier sistema operativo es el conjunto de llamadas al sistema que puede manejar. Estas llamadas indican lo que realmente hace el sistema operativo.

1.4.1 Visión general y entorno de desarrollo de Android.

La telefonía móvil está cambiando la sociedad actual de una forma tan significativa como lo ha hecho Internet. Esta revolución no ha hecho más que empezar, los nuevos terminales ofrecen unas capacidades similares a las de un ordenador personal, lo que permite que puedan ser utilizados para leer correos o navegar por Internet. Pero a diferencia de un ordenador, un teléfono móvil siempre está en el bolsillo del usuario. Esto permite un nuevo abanico de aplicaciones mucho más cercanas al usuario. De hecho, muchos autores coinciden en que el nuevo ordenador personal del siglo veintiuno será un terminal móvil.

El lanzamiento de Android como nueva plataforma para el desarrollo de aplicaciones móviles ha causado una gran expectación y está teniendo una importante aceptación, tanto por los usuarios como por la industria. En la actualidad se está convirtiendo en la alternativa estándar frente a otras plataformas como *iPhone, Windows Phone* o *BlackBerry*.

A lo largo de este capítulo se verán las características de Android que lo hacen diferente de sus competidores. Se explicará también cómo instalar y trabajar con el entorno de desarrollo (Android Studio + Android SDK).

Como se ha comentado, existen muchas plataformas para móviles (*iPhone, Symbian, Windows Phone, BlackBerry, Palm, Java Mobile Edition, Linux Mobile (LiMo)*, etc.); sin embargo, Android presenta una serie de características que lo hacen diferente. Es el primero que combina, en una misma solución, las siguientes cualidades:

• Plataforma realmente abierta. Es una plataforma de desarrollo libre basada en Linux y de código abierto. Una de sus grandes ventajas es que se puede usar y "customizar" el sistema sin pagar royalties.

• Portabilidad asegurada. Las aplicaciones finales son desarrolladas en Java, lo que asegura que podrán ser ejecutadas en una gran variedad de dispositivos, tanto presentes como futuros. Esto se consigue gracias al concepto de máquina virtual.

• Arquitectura basada en componentes inspirados en Internet. Por ejemplo, el diseño de la interfaz de usuario se hace en XML, lo que permite que una misma aplicación se ejecute en un móvil de pantalla reducida o en un *netbook*.

• Filosofía de dispositivo siempre conectado a Internet.

• Gran cantidad de servicios incorporados. Por ejemplo, localización basada tanto en GPS como en redes, bases de datos con SQL, reconocimiento y síntesis de voz, navegador, multimedia, etc.

• Aceptable nivel de seguridad. Los programas se encuentran aislados unos de otros gracias al concepto de ejecución dentro de una caja que hereda de Linux. Además, cada aplicación dispone de una serie de permisos que limitan su rango de actuación (servicios de localización, acceso a Internet, etc.).

• Optimizado para baja potencia y poca memoria. Por ejemplo, Android utiliza la Máquina Virtual Dalvik. Se trata de una implementación de Google de la máquina virtual de Java optimizada para dispositivos móviles.

• Alta calidad de gráficos y sonido. Gráficos vectoriales suavizados, animaciones inspiradas en *Flash*, gráficos en 3 dimensiones basados en OpenGL. Incorpora los codees estándar más comunes de audio y vídeo, incluyendo H.264 (AVC), MP3, AAC, etc.

Como se ha visto, Android combina características muy interesantes. No obstante, la pregunta del millón es, ¿se convertirá Android en el estándar de sistema operativo (S O.) para móviles? Para contestar a esta pregunta habrá que esperar un tiempo para ver la evolución del iPhone de Apple y cuál es la respuesta de Windows con el lanzamiento de su nuevo S.O. para móviles.

En conclusión, Android ofrece una forma sencilla y novedosa de implementar potentes aplicaciones para móviles.

Google adquiere Android Inc. en el año 2005. Se trataba de una pequeña compañía que acababa de ser creada, orientada a la producción de aplicaciones para terminales móviles. Ese mismo año empiezan a trabajar en la creación de una máquina virtual Java optimizada para móviles (*Dalvik VM*).

En el año 2007 se crea el consorcio Handset Alliance1 con el objetivo de desarrollar estándares abiertos para móviles. Está formado por Google, Intel, Texas Instruments, Motorola, T-Mobile, Samsung, Ericsson, Toshiba, Vodafone, NTT DoCoMo, Sprint Nextel y otros. Una pieza clave de los objetivos de esta alianza es promover el diseño y difusión de la plataforma Android. Sus miembros se han comprometido a publicar una parte importante de su propiedad intelectual como código abierto bajo la licencia Apache v2.0.

En noviembre del 2007 se lanza una primera versión del *Android SDK*. Al año siguiente aparece el primer móvil con Android (*T-Mobile G1*). En octubre, Google libera el código fuente de Android, principalmente bajo licencia de código abierto Apache (licencia GPL v2 para el núcleo). Ese mismo mes, se abre *Android Market* para la descarga de aplicaciones. En abril del 2009, Google lanza la versión 1.5 del SDK que incorpora nuevas características como el teclado en pantalla. A finales del 2009 se lanza la versión 2.0 y durante el 2010 las versiones 2.1, 2.2 y 2.3.

Durante el año 2010, Android se consolida como uno de los sistemas operativos para móviles más utilizados, con resultados cercanos al iPhone, e incluso superando al sistema de Apple en E.U.A.

En el 2011 se lanzan la versión 3.x específica para tabletas y 4.x tanto para móviles como para tabletas. Durante este año, Android se consolida como la plataforma para móviles más importante, alcanzando una cuota de mercado superior al 50%. En 2012, Google cambia su estrategia en su tienda de descargas online, remplazando *Android Market* por *Google Play Store*, donde en un solo portal unifica la descarga de aplicaciones como de contenidos.

1.5 Historia del GPS

1.5.1 Sistema TRANSIT

Primer sistema de navegación basado en satélites. Entrada en servicio en 1965.
Al principio de los 60 los departamentos de defensa, transporte y la agencia espacial norteamericanas (DoD, DoT y NASA respectivamente) tomaron interés en desarrollar un sistema para determinar la posición basado en satélites.
El sistema debía cumplir los requisitos de globalidad, abarcando toda la superficie del globo; continuidad, funcionamiento continuo sin afectarle las condiciones atmosféricas; altamente dinámicas, para posibilitar su uso en aviación y precisión.
Esto llevó a producir diferentes experimentos como el *Timation* y el sistema 621B en desiertos simulando diferentes comportamientos.

El sistema TRANSIT estaba constituido por una constelación de seis satélites en órbita polar baja, a una altura de 1074 Km. Tal configuración conseguía una cobertura mundial pero no constante. La posibilidad de posicionarse era intermitente, pudiéndose acceder a los satélites cada 1.5 h. El cálculo de la posición requería estar siguiendo al satélite durante quince minutos continuamente.

Constelación TRANSIT: TRANSIT trabajaba con dos señales en dos frecuencias, para evitar los errores debidos a la perturbación ionos feérica. El cálculo de la posición se basaba en la medida continua de la desviación de frecuencia Doppler de la señal recibida y su posterior comparación con tablas y

gráficos. El error de TRANSIT estaba en torno a los 250 m. Su gran aplicación fue la navegación de submarinos y de barcos.

NAVSTAR. Sistema de posicionamiento global (GPS)

TRANSIT tenía muchos problemas. La entonces URSS tenía un sistema igual que el TRANSIT, de nombre TSICADA. Había que dar un gran salto. La guerra fría fomentaba invertir unos cuantos billones de pesetas en un revolucionario sistema de navegación, que dejara a la URSS definitivamente atrás.

Se concibió un sistema formado por 24 satélites en órbita media, que diera cobertura global y continua. ROCKWELL (California) se llevó uno de los contratos más importantes de su época, con el encargo de 28 satélites por 170.000.000.000 (ciento setenta mil millones) de pesetas.

El primer satélite se lanzó en 1978, y se planificó tener la constelación completa ocho años después. Unido a varios retrasos, el desastre de la lanzadera Challenger paró el proyecto durante tres años. Por fin, en diciembre de 1983 declaró la fase operativa inicial del sistema GPS. El objetivo del sistema GPS era ofrecer a las fuerzas de los EE.UU. la posibilidad de posicionarse (disponer de la posición geográfica) de forma autónoma o individual, de vehículos o de armamento, con un coste relativamente bajo, con disponibilidad global y sin restricciones temporales. La iniciativa, financiación y explotación corrieron a cargo del Departamento de Defensa de los EE.UU. (DoD), el GPS se concibió como un sistema militar estratégico.

En 1984 un vuelo civil de Korean Airlines fue derribado por la Unión Soviética al invadir por error su espacio aéreo. Ello llevó a la administración Reagan a ofrecer a los usuarios civiles cierto nivel de uso de GPS, llegando finalmente a ceder el uso global y sin restricciones temporales, de esta forma se conseguía un retorno a la economía de los EE.UU. inimaginables unos años atrás. Además, suponía un gran liderazgo tecnológico originando un vertiginoso mercado de aplicaciones.

Desde 1984, con muy pocos satélites en órbita, aparecieron tímidamente fabricantes de receptores GPS destinados al mundo civil (*Texas Instruments y Trimble Navigation*).

1.5.2 El GPS hoy

Hoy en día el GPS supone un éxito para la administración y economía americana no interesando a nadie que se reduzca la inversión en el sistema, sino todo lo contrario. La política de la administración de

EE.UU. es mantener coste 0 para el usuario el sistema GPS, potenciar sus aplicaciones civiles a la vez que se mantiene el carácter militar.

Las aplicaciones disponibles se orientan a principalmente a sistemas de navegación y aplicaciones cartográficas: topografía, cartografía, geodesia, sistema de información geográfica (GIS), mercado de recreo (deportes de montaña, náutica, expediciones de todo tipo, etc.), patrones de tiempo y sistemas de sincronización, aplicaciones diferenciales que requieran mayor precisión además de las aplicaciones militares y espaciales.

En cuanto al reparto del mercado los más importantes son la navegación marítima, la aérea y la terrestre.

Con una flota de 46 millones embarcaciones en todo el mundo, de los que el 98% son de recreo, la navegación marítima supone un mercado nada despreciable. Recreo, pesqueros, mercantes, petroleros, dragados y plataformas petrolíferas son perfectos candidatos al uso del GPS. El volumen de venta de equipos GPS en está en torno a los 300 millones de dólares anuales.

En cuanto a la navegación aérea con unos 300.000 aviones en todo el mundo. El equipamiento de GPS para navegación intercontinental o entre aeropuertos tiene una penetración anual del 5% (aproximadamente unas 15.000 unidades). Sin embargo en aproximación el GPS no tiene la suficiente integridad y precisión aunque la FAA está financiando el proyecto WAAS (*Wide Area Augmentation System*) que refuerza el sistema GPS y será útil para aproximaciones de clase I (en E.U.A).

Pero el auténtico mercado del GPS en el mundo es la navegación terrestre. Con 435 millones de turismos y 135 millones de camiones es el más amplio mercado potencial de las aplicaciones comerciales del GPS. De hecho, el crecimiento de equipamiento de GPS mundial es en torno a los 2.000 millones de dólares anuales, lo que lleva a una penetración del 4% en el año 2001. Entre las aplicaciones con más desarrollo contamos con sistemas de navegación independiente, sistemas de seguimiento automático, control de flotas, administración de servicios, etc. Solo en los EE.UU existen 25.000 autobuses equipados con GPS y en Japón hay ya un millón y medio de vehículos privados que cuentan con sistema GPS en su equipamiento.

En España el mercado del GPS está en plena expansión habiendo alcanzado en 1998 las 200 unidades para aplicaciones topográficas y geodésicas, unas 300 para aeronáutica, más de 3.500 para la náutica y alrededor de 4.000 unidades OEM para aplicaciones terrestres.

1.5.3 Descripción del funcionamiento general del GPS

El GPS funciona mediante unas señales de satélite codificadas que pueden ser procesadas en un receptor de GPS permitiéndole calcular su posición, velocidad y tiempo.

Se utilizan cuatro señales para el cálculo de posiciones en tres dimensiones y del ajuste del reloj del receptor en el bloque receptor.

Segmento espacial
El segmento del espacio del sistema está formado por los satélites GPS que mandan señales de radio desde el espacio.

Nominalmente la constelación operacional de GPS consiste en 24 satélites que orbitan alrededor de la tierra en 12 horas.

Normalmente hay más número de satélites ya que se ponen en órbita unidades nuevas para reponer satélites antiguos que tienen una vida media aproximada de siete años y medio. Hasta la actualidad ha habido tres generaciones de satélites, los Block I (actualmente inoperativos), Block II (9 satélites entre 1989 y 1990 y 19 adicionales hasta el 1997) y Block IIR (un satélite en 1998). En enero de 1999 orbitaban 27 satélites GPS en total.

Los satélites están situados a 20.180 Km de altura desplazándose a una velocidad de 14.500 Km./h. Las órbitas son casi circulares y se repite el mismo recorrido sobre la superficie terrestre (mientras la tierra rota a su vez sobre si misma) de esta forma en prácticamente un día (24 horas menos 4 minutos) un satélite vuelve a pasar sobre el mismo punto de la tierra. Los satélites quedan situados sobre 6 planos orbitales (con un mínimo de 4 satélites cada uno), espaciados equidistantes a 60 grados e inclinados unos 15 grados respecto al plano ecuatorial. Esta disposición permite que desde cualquier punto de la superficie terrestre sean visibles entre cinco y ocho satélites

1.5.4 Segmento de control

El segmento de control consiste en un sistema estaciones de seguimiento localizadas alrededor del mundo.

La estación maestra de control (MCS) está situada en Falcon AFB en Colorado Spring. Las estaciones de control miden las señales procedentes de los satélites y son incorporadas en modelos orbitales para cada satélite. Los modelos calculan datos de ajuste de órbita (efemérides) y correcciones de los relojes de cada satélite. La estación maestra envía las efemérides y correcciones de reloj a cada satélite. Cada satélite envía posteriormente subconjuntos de estas informaciones a los receptores de GPS mediante señales de radio.

1.5.5 Segmento de usuario

El segmento de usuario lo forman los receptores y la comunidad de usuarios. Los receptores convierten las señales recibidas de los satélites en posición, velocidad y tiempo estimados. Se requieren cuatro satélites para el cálculo de la posición en cuatro dimensiones X, Y, Z y tiempo. Los receptores son utilizados para navegación, posicionamiento, estimaciones temporales y otras investigaciones.

La navegación en tres dimensiones es la función principal del GPS. Se construyen receptores GPS para aeroplanos, embarcaciones, vehículos terrestres y equipos portátiles de pequeño tamaño.

El posicionamiento preciso es posible usando receptores en posiciones de referencia proporcionando datos de corrección y posicionamiento relativo a receptores remotos. Vigilancia, control geodésico y estudios de las placas tectónicas son ejemplos.

Las aplicaciones de tiempo y estabilización de frecuencia se basan en la precisión de los relojes que incorporan los satélites y que son monitorizados continuamente por las estaciones de control. Los satélites actuales incorporan cuatro relojes atómicos, dos de Rubidio y otros dos de Cesio que ofrecen una estabilidad de frecuencia equivalente a un error de un segundo en 30.000 años. (Hay que tener en cuenta que un error de 30ns Provoca un error de 30cm). Los observatorios astronómicos, sistemas de telecomunicaciones, sincronización de centrales eléctricas y laboratorios de certificación pueden obtener señales de tiempo y frecuencia de alta precisión mediante receptores especiales de GPS. Las señales de GPS han sido utilizadas para medir parámetros atmosféricos.

1.6 Sumario

El presente capítulo muestra un panorama a cerca de la historia del sistema de telefonía celular, con la finalidad de dar una idea de la evolución de este dispositivo y las redes que facilitan la comunicación entre las personas. De la misma manera se hace mención de los sistemas operativos que hoy en día gobiernan la telefonía celular, haciendo a este tipo de dispositivos más versátiles y eficientes para su uso diario. Se trató la historia de los dispositivos GPS, desde su uso puramente militar, hasta el uso comercial para todo público.

Todo esto se explicó con el fin de dar una introducción al siguiente capítulo donde se tratarán las diferentes herramientas que se utilizarán para el desarrollo de una aplicación prototipo.

Capítulo II. Herramientas y recursos para el desarrollo de la aplicación.

2.1 ¿Por qué *Android*?

El código de *Android* es abierto: *Google* liberó *Android* bajo licencia Apache. Cualquier persona puede realizar una aplicación para *Android*.

Hoy día hay más de 650.000 aplicaciones disponibles para teléfonos Android, aproximadamente 2/3 son gratis. Además, la libertad de código permite adaptar Android a bastantes otros dispositivos además de teléfonos celulares. Está implantado en *Tablets*, GPS, relojes, microondas… incluso hay por internet una versión de *Android* para PC.

El sistema *Android* es capaz de hacer funcionar a la vez varias aplicaciones y además se encarga de gestionarlas, dejarlas en modo suspensión si no se utilizan e incluso cerrarlas si llevan un periodo determinado de inactividad. De esta manera se evita un consumo

2.1.1 Comparativa de los sistemas operativos

En este apartado se describirán las características de las principales plataformas móviles disponibles en la actualidad. Dado la gran cantidad de datos que se indican, se ha utilizado una tabla para representar la información. De esta forma resulta más sencillo comparar las distintas plataformas.

Tabla 2.1–1 Android y los Sistemas Operativos más usados

			
Compañía	Apple	Open Handset Alliance	Windows
Núcleo del SO	Mac OS X	Linux	Windows CE
Familia CPU soportada	ARM	ARMS, MIPS, Power, x86	ARM

Lenguaje de programación	Objective C, C++	Java, C ++	C#, muchos
Licencia de software	Propietaria	Software libre y abierto	Propietaria
Año de lanzamiento	2007	2008	2010
Motor del navegador	Webkit	Webkit	Pocket internet Explorer
Soporte flash	No	Si	No
HTML 5	Si	Si	Parcial
Tienda de aplicaciones	App store	Google Play	Windows Marketplace
Número de aplicaciones	400.00	300.000	50.000
Coste publicar	$ 99 al año	$ 25 una vez	$ 99 al año
Plataforma de desarrollo	Mac	Mac, Linux, Windows	Windows
Interfaz personalizable	No	Si	Si
Actualizaciones automáticas del SO	Si	Depende del fabricante	Depende del fabricante
Variedad de dispositivos	Modelo único	Muy alta	Baja
Aplicaciones nativas	Si	Si	No

Otro aspecto fundamental a la hora de comparar las plataformas móviles es su cuota de mercado. En la siguiente gráfica se puede ver un estudio realizado por la empresa Gratner Group, donde se muestra la evolución del mercado de los sistemas operativos para móviles según el número de terminales vendidos. Destacando: el importante descenso de ventas de la plataforma Symbian de Nokia; el declive continuo de BlackBerry; como la plataforma de Windows parece que no despega; como Apple tiene afianzada una cuota de mercado y ha experimentado un importante repunte a finales del 2011. Finalmente, se destaca el gran ascenso de la plataforma Android, que le ha permitido alcanzar en dos años una cuota de mercado superior al 50%.

Se han realizado otros tipos de estudios que miden la actividad de los usuarios en Internet. En estos casos se comprueba como los usuarios de Android e iPhone son los más activos, mientras que los usuarios con otras plataformas, como Symbian, utilizan sus terminales de forma más convencional.

2.2 Java

Es un lenguaje de programación de alto nivel, orientado a objetos, el cual tiene la capacidad de ser ejecutado en una gran cantidad de dispositivos y electrodomésticos ya que este fue el fin para el cual fue creado, en su mayor parte la sintaxis de Java está influenciada por el popular lenguaje C y por Visual Basic, aunque deja de lado algunas características de bajo nivel que tiene C como son el manejo de memoria y los apuntadores (punteros). Para las gestiones de manejo de memoria Java utiliza Algo llamado Recolector de Basura (*garbage collector*) el cual se encarga de limpiar la memoria.

Java posee todas las características del paradigma de la programación orientada a objetos herencia, encapsulamiento, abstracción, polimorfismo, modularidad entre otras.

2.3 Android SDK

El *SDK de Android*, incluye un conjunto de herramientas de desarrollo. Comprende un depurador de código, biblioteca, un simulador de teléfono basado en QEMU, documentación, ejemplos de código y tutoriales. Las plataformas de desarrollo soportadas incluyen Linux (cualquier distribución moderna), Mac OS X 10.4.9 o posterior, y Windows XP o posterior. La plataforma integral de desarrollo (IDE, Ambiente de desarrollo integrado) soportada oficialmente es *Android Studio* junto con el complemento ADT (Herraminetas plugin de desarrollo de Android), aunque también puede utilizarse un editor de texto para escribir ficheros Java y Xml y utilizar comandos en un terminal (se necesitan los paquetes JDK, Java Development Kit y Apache Ant) para crear y depurar aplicaciones. Además, pueden controlarse dispositivos Android que estén conectados (reiniciarlos, instalar aplicaciones en remoto).
Las Actualizaciones del SDK están coordinadas con el desarrollo general de Android. El SDK soporta también versiones antiguas de Android, por si los programadores necesitan instalar aplicaciones en dispositivos ya obsoletos o más antiguos. Las herramientas de desarrollo son componentes descargables, de modo que una vez instalada la última versión, pueden instalarse versiones anteriores y hacer pruebas de compatibilidad.

Una aplicación Android está compuesta por un conjunto de ficheros empaquetados en formato. apk y guardada en el directorio /data/app del sistema operativo Android (este directorio necesita permisos de súper usuario, root, por razones de seguridad). Un paquete APK incluye ficheros. dex (ejecutables Dalvik, un código intermedio compilado), recursos, etc.

2.4 Android Studio

Android Studio es el oficial IDE (Integrated Development Enviroment) ó en español también llamado Ambiente de Desarrollo Integrado, para desarrollo de aplicaciones de Android basado en Java.
Android Studio ofrece:

- Sistema de construcción a base de Gradle Flexible.
- Construir variantes y múltiples apk generación de archivos.
- Plantillas de código para ayudar a construir las características de aplicaciones comunes.
- El soporte integrado para Cloud Platform Google, por lo que es fácil de integrar *Google Cloud* Mensajería y App Engine.

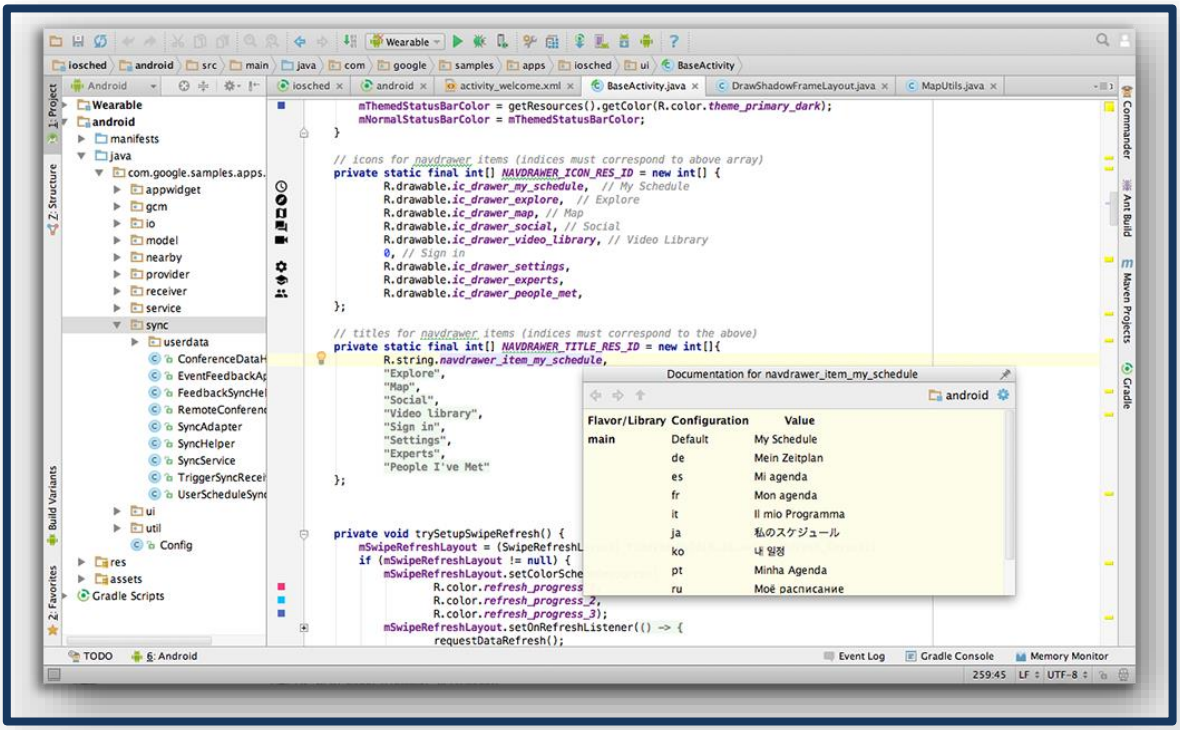


Figura 2.4-1 Android Studio

2.4.1.1 Vista *Android* proyecto.

Por defecto, Android Studio muestra los archivos del proyecto en la vista *Android* proyecto. Esta vista muestra una versión aplanada de la estructura de su proyecto que proporciona acceso rápido a los archivos de código fuente clave de los proyectos de Android y le ayuda a trabajar con el sistema de construcción a base de *Gradle*.

- Muestra los directorios de origen más importantes en el nivel superior de la jerarquía módulo.
- Grupos Los archivos de creación para todos los módulos en una carpeta común.
- Grupos todos los archivos de manifiesto para cada módulo en una carpeta común.
- Muestra los archivos de recursos de todos los conjuntos de código Gradle.
- Grupos de recursos archivos para diferentes lugares, orientaciones y tipos de pantalla en un solo grupo por tipo de recurso.

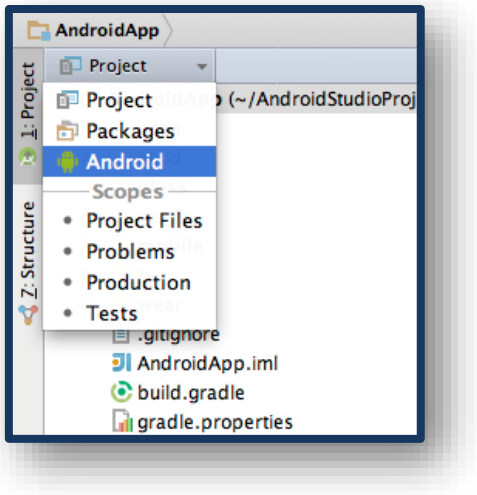


Figura 2.4-2 Muestra de la vista del proyecto Android

2.5 La Arquitectura de *Android*

El siguiente gráfico muestra la arquitectura de Android. Como se puede ver, está formada por cuatro capas. Una de las características más importantes es que todas las capas están basadas en *software* libre.

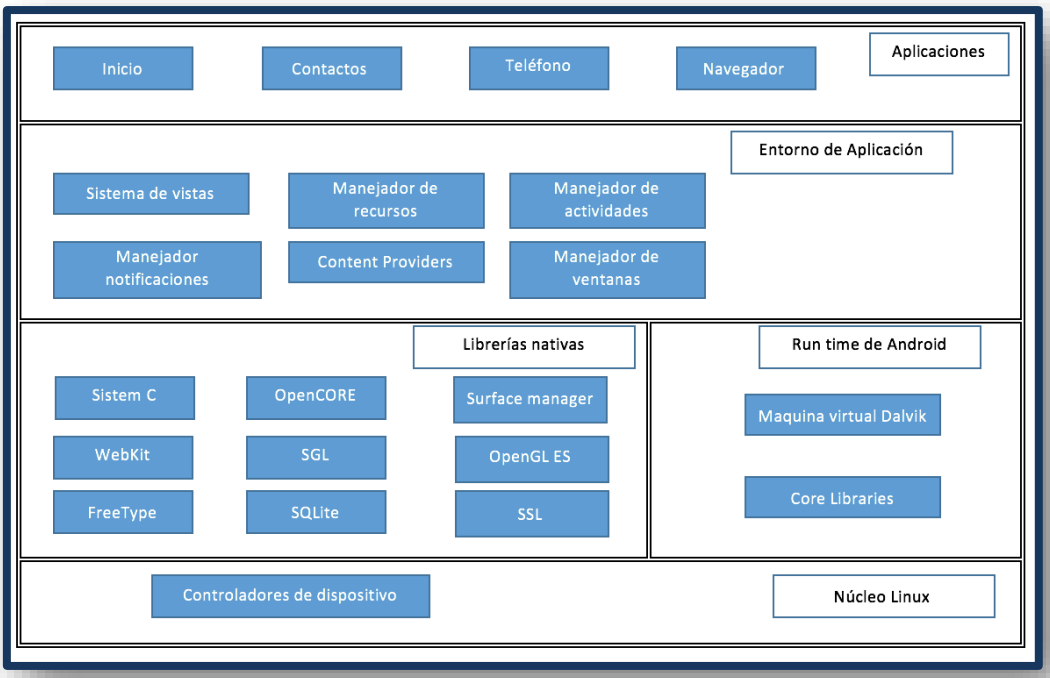


Figura 2.5-1 Arquitectura Android

2.5.1 El núcleo de Linux

El núcleo de Android está formado por el sistema operativo Linux, versión 2.6. Esta capa proporciona servicios como la seguridad, el manejo de la memoria, el multiproceso, la pila de protocolos y el soporte de para dispositivos.

Esta capa del modelo actúa como capa de abstracción entre el hardware y el resto de la pila. Por lo tanto, es la única que es dependiente del *hardware.*

2.5.2 *Runtime* de *Android*

Está basado en el concepto de máquina virtual utilizado en Java. Dado las limitaciones de los dispositivos donde ha de ejecutarse Android (poca memoria y procesador limitado) no fue posible utilizar una máquina virtual Java estándar. Google tomó la decisión de crear una nueva, la máquina virtual Dalvik, que respondiera mejor a estas limitaciones.

Algunas características de la máquina virtual Dalvik que facilitan esta optimización de recursos son: que ejecuta ficheros Dalvik ejecutables (.dex) (formato optimizado para ahorrar memoria). Además, está

basada en registros. Cada aplicación corre en su propio proceso Linux con su propia instancia de la máquina virtual Dalvik. Delega al kernel de Linux algunas funciones como *threading* y el manejo de la memoria a bajo nivel.

También se incluye en el *Runtime de Android* el "core libraries" con la mayoría de las librerías disponibles en el lenguaje Java.

2.5.3 Librerías nativas

1 Incluye un conjunto de librerías en C/C++ usadas en varios componentes de Android. Están compiladas en el código nativo del procesador. Muchas de las librerías utilizan proyectos de código abierto. Algunas de estas librerías son:

2 ***System C library:*** una derivación de la librería BSD de C estándar (libe), adaptada para dispositivos embebidos basados en Linux.

3 ***Media Framework:*** librería basada en *PacketVideo's* OpenCORE; soporta *codees* de reproducción y grabación de multitud de formatos de audio, vídeo e imágenes MPEG4, H.264, MP3, AAC, AMR, JPG y PNG.

4 ***Surface Manager:*** maneja el acceso al subsistema de representación gráfica en 2D y 3D.

5 ***WebKit:*** soporta un moderno navegador web utilizado en el navegador Android y en la vista *webview*. Se trata de la misma librería que utiliza Google Chrome y Safari de Apple.

6 **SGL:** motor de gráficos 2D.

7 **Librerías 3D:** implementación basada en OpenGL ES 1.0 API. Las librerías utilizan el acelerador hardware 3D si está disponible, o el software altamente optimizado de proyección 3D.

8 ***FreeType*:** fuentes en bitmap y renderizado vectorial.

9 ***SQLite*** potente y ligero motor de bases de datos relaciónales disponible para todas las aplicaciones.

10 **SSL:** proporciona servicios de encriptación *Secure Socket Layer.*

2.5.4 Entorno de aplicación

Proporciona una plataforma de desarrollo libre para aplicaciones con gran riqueza e innovaciones (sensores, localización, servicios, barra de notificaciones, etc.). Esta capa ha sido diseñada para simplificar la reutilización de componentes. Las aplicaciones pueden publicar sus capacidades y otras pueden hacer uso de ellas (sujetas a las restricciones de seguridad). Este mismo mecanismo permite a los usuarios remplazar componentes. Una de las mayores fortalezas del entorno de aplicación de Android es que se aprovecha el lenguaje de programación Java. El SDK de Android no acaba de ofrecer

todo lo disponible para su estándar del entorno de ejecución Java (JRE), pero es compatible con una fracción muy significativa de la misma. Los servicios más importantes que incluye son:

1. ***Views:*** extenso conjunto de vistas (parte visual de los componentes).
2. ***Resource Manager*****:** proporciona acceso a recursos que no son en el código.
3. ***Activity Manager:*** maneja el ciclo de vida de las aplicaciones y proporciona un sistema de navegación entre ellas.
4. ***Notification Manager:*** permite a las aplicaciones mostrar alertas personalizadas en la barra de estado.
5. ***Content Providers:*** mecanismo sencillo para acceder a datos de otras aplicaciones (como los contactos).

2.6 Elementos de un proyecto *Android*

Un proyecto Android está formado básicamente por un descriptor de la aplicación *(AndroidManifest.xml*), el código fuente y una serie de ficheros con recursos. Cada elemento se almacena en una carpeta específica:

- ***Src:*** Carpeta que contiene el código fuente de la aplicación. Como puedes observar los ficheros Java se almacenan en un espacio de nombres.
- ***Gen:*** Carpeta que contiene el código generado de forma automática por el SDK. Nunca hay que modificar de forma manual estos ficheros. Dentro se encuentra el siguiente fichero:
- ***R.java****:* Define una clase que asocia los recursos de la aplicación con identificadores. De esta forma los recursos podrán ser accedidos desde Java.
- ***Android x.x:*** Código JAR, el API de Android según la versión seleccionada.

Assets**:** Carpeta que puede contener una serie arbitraria de ficheros o carpetas que podrán ser utilizados por la aplicación (ficheros de datos, ficheros JAR externos, fuentes, etc.). A diferencia de la carpeta *res*, nunca se modifica el contenido de los ficheros de esta carpeta.

- ***Res****:* Carpeta que contiene los recursos usados por la aplicación.

Drawable: En esta carpeta se almacenan los ficheros de imágenes y descriptores de imágenes.

Layout*:* Contiene ficheros XML con vistas de la aplicación. Las vistas nos permitirán configurar las diferentes pantallas que compondrán la interfaz de usuario de la aplicación. Serán tratadas en el siguiente capítulo.

Menú: Ficheros XML con los menús de la aplicación.

Valúes: También se utiliza ficheros XML para indicar valores del tipo string, color o estilo. De esta manera se puede cambiar los valores sin necesidad de ir al código fuente. Por ejemplo, permitiría traducir una aplicación a otro idioma

Anim: Contiene ficheros XML con descripciones de animaciones.

Xml: Otros ficheros XML requeridos por la aplicación.

Visión general y entorno de desarrollo

Raw: Ficheros adicionales que no se encuentran en formato XML.

Doc: Documentación asociada al proyecto.

AndroidManifestxml*:* Este fichero describe la aplicación Android. En él se indican las *actividades, intenciones, servicios* y *proveedores de contenido* de la aplicación. También se declaran los permisos que requerirá la aplicación. Se indica la versión mínima de Android para poder ejecutarla.

Default.properties: Fichero generado automáticamente por el SDK. Nunca hay que modificarlo. Se utiliza para comprobar la versión del API y otras características cuando se instala la aplicación en el terminal.

NOTA: Como se acaba de ver, el uso de XML es ampliamente utilizado en las aplicaciones Android. También es conocido el excesivo uso de espacio que supone utilizar este formato para almacenar información. Esto parece contradecir la filosofía de Android que intenta optimizar al máximo los recursos. Para solucionar el problema, los ficheros XML son compilados a un formato más eficiente antes de ser transferidos al terminal móvil.

2.7 Componentes de una aplicación

Existen una serie de elementos clave que resultan imprescindibles para desarrollar aplicaciones en Android. En este apartado vamos a realizar una descripción inicial de algunos de los más importantes. A lo largo del libro se describirán con más detalle las clases Java que implementan cada uno de estos componentes.

2.7.1 Vista (*View*)

Las *vistas* son los elementos que componen la interfaz de usuario de una aplicación. Son, por ejemplo, un botón, una entrada de texto. Todas las vistas van a ser objetos descendientes de la clase *View,* y por

tanto, pueden ser definidos utilizando código Java. Sin embargo, lo habitual va a ser definir las vistas utilizando un fichero XML y dejar que el sistema cree los objetos por sí mismo a partir de este fichero. Esta forma de trabajar es muy similar a la definición de una página web utilizando código HTML.

2.7.2 Layout

Un *Layout* es un conjunto de vistas agrupadas de una determinada forma. Vamos a disponer de diferentes tipos de Layouts para organizar las vistas de forma lineal, en cuadrícula o indicando la posición absoluta de cada vista. Los Layouts también son objetos descendientes de la clase *View.* Igual que las vistas, los Layouts pueden ser definidos en código, aunque la forma habitual de definirlos es utilizando código XML.

2.7.3 Actividad (*Activity*)

Una aplicación en Android va a estar formada por un conjunto de elementos básicos de visualización, coloquialmente conocidos como pantallas de la aplicación. En Android cada uno de estos elementos, o pantallas, se conoce como *actividad.* Su función principal es la creación del interfaz de usuario. Una aplicación suele necesitar varias *actividades* para crear el interfaz de usuario. Las diferentes *actividades* creadas serán independientes entre sí, aunque todas trabajarán para un objetivo común. Toda actividad ha de pertenecer a una clase descendiente de *Activity.*

2.7.4 Servicio (*Service*)

Un servicio es un proceso que se ejecuta "detrás", sin la necesidad de una interacción con el usuario. Es algo parecido a un demonio en Unix o a un servicio en Windows. Android dispone de dos tipos de servicios: servicios locales, que pueden ser utilizados por aplicaciones del mismo terminal y servicios remotos, que pueden ser utilizados desde otros terminales.

2.7.5 Receptor de anuncios *(Broadcast receiver)*

Un *receptor de anuncios* recibe y reacciona ante anuncios de tipo *broadcast.* Existen muchos originados por el sistema, como por ejemplo *Batería baja, Llamada entrante.* Aunque, las aplicaciones también pueden lanzar un *anuncio broadcast.* No tienen interfaz de usuario, aunque pueden iniciar una actividad para atender a un anuncio.

2.8 Intención (*Intent*)

Una intención representa la voluntad de realizar alguna acción; como realizar una llamada de teléfono, visualizar una página web. Se utiliza cada vez que queramos:

- Lanzar una actividad.
- Lanzar un servicio.
- Lanzar un anuncio de tipo broadcast.
- Comunicarse con un servicio.

Los componentes lanzados pueden ser internos o externos a la aplicación. También se hace uso de las intenciones para el intercambio de información entre estos componentes.

En muchas ocasiones una intención no será inicializada por la aplicación, si no por el sistema, por ejemplo, cuando pedimos visualizar una página web. En otras ocasiones será necesario que la aplicación inicialice su propia intención. Para ello se creará un objeto de la clase Intent.

2.9 Receptor de anuncios (*Broadcast receiver*)

Un *receptor de anuncios* recibe y reacciona ante anuncios de tipo *broadcast.* Existen muchos originados por el sistema, como por ejemplo *Batería baja, Llamada entrante, ...* Aunque, las aplicaciones también pueden lanzar un *anuncio broadcast.* No tienen interfaz de usuario, aunque pueden iniciar una actividad para atender a un anuncio.

2.10 Comunicación de software por internet.

Arquitectura cliente servidor

Las aplicaciones de Internet suelen seguir la arquitectura cliente/servidor. Esta arquitectura se caracteriza por descomponer el trabajo en dos partes (es decir dos programas: el servidor, que centraliza el servicio, y el cliente, que controla la interacción con el usuario. El servidor ha de ofrecer sus servicios a través de una dirección conocida. Algunos ejemplos de aplicaciones basadas en la arquitectura cliente/servidor son WWW o el correo electrónico. Se suelen seguir las siguientes pautas de comportamiento en esta arquitectura.

Cliente	Servidor
1.- Se conecta el servidor. 2.- Solicita alguna información al servidor. 3.- Recibe la respuesta. 4.- Ir al punto 2. 5.- Cierra la conexión.	1.- A la espera de que algún cliente se conecte. 2.- Recibe la solicitud. 3.-Envía respuesta. 4.- Ir al punto 2. 5.- Cierra la conexión. 6.- Ir al punto 1.

2.10.1 Socket

¿Qué es un *socket*?

Cada una de las diferentes aplicaciones en Internet (web, correo electrónico, etc.) ha de poder intercambiar información entre programas situados en diferentes ordenadores o dispositivos. Con este propósito, se va a hacer uso de pila de transporte de protocolos TCP/IP, cuyo objetivo final es permitir intercambio de información a través de la red en forma fiable y transparente.

Un socket es el punto final de una comunicación bidireccional entre dos programas que intercambian información a través de internet (*socket* se traduce literalmente como enchufe).

Dado que un dispositivo se puede estar ejecutando de forma simultanea diferentes aplicaciones que utilizan internet para comunicarse, resulta imprescindible identificar cada socket con una dirección diferente. Un socket se va identificar por la dirección IP de cada dispositivo donde está, más un numero de puerto (de 16 bits). Una conexión está determinada por un par de sockets, que son los extremos de la conexión.

Existen dos tipos de socket, socket stream y socket datagrama los cuales se describen a continuación:

- Socket Stream (TCP)

Los sockets stream ofrecen un servicio orientado a la conexión, donde los datos se transfieren como un flujo continuo, sin encuadrarlos en registros o bloques. Este tipo de socket se basa en el protocolo TCP, que es un protocolo orientado a conexión. Esto implica que antes de transmitir la información hay que establecer una conexión entre los dos sockets. Mientras uno de los sockets atiende peticiones de conexión (servidor), el otro solicita la conexión (cliente). Una vez que los dos sockets están conectados, ya se pueden transmitir datos en ambas direcciones. El protocolo incorpora de forma transparente al programador la corrección de errores. Es decir, si detecta que parte de la información no llego a su destino correctamente, esta volverá a ser trasmitida. Además, no limita el tamaño máximo de información a transmitir.

- Sockets Datagram (UDP)

Los sockets datagram se basan en el protocolo UDP y ofrecen un servicio de transporte sin conexión. Es decir, puede mandar información a un destino sin necesidad de realizar una conexión previa. El protocolo UDP es más eficiente que TCP, pero tiene el inconveniente que no se garantiza la fiabilidad. Además, los datos se envían y reciben en datagramas (paquetes de información) de tamaño limitado. La entrega de un datagrama no está garantizada: estos pueden ser duplicados, perdidos o llegar en un orden diferente al que se envió.

La gran ventaja de este tipo de sockets es que apenas introduce sobrecarga sobre la información transmitida. Además, los retrasos introducidos son mínimos, lo que los hace especialmente interesantes para aplicaciones en tiempo real, como la transmisión de audio y vídeo sobre Internet. Sin embargo, presenta muchos inconvenientes al programador: cuando transmite un datagrama no se tiene la certeza de que este llegue a su destino, por lo que, si fuera necesario, tendríamos que implementar nuestro propio mecanismo de control de errores. Otro inconveniente es el hecho de que existe un tamaño máximo de datagrama, unos 1.500 bytes dependiendo de la implementación. Si la información a enviar es mayor, tendríamos que fraccionarla y enviar varios datagramas independientes. En el destino tendríamos que concatenarlos en el orden correcto.

- MulticastSocket

La operación de multicast consiste en enviar un único mensaje desde un proceso a cada uno de los miembros de un grupo de procesos, de modo que la pertenencia a un grupo sea transparente al emisor, es decir, el emisor no conoce el número de miembros del grupo ni sus direcciones IP.

Un grupo multicast está especificado por una dirección IP clase D y un puerto. Las direcciones IP clase D están en el rango 224.0.0.0 a 239.255.255.255, dentro de este rango existen direcciones reservadas, en concreto, la 224.0.0.1 y la 224.0.0.255. El resto de direcciones del rango pueden ser utilizadas por grupos temporales, los cuales deben ser creados antes de su uso y dejar de existir cuando todos los miembros lo hayan dejado.

Java proporciona una interfaz de datagramas para multicast IP a través de la clase MulticastSocket, que es una subclase de DatagramSocket, con la capacidad adicional de ser capaz de pertenecer a grupos multicast.

La clase MulticastSocket proporciona dos constructores alternativos:
MulticastSocket (): que crea el socket en cualquiera de los puertos locales libres.
MulticastSocket (int port): que crea el socket en el puerto local indicado.

Un proceso puede pertenecer a un grupo multicast invocando el método joinGroup (InetAddress mcastaddr) de su socket multicast. Así, el socket pertenecerá a un grupo de multidifusión en un puerto dado y recibirá los datagramas enviados por los procesos en otros computadores a ese grupo en ese puerto. Un proceso puede dejar un grupo dado invocando el método leaveGroup (InetAddress mcastaddr) de su socket multicast.

Para enviar datos a un grupo multicast se utiliza el método send (DatagramPacket p, byte ttl), este método es muy similar al de la clase DatagramSocket, la diferencia es que este datagrama será enviado a todos los miembros del grupo multicast. El parámetro TTL, Time-To-Live, se pone siempre a 1, valor por defecto, para que sólo se difunda en la red local.

Para recibir datos de un grupo multicast se utiliza el método receive (DatagramPacket p) de la clase DatagramSocket superclase de MulticastSocket.

Es necesario pertenecer a un grupo para recibir mensajes multicast enviados a ese grupo, pero no es necesario para enviar mensajes.En el programa de Android se implementará las acciones que debe de hacer un participante en un grupo multicast, como no es necesario pertenecer a un grupo multicast para enviar datos la aplicación del trolebús (servidor) enviará un mensaje al grupo sin que esta pertenezca a él, en el caso de la aplicación de usuario el socket consultará el puerto y la IP preestablecida para obtener el dato de las coordenadas.

2.11 Las aplicaciones y el internet

Sin duda alguna el mundo no sería el mismo sin el internet derivado de ello existen las aplicaciones. Las aplicaciones han tomado parte importante en la vida cotidiana del ser humano esto va desde la aplicación que despierta hasta la aplicación que te avisa del tráfico de la ciudad.

Esto trajo como consecuencia que la mayoría de los programadores y gente con simples conocimientos de programación se enfocaran e interesaran en este campo. Otro resultado que trajeron las aplicaciones fue sin duda a nivel competencia, la razón es que debido al auge de los sistemas operativos las aplicaciones fueron en conjunto evolucionando y creando tendencia y por ello daban valor agregado a cierto sistema operativo, esto por la exclusividad que le deba agregar una aplicación que otros sistemas no incluían, lo que en cuestiones de mercadotecnia se traducía en muy apreciables números.

Las aplicaciones a partir de ese punto significaron una fuente de ingresos muy remunerada, bastando una idea innovadora y conocimientos de programación.
Como propósito esta aplicación pretende llenar una necesidad que derivado de lo antes ya mencionado (Trolebuses) puede ser de valor útil en un futuro a transportes de mayor concurrencia.

2.12 El mundo de las aplicaciones con respecto al transporte.

Seria de poca credibilidad decir que no hay aplicaciones similares a la nuestra por ello, a continuación, se mostrarán las aplicaciones que hoy en día están establecidas y que son competidores potenciales a la aplicación.

- **La aplicación de transporte Moovit.**

La aplicación se basa en la información que mandan sus propios usuarios sobre el tiempo que tardan en hacer un recorrido. Todos estos datos son recogidos por el servicio y enviados a todo usuario al que le podrían ser útiles.

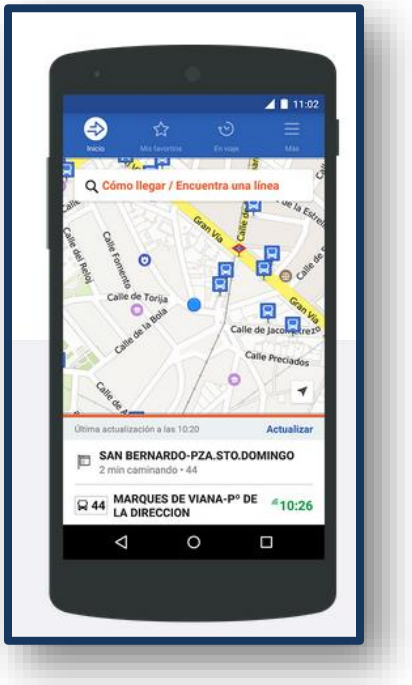


Figura 2.12-1 Aplicación Moovit

Un usuario de Moovit es capaz de comprobar cuanto tiempo está tardando otro usuario en realizar un trayecto, de manera que pude decidir si tomar o no esa ruta. La aplicación también ofrece diferentes posibilidades, mostrando al usuario lo que se tarda en hacer el mismo recorrido en diferentes medios de transporte.

Moovit es especialmente útil para aquellos que se mueven en transporte público está disponible en los principales sistemas operativos: Android, iOS y Windows Mobile.

- **Waze**

Obtiene la mejor ruta, con la ayuda en tiempo real de otros conductores. Waze es la aplicación de tráfico y navegación basada en la comunidad más grande del mundo (*véase Figura 2.11-2*).

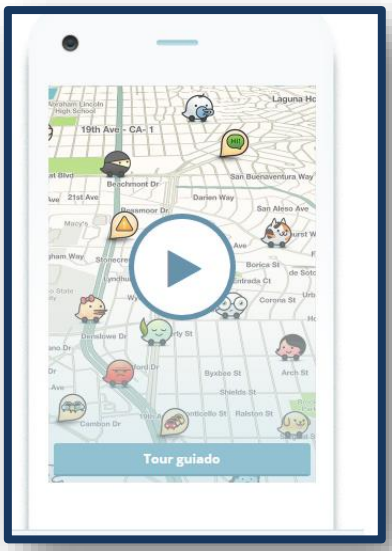


Figura 2.12-2 Aplicación Waze

2.13 Sumario

El presente capítulo se describe de manera general las diferentes herramientas que se utilizarán para el desarrollo de la aplicación prototipo, se observan las diferentes partes de las cuales está conformada una aplicación para el sistema operativo *Android*.

De la misma forma se toca un tema donde se trata de explicar cómo es la integración del internet en las aplicaciones móviles, además de dar puntos de vista de cómo es que este tipo de *software* móvil han cambiado la forma de vida de los usuarios.

Por último, se trató el tema de las aplicaciones que en la actualidad lideran la unión de *software*, transporte e internet, esto con la finalidad de dar al lector una idea de hacia dónde va dirigida la aplicación que se desarrollará en este proyecto, en el siguiente capítulo se redactará como se hizo el desarrollo de la aplicación prototipo, la cual intenta lograr los objetivos planteados en este escrito.

Capítulo 3. Diseño y construcción de la aplicación.

3.1 Ubicación del prototipo a desarrollar.

El desarrollo de la aplicación para el trolebús tiene como lugar la Unidad Profesional Adolfo López Mateos. La unidad consta de un circuito para trolebús que a continuación se ilustrará:

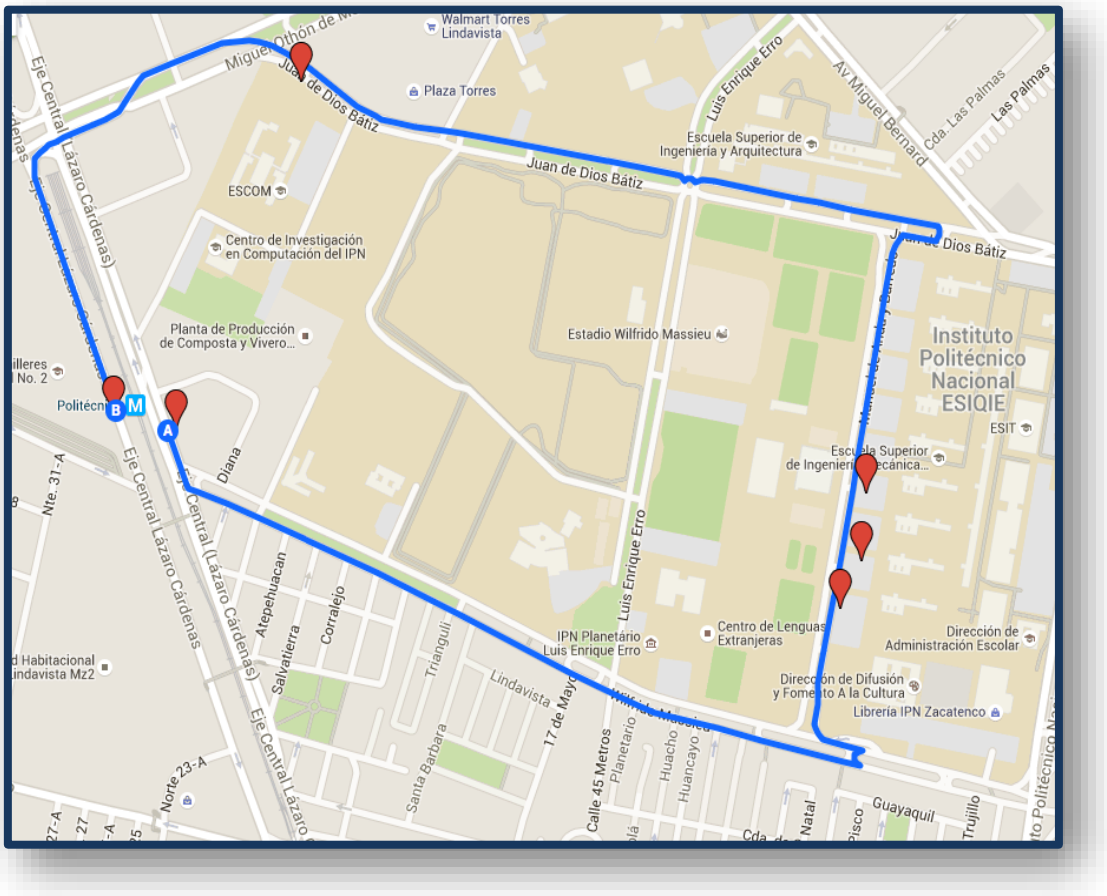


Figura 3.1-1 Ruta del Transporte (Trolebús)

3.2 Localización.

Antes de describir cual es el código necesario para obtener las coordenadas del trolebús primeramente se tendrá un análisis de las facilidades que nos da Android para obtener las coordenadas de los dispositivos que tengan instalado este sistema operativo.

La plataforma *Android* dispone de un interesante sistema de posicionamiento que combina varias tecnologías:

Sistema de localización global basado en GPS. Este sistema solo funciona si se dispone de visibilidad directa de los satélites. Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios.

Estos servicios se encuentran totalmente integrados en el sistema y son usados por gran variedad de aplicaciones. Por ejemplo, una aplicación de *Android* puede adaptar la configuración del teléfono según donde se encuentre. Podría por ejemplo poner el modo de llamada en vibración en el trabajo.

El sistema de posicionamiento global, GPS, fue diseñado inicialmente con fines militares, pero hoy en día es ampliamente utilizado para uso civil. Gracias al desfase temporal de las señales recibidas por varios de los 31 satélites desplegados, este sistema es capaz de posicionar a cualquier persona en cualquier parte del planeta con una precisión de 15 metros.

El GPS presenta un inconveniente; solo funciona cuando hay visión directa de los satélites. Para solventar este problema, *Android* combina esta información con la recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.

Por motivos de seguridad Android impide a las aplicaciones acceder a la ubicación del usuario, por lo tanto, si estas desean hacer uso de dicho servicio han de solicitar permisos especiales y estos hay que indicarlos en el fichero AndrodiManifest.xml en concreto las aplicaciones necesitan los permisos de localización precisa e imprecisa que se muestran a continuación.

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

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

3.3 Aplicación del trolebús (servidor)

El funcionamiento de la aplicación para del trolebús consiste básicamente en la obtención de las coordenadas del trolebús y su transmisión de los datos hacia un cliente o usuario, mediante la

implementación de diferentes líneas de código escritas en lenguaje java y xml, las cuales se explican más adelante. Los pasos para ejecutar esta aplicación, son mejor apreciados en su diagrama de flujo.

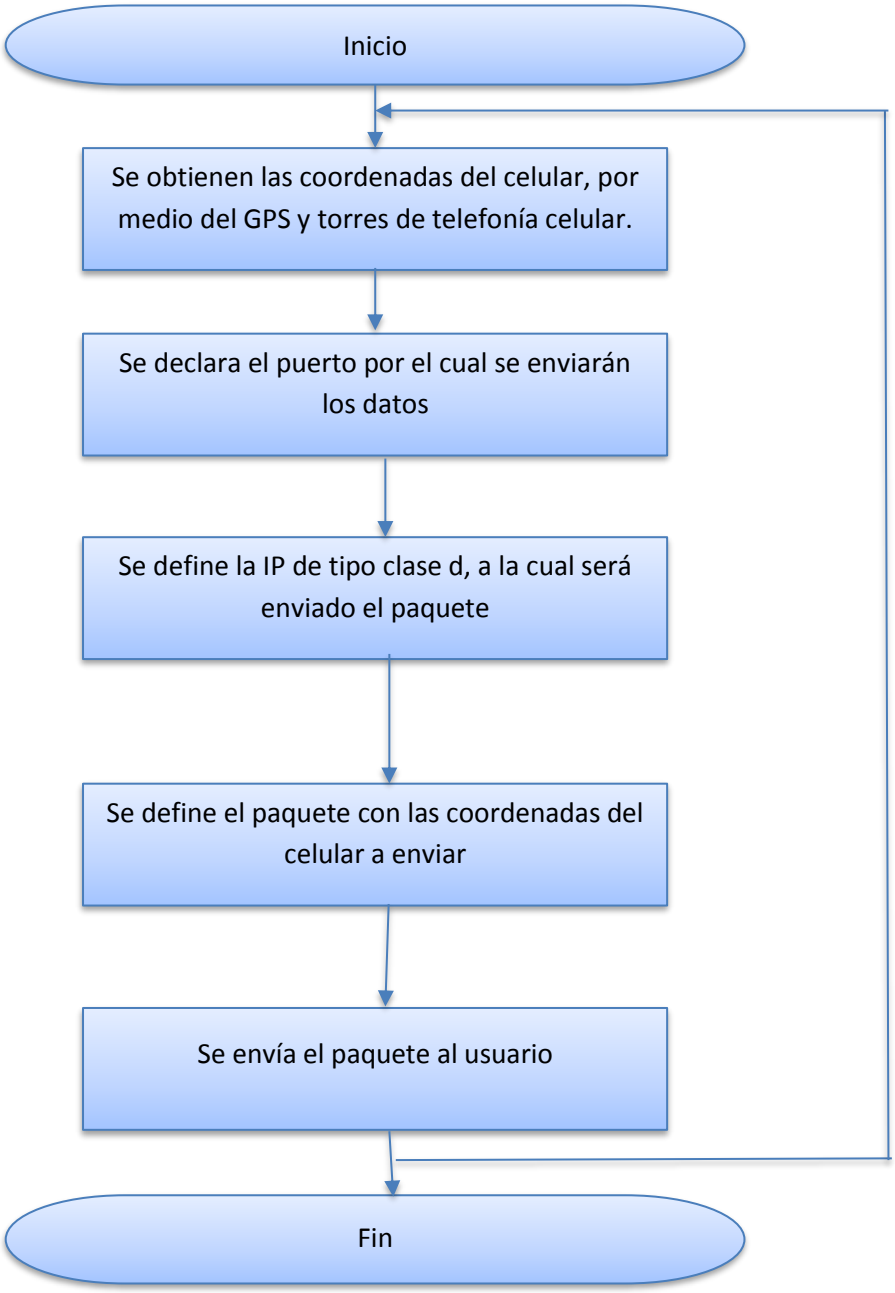


Figura 3.3-1 Diagrama de flujo de la aplicación del servidor

3.3.1 Obtención de las coordenadas

La plataforma Android dispone de un interesante sistema de posicionamiento, que se apoya en dos métodos:

- Sistema de localización global basado en GPS. Deberá de estar a cielo abierto
- Sistema de localización basado en la información recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi. Funciona en el interior de los edificios.

Para solventar el problema de la visibilidad de los satélites, Android combina esta información con la recibida de las torres de telefonía celular y de puntos de acceso Wi-Fi.

Como se mencionó anteriormente en la parte de la localización, se necesita declarar los permisos en el fichero AndrodiManifest.xml, que son principalmente los permisos de localización precisa e imprecisa, entre otros.

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

<uses-permission android:name="android.permission.LOCATION_HARDWARE"></uses-permission>
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-permission>
```

Figura 3.3-2 Solicitudes de permisos

Todas las pantallas que se observan en *Android* se deben diseñar en el archivo Layout.xml ubicada en la siguiente ruta app\src\main\res\layout. La plataforma *AndroidStudio* da la facilidad de pre visualizar la interfaz que se utilizara en los dispositivos como se muestra en la figura 3.2-1.

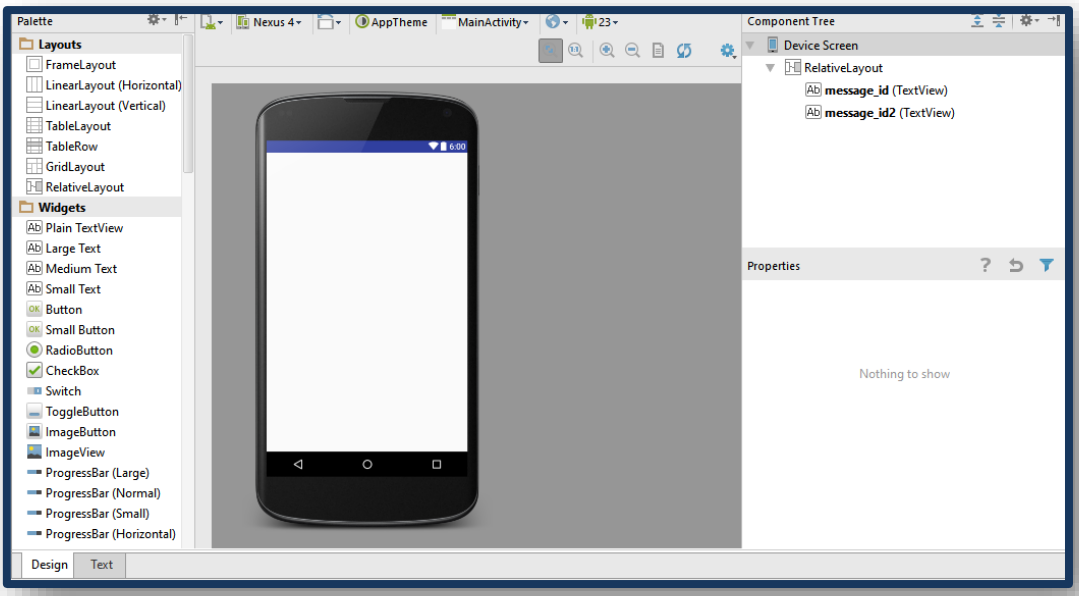


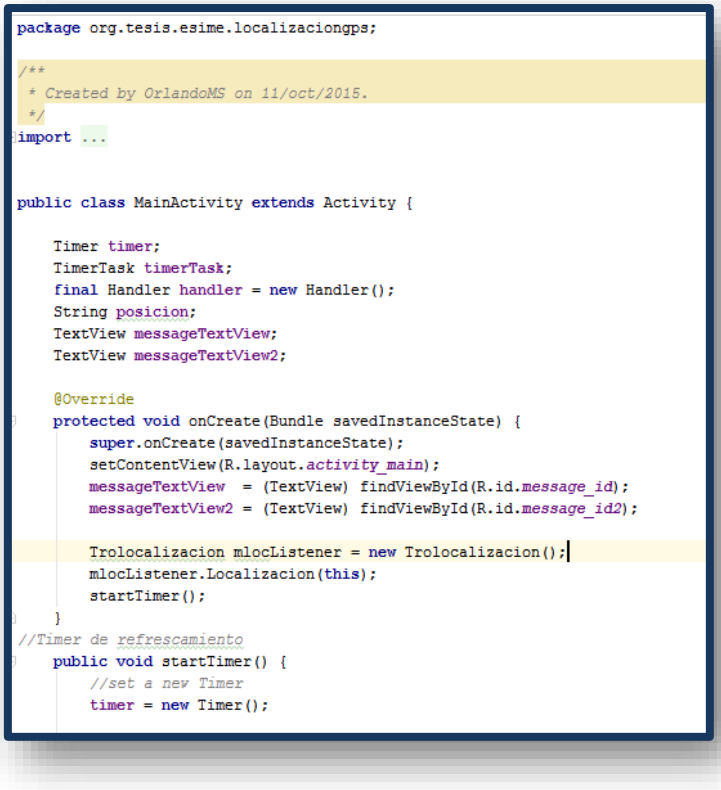
Figura 3.3-3Pre visualización de Android Studio

Por otra parte, también permite definir la interfaz en forma de código obteniendo mayor exactitud y detalle. La interfaz cuenta con dos campos de texto (TextView), en los cuales se imprimen las coordenadas (latitud y longitud) del dispositivo.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/message_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        />

    <TextView
        android:id="@+id/message_id2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/message_id"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="30dp"
        android:text="" />

</RelativeLayout>
```

Figura 3.3-4 Text View de la interfaz

Primeramente, se inicia el código declarando las variables de forma general utilizadas en todo el programa, se crea un método OnCreate el cual inicializa la actividad que se utiliza para la aplicación, dicho método manda a llamar por nombre id a la actividad creada en el layout, una vez que se haya realizado esto, se genera un variable mlocListener para mandar a llamar al constructor de la clase Trolocalización. En la siguiente línea el objeto anteriormente creado invoca el método Localización y rellena la variable con sí mismo, en el mismo método OnCreate se inicia el método starTimer como se muestra en el siguiente código:



```
package org.tesis.esime.localizaciongps;

/**
 * Created by OrlandoMS on 11/oct/2015.
 */
import ...


public class MainActivity extends Activity {

    Timer timer;
    TimerTask timerTask;
    final Handler handler = new Handler();
    String posicion;
    TextView messageTextView;
    TextView messageTextView2;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        messageTextView  = (TextView) findViewById(R.id.message_id);
        messageTextView2 = (TextView) findViewById(R.id.message_id2);

        Trolocalizacion mlocListener = new Trolocalizacion();
        mlocListener.Localizacion(this);
        startTimer();
    }
//Timer de refrescamiento
    public void startTimer() {
        //set a new Timer
        timer = new Timer();
```

Figura 3.3-5 Creación de métodos

A continuación se muestran las acciones que ejecuta el método startTimer, en primera instancia se toma el objeto timer (el cual se había generado anteriormente), este manda a llamar el constructor del método Timer, en la siguiente línea se manda invocar el método initializeTimerTask() y finalmente enseguida se declara cuando se va iniciar el timer y cada cuando se va a iniciar nueva mente respectivamente.

En el método initializaeTimerTask a grandes rasgos es el encargado de mandar a llamar el constructor Socket_UPD y rellenara con la variable *posición*, que es el dato de las coordenadas del trolebús para posteriormente ser enviadas al cliente.

```
public void startTimer() {

    timer = new Timer();
    initializeTimerTask();
    timer.schedule(timerTask, 1000, 1000); //
}

public void stoptimertask(View v) {
    //stop the timer, if it's not already null
    if (timer != null) {
        timer.cancel();
        timer = null;
    }
}

public void initializeTimerTask() {

    timerTask = new TimerTask() {
        @Override
        public void run() {
            try {
                handler.post(() -> {
                        new Socket_UDP(posicion).execute("");
                });
            }
            catch (Exception e){
                Toast toast = Toast.makeText(getApplicationContext(),
                        e.toString(), Toast.LENGTH_SHORT);
                toast.show();
            }
        }
    };
}
```

Figura 3.3-6 Métodos startTimer e initializeTimerTask

La clase Trolocalizacion es la encargada de obtener las coordenadas del trolebús, esta clase implementa el LocationListener, la cual se encarga de estar siempre atenta a cualquier cambio de localidad recibido en el GPS del dispositivo, esto se podría interpretar como un radar que detecta toda señal de cambio de ubicación que el GPS emite.

En el código se observa el método Localización (MainActivity locate), esta variable es la que se rellena con el objeto posición, el cual se visualizó en el método OnCreate. El método Localización a grandes rasgos es el encargado de definir al proveedor de las coordenadas al dispositivo del trolebús, en esta aplicación se utilizan los proveedores por GPS e Internet para más detalles del código observar Apéndice 1.

```
public class Trolocalizacion implements LocationListener {
    MainActivity coor;
    LocationManager mlocManager,manejador;

    public void Localizacion(MainActivity locate) {
        mlocManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        manejador = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        if(mlocManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
            mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                    0,0, this);
        }


        if(manejador.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
            manejador.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                    0,0, this);
        }

        this.coor= locate;
        messageTextView.setText("LocationListener agregado\nBuscado coordenadas...");
        messageTextView2.setText("");

    }
```

Figura 3.3-7 Método Trolocalizacion e Localizacion

La siguiente serie de métodos sirve para configurar diferentes acciones del LocationListener, pero el más importante es el método onLocationChanged(*Location sat*), aquí es donde se encuentran las coordenadas del dispositivo (trolebús), en el se observa que el objetivo sat obtiene los atributos de los métodos getLatitude() y getLongitude(), una vez obtenidos estos datos se mandan a dos cadenas, una es *Texto* que posteriormente se mandara a imprimir en pantalla y la otra es *Éxito* esta variable se iguala con el objeto posición para posteriormente ser enviado por medio del Socket.

Los otros métodos mandan a imprimir las cadenas respectivas dependiendo si los proveedores están habilitados o deshabilitados.

```
    @Override
    public void onLocationChanged(Location sat) {
        sat.getLatitude();
        sat.getLongitude();
        String Texto = "Mi ubicación actual es: " + "\n Lat = "
                + sat.getLatitude() + "\n Long = " + sat.getLongitude();
        String Exito=sat.getLatitude()+","+sat.getLongitude();
        messageTextView.setText(Texto);
        ////////////////////////////////////////////
        this.coor.setLocation(sat);
        posicion = Exito;
    }

    @Override
    public void onProviderDisabled(String provider) {
        messageTextView.setText("GPS Desactivado");
    }

    @Override
    public void onProviderEnabled(String provider) {
        messageTextView.setText("GPS Activado");
    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }
}
```

Figura 3.3-8 Configuración de LocationListener

3.3.2 Socket del servidor para la trasferencia de coordenadas.

La clase Socket_UDP es una de las más importantes de este programa, ya que es la encargada de enviar las coordenadas del trolebús a la aplicación de usuario. Esta inicia creando un objeto cadena llamado coordenada, posteriormente se crea el constructor de esta clase que solo llena los atributos de coordenada los cuales son tomados del objeto posición gracias a timer que se explicó anteriormente, estos atributos se los aplica a el objeto coordenada que se generó en la misma clase, para diferenciar los objetos uno del otro se hace mención de la palabra reservada “this” la cual indica que es el objeto que se generó en la misma clase Socket_UDP.

Se genera un subproceso que se ejecuta en el fondo para que el socket funcione de manera que no intervenga en el programa principal, se identifica el puerto, el cual es una variable de tipo entero, se genera una sentencia try, esta es la encargada de intentar ejecutar el socket, este inicia generando una variable de tipo cadena que tomara los datos de coordenada, posteriormente se declara un objeto de

tipo MulticastSocket y este llama a el constructor de esta clase dando el puerto al cual se va a vincular, enseguida se manda a convertir la cadena str en bytes para poder ser enviada.

En la línea siguiente es la destinada de fijar la ip de tipo multicast a la que se enviaran los datos, una vez fijados todos estos datos se manda a preparar los paquetes de datos con la variable send_packet de tipo DatagramPacket, acto seguido el programa envía todos estos paquetes por medio de la línea socketCliente.send(send_packet).

```
public class Socket_UDP extends AsyncTask<String, Void, String> {
    String coordenada;
    public Socket_UDP(String coordenada){

        this.coordenada=coordenada;
    }

    @Override
    protected String doInBackground(String... params) {

        int puerto = 9010;

        try {
            String str =coordenada ;

            MulticastSocket socketCliente = new MulticastSocket(puerto);
            byte[] enviar = str.getBytes();
            InetAddress Host = InetAddress.getByName("228.5.6.7");

            DatagramPacket send_packet = new DatagramPacket(enviar, str.length(), Host, puerto);
            socketCliente.send(send_packet);

        } catch (SocketException e) {
            e.printStackTrace();
        } catch (UnknownHostException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }catch (Exception e){

        }
        //////////////////////
        return null;
```

Figura 3.3-9 Socket UDP

3.4 Aplicación de usuario (cliente).

Para la parte de la aplicación del usuario, se lleva a cabo la implementación de la API de Google Maps, ya que permite desplegar el mapa con el cual debe de interactuar el usuario y donde, además, se pueden visualizar por medio de marcadores su posición y la del servidor (trolebús) el cual está enviando repetitivamente un paquete de datos con sus coordenadas. Posteriormente, se hace la inclusión de un buscador que permite ingresar direcciones y, por medio de un botón, esta aplicación podrá llevar al usuario al cualquier lugar de interés.

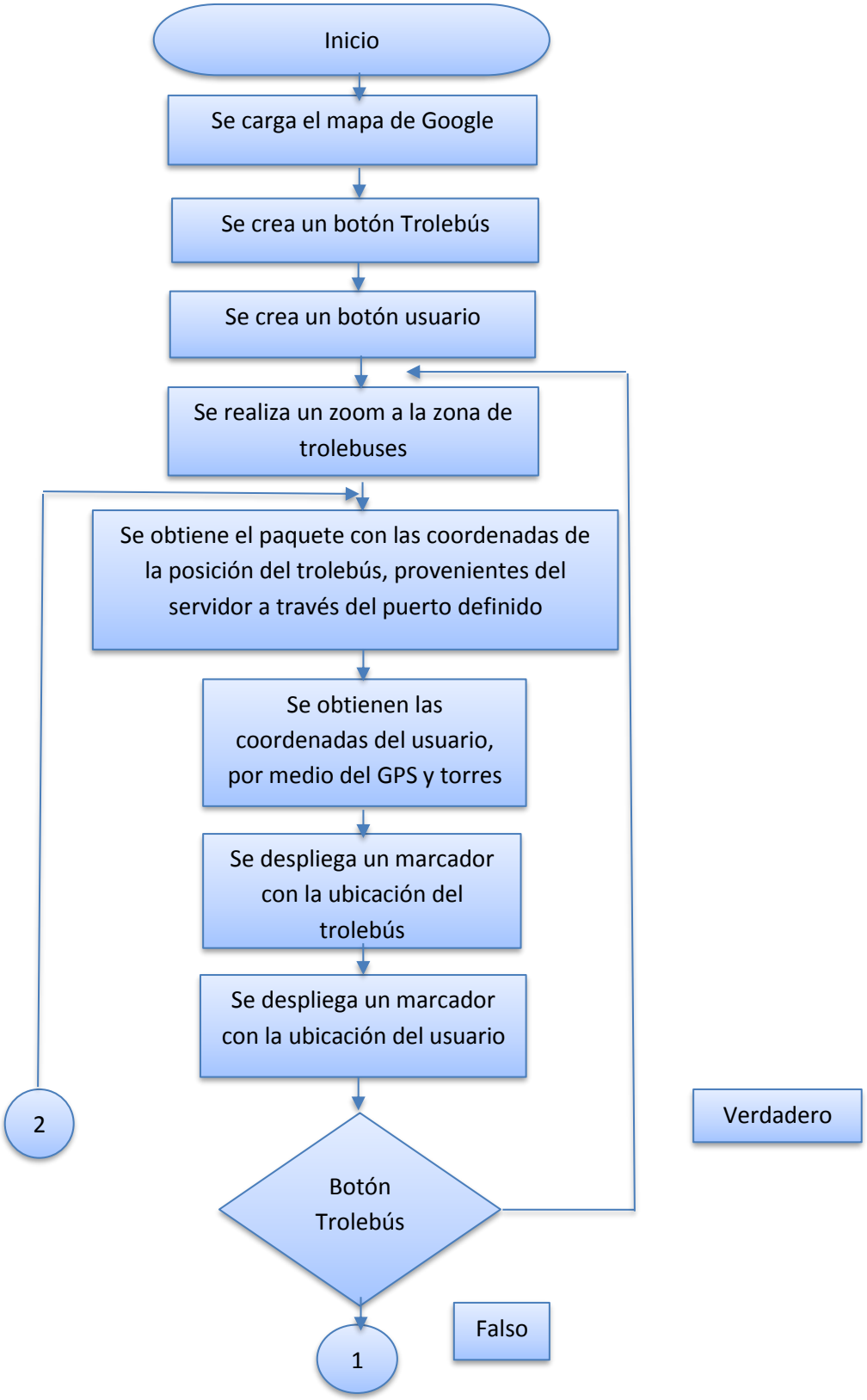
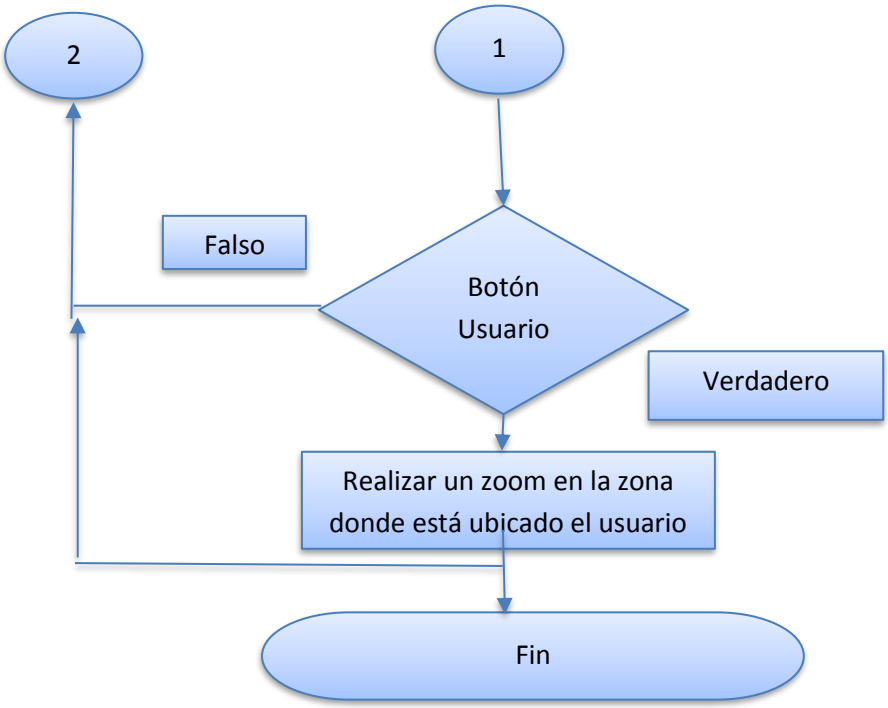


Figura 3.4-1 Diagrama de flujo aplicación usuario



3.4.1 Obtención de la llave para el uso de google maps

Para poder hacer uso de la API de Google Maps se tiene que emplear una llave que es proporcionada por Google, la cual es exclusivamente para la aplicación desde donde se hace la solicitud.

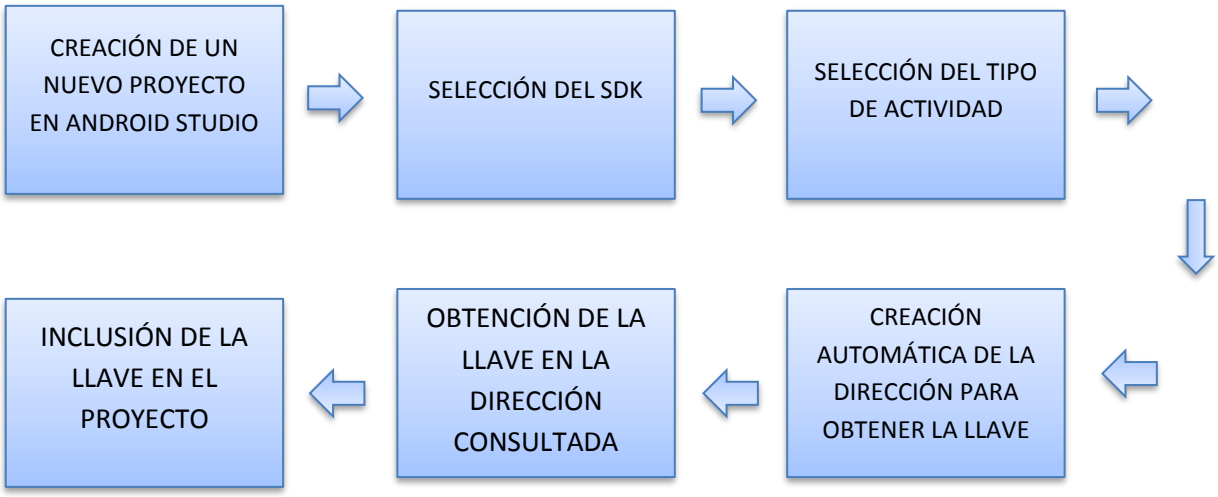


Figura 3.4-2 Diagrama conceptual para la obtención de la llave

La obtención de la llave que habilite a la API de Google Maps se detalla con mayor profundidad en los siguientes pasos:

1.- Se crea un nuevo archivo en la plataforma Android Studio, en este caso, con el nombre de Aplicación-Trolebús.

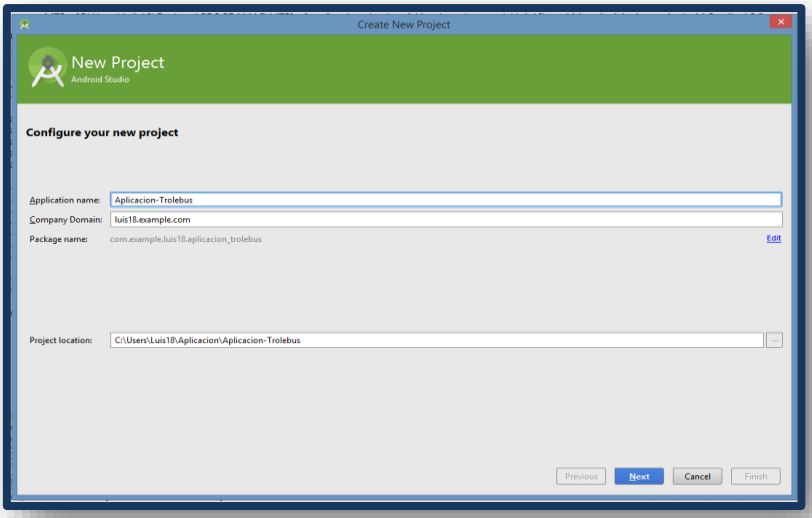


Figura 3.4-3 Creación de un proyecto Android

2.- Se selecciona el SDK mínimo que requerirá la aplicación para operar correctamente, por lo tanto, se selecciona la versión Android 4.0, que es compatible con la mayoría de los dispositivos móviles Android en circulación.

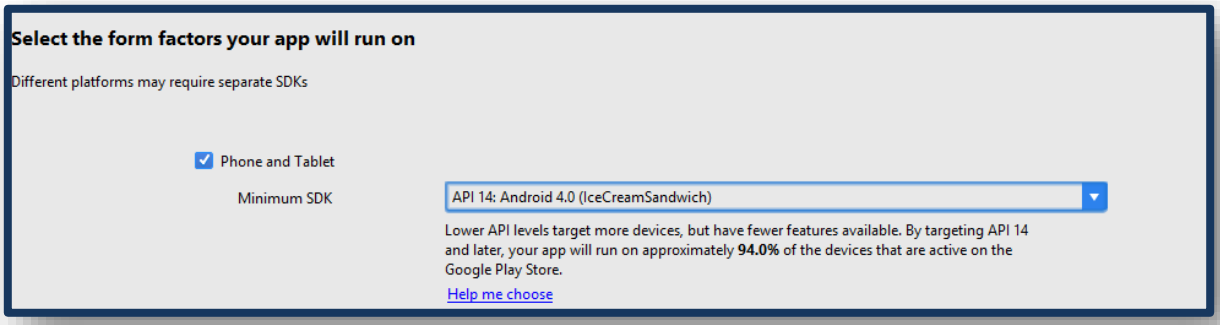


Figura 3.4-4 Selección de API

3.- Se selecciona el tipo de actividad con la que se desea que trabaje desde un principio la aplicación, por lo cual se escoge Google Maps Activity que proveerá las condiciones necesarias para trabajar con la API de Google Maps.

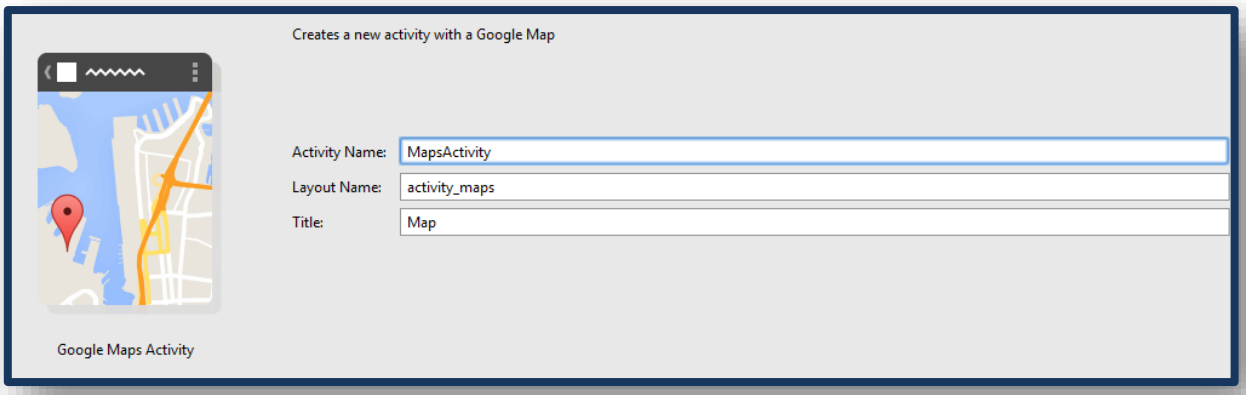


Figura 3.4-5 Creación de la Actividad

4.- Posteriormente al haberse creado el archivo se genera automáticamente un link con la dirección para obtener la llave de la aplicación, además se indica en que parte se debe de colocar la llave.

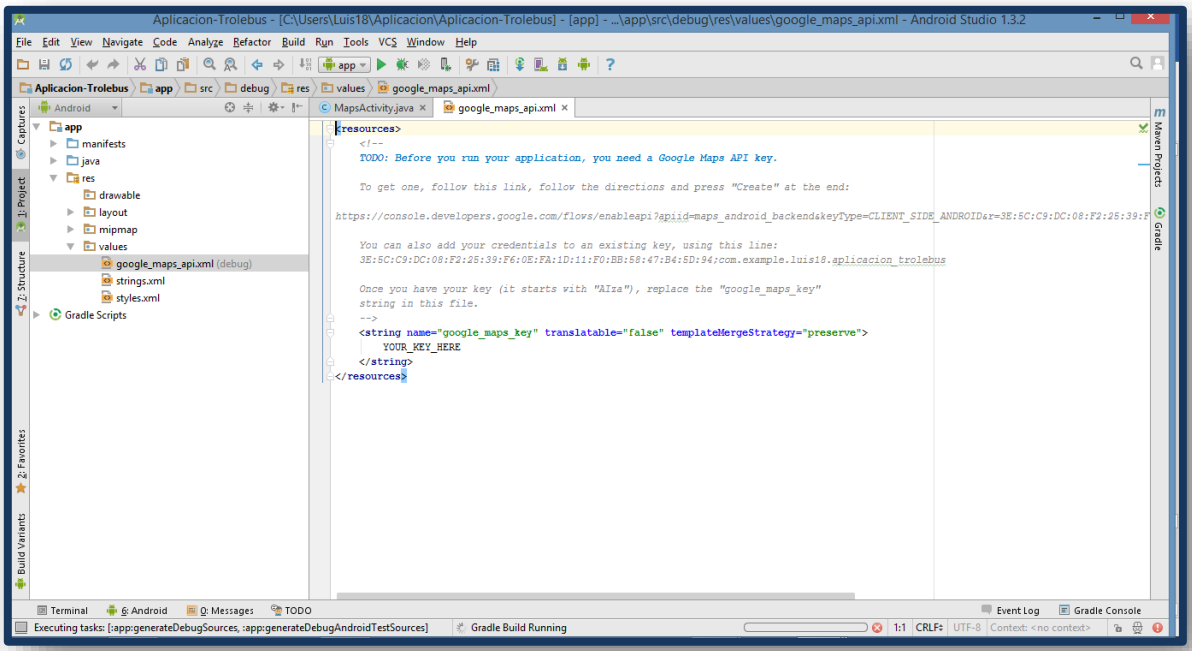


Figura 3.4-6 Ruta para la obtención de la llave de Google Maps

5.- Al ingresar el link a un buscador se direcciona a la página de la figura 3.3:5. En donde se deben de aceptar los términos y condicione para el uso del servicio.

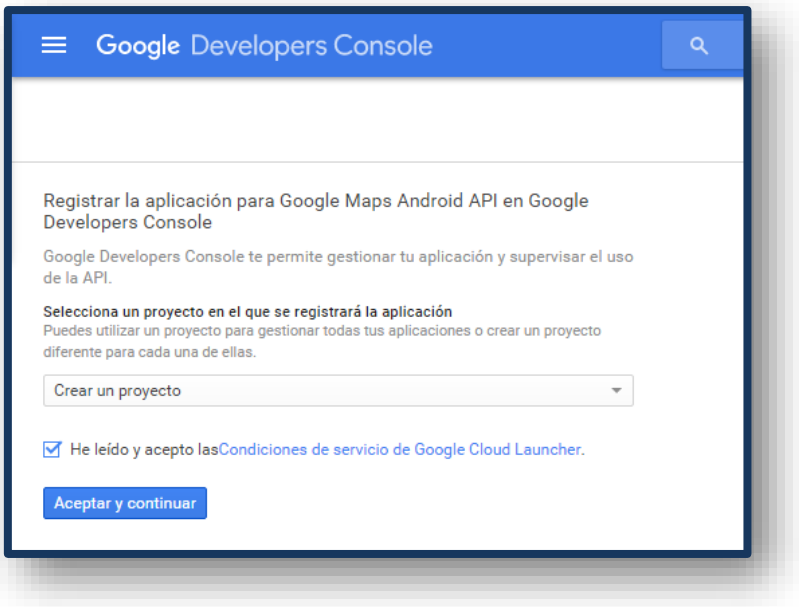


Figura 3.4-7 Registro de la aplicación

6.- Se indica la habilitación exitosa de la API de Google Maps, para que posteriormente se pueda acceder a la parte de las credenciales.

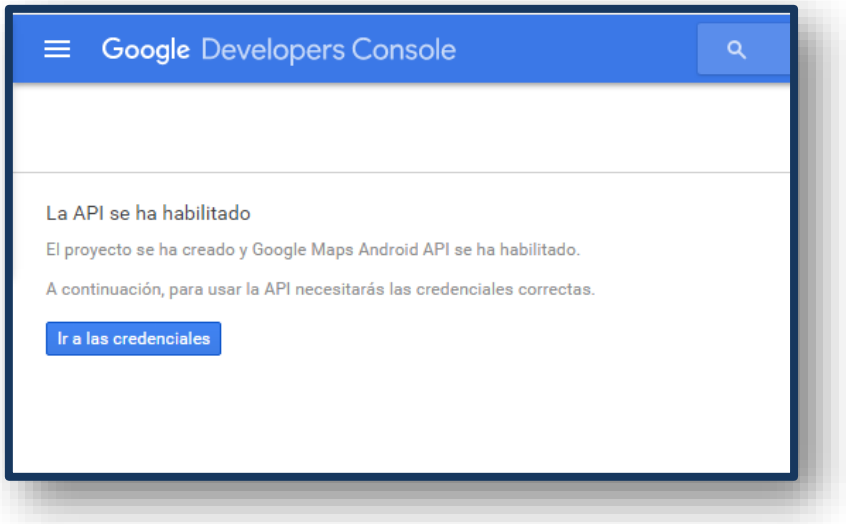


Figura 3.4-8 Habilitación de la API

7.- En este punto se habilita la opción para crear la llave a partir del nombre del paquete de la aplicación y de la Huella digital de certificado SHA-1.

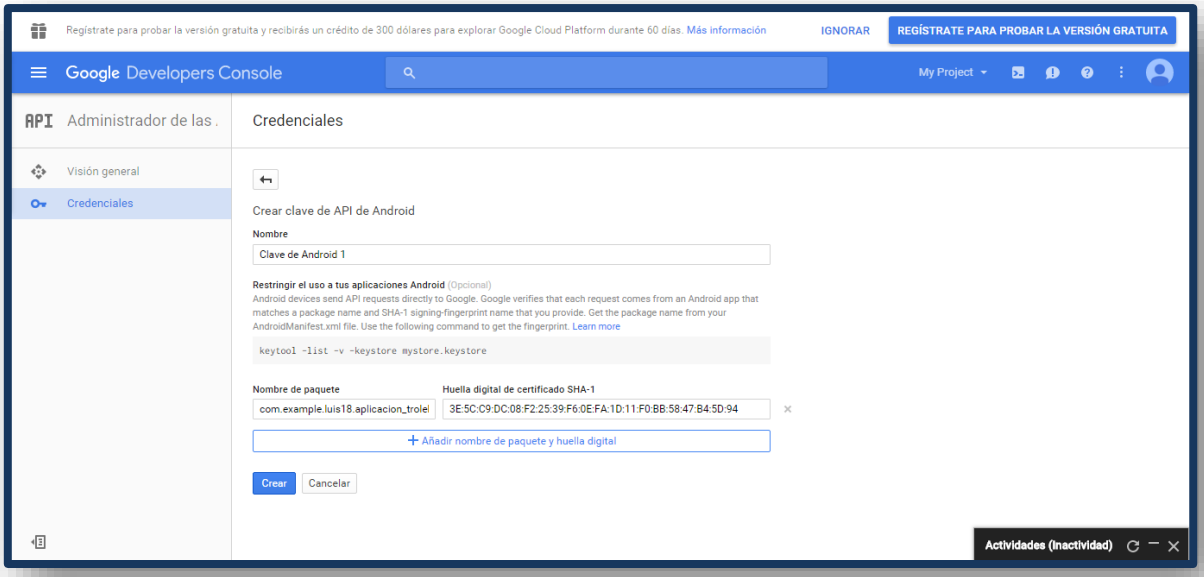


Figura 3.4-9 Creación de la llave de Google Maps

8.- Se obtiene finalmente la llave requerida que concede el permiso para poder utilizar los mapas necesarios en la aplicación.

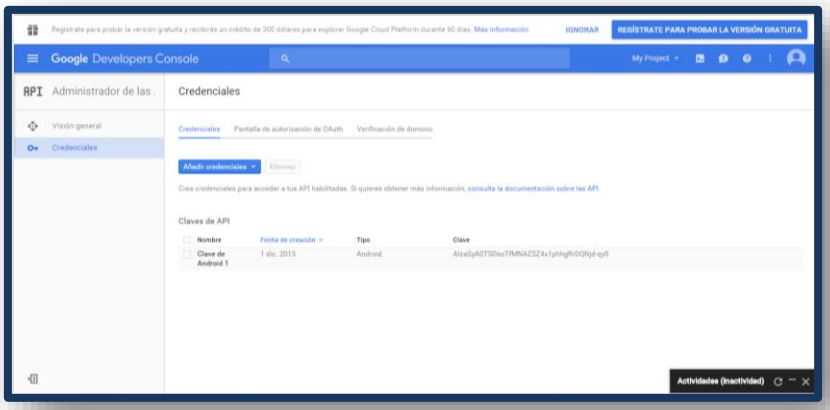


Figura 3.4-10 Administrador de las credenciales

9.- La llave conseguida se declara de la siguiente manera en el archivo *google_maps_api.xml*:

```
    <string name="google_maps_key" translatable="false" templateMergeStrategy="preserve">
        AIzaSyBKiN4WtqpXch6SaIoT1Ts0EKfXjicXqAs
    </string>
</resources>
```

Figura 3.4-11 Declaración de la llave en google_maps_api.xml

De igual manera se tienen que declarar los permisos en el archivo manifest necesarios para poder hacer uso de internet y se autorice la descarga del mapa de Google Maps, además de otros permisos que concedan la localización del usuario. Estos permisos se muestran en la figura 3.4:12.

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

Figura 3.4-12 Permisos de acceso a internet y Google Maps

3.4.2 Diseño de la interfaz.

La parte visual de la aplicación se declara en un archivo .XML, de tal manera que se compone de 4 elementos: Un elemento Fragment, en donde se despliega el mapa con el cual va trabajar la aplicación, el usuario podrá visualizar su posición y la del trolebús; otro elemento EditText que es la barra en donde se ingresa alguna dirección para proceder posteriormente a su búsqueda; y dos botones nombrados buscar y trole, en donde la función del primer botón es la de enfocar y generar un marcador en la locación que se desee buscar, por su parte el segundo al ser presionado redirigirá el mapa al lugar donde circulan los trolebuses. La distribución de estos elementos se realiza sobre un RelativeLayout, configurando este para que se organicen los elementos en forma vertical.

Figura 3.4-13 Archivo XML de la interfaz de usuario

En la figura 3.3:12 se muestra con rectángulos de colores cada elemento de la interfaz de la aplicación, esto con el objetivo de identificar como se observa la pantalla de la aplicación de usuario con respecto al código.

Rojo = RelativeLayout.

Morado = Fragment.

Amarillo = Button “Buscar”.

Azul = Button “Trole”.

Verde = EditText

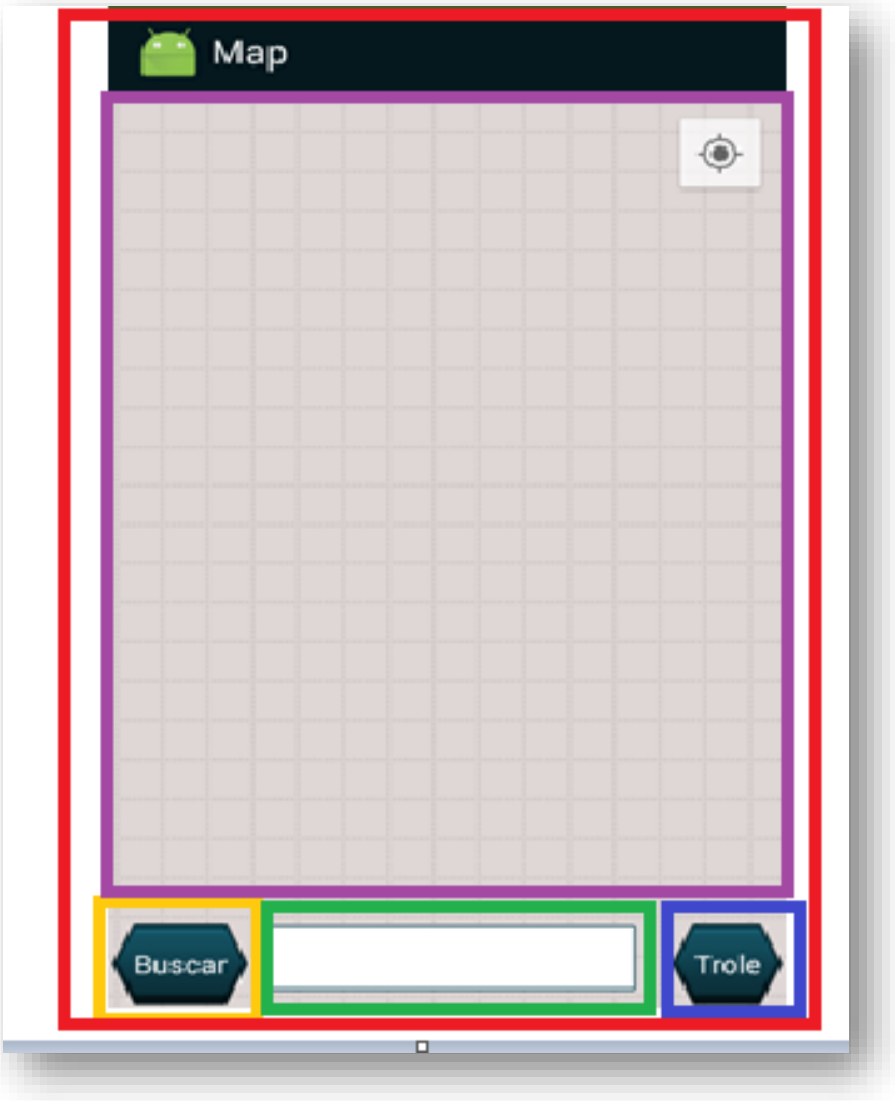


Figura 3.4-14 Interfaz de usuario

La implementación del usuario se basa en dos métodos fundamentales y en la creación de un objeto llamado mMap que es usado en estos, los cuales son declarados dentro del archivo java, y que son mostrados a continuación en la figura 3.4:15.

```
private GoogleMap mMap;

setUpMapIfNeeded();
trolezona();
```

Figura 3.4-15 Método mMap

El método s*etUpMapIfNeeded* va proveer a la aplicación con el mapa, y en caso de que este no se haya cargado repetir la acción hasta conseguirlo, para posteriormente llamar al método *SetUpMap*. El código es el siguiente:

```
private void setUpMapIfNeeded() {
    // Se realiza una condición para revisar que el mapa allá cargado
    if (mMap == null) {
        //Se obtiene el mapa de SuportFragment.
        mMap = ((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map))
                .getMap();
        //se obtiene el mapa de SuportFragment.
        if (mMap != null) {
            setUpMap();
        }
    }
}
```

Figura 3.4-16 Método setUpMapIfNeeded

Como se mencionó anteriormente, en caso de que se haya conseguido con éxito la descarga del mapa, se hace el llamado al método *SetUpMap*. Este método va desplegar un marcador para el usuario y otro para el servidor (trolebús), además de proporcionar un botón para dirigirse a la ubicación del usuario.

```
    private void setUpMap() {

//Se habilita la ubicación del usuario, la cual se muestra en el mapa junto con un botón de retorno
        mMap.setMyLocationEnabled(true);
        if(trac!=null&&suck!=null){

            mMap.clear();
//Se realiza el cambio de tipo de variable, de String a Double
            Double latitude = Double.parseDouble(trac);
            Double longitude =Double.parseDouble(suck);
//Se crea el marcador con las coordenadas recibidas, para mostrar la ubicación del trolebús.
            mMap.addMarker(new MarkerOptions().position(new LatLng(latitude, longitude)).title("Ubicación Trolebús!"));

        }

    }
```

Figura 3.4-17 Método SetUpMap

El método *trolezona* realiza un enfoque en la aplicación, ya que se realiza un acercamiento en la zona de tránsito de los trolebuses, lo cual es explicado en la figura 3.4:18.

```
    private void trolezona(){
//Se declaran las coordenadas de la zona de circulación de trolebuses
        Double latitude= 19.501654935817502;
        Double longitude = -99.13990433471832;
//Se crea la variable que contiene las coordenadas
        LatLng latLng = new LatLng(latitude, longitude);
//Se realiza un acercamiento en el mapa de acuerdo  a la variable con las coordenadas
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
//Se define el valor del acercamiento de la cámara
        mMap.animateCamera(CameraUpdateFactory.zoomTo(15));
    }
```

Figura 3.4-18 Método trolezona

Posteriormente se tiene que en la interfaz del usuario se hace uso de dos botones, los cuales son implementados mediante dos métodos llamados *on_Trole* y *onSearch*.

Cada vez que el botón (Trole) es habilitado, el método *on_Trole* realiza un enfoque a la zona de trolebuses.

```
    public void on_Trole(View view) {
//Se declaran las coordenadas de la zona de circulación de trolebuses
        Double latitude = 19.501654935817502;
        Double longitude = -99.13990433471832;
//Se crea la variable que contiene las coordenadas
        LatLng latLng = new LatLng(latitude, longitude);
//Se realiza un acercamiento en el mapa de acuerdo  a la variable con las coordenadas
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));
//Se define el valor del acercamiento de la cámara
        mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));
        mMap.animateCamera(CameraUpdateFactory.zoomTo(15));
    }
```

Figura 3.4-19 Método on_Trole

Por otra parte, el método *on_Search* realiza el enfoque a la zona que se le ha ingresado mediante el buscador, de igual manera cada vez que su botón es habilitado.

```
public void onSearch(View view) {

//Se guarda la dirección ingresada
    EditText location_tf = (EditText)findViewById(R.id.TFaddress);
//Se cambia el texto ingresado a una variable tipo String
    String location = location_tf.getText().toString();
    List<Address> addressList;
    if(!location.equals("")){
//Se crea el objeto que  permitirá obtener las coordenadas de la dirección ingresada
        Geocoder geocoder = new Geocoder(this);
        try {
//Las coordenadas obtenidas se guardan en el objeto adress
            addressList = geocoder.getFromLocationName(location, 1);
            Address address = addressList.get(0);
//Se transfieren las coordenadas al objeto latLng
            LatLng latLng = new LatLng(address.getLatitude() , address.getLongitude());
//Se crea el marcador con las coordenadas recibidas
            mMap.addMarker(new MarkerOptions().position(latLng).title(location));
//Se define el valor del acercamiento de la cámara
            mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));

        } catch (IOException e){
            e.printStackTrace();
        }
    }else {
        Toast toast=Toast.makeText(this,"Inserte locación",Toast.LENGTH_SHORT);
        toast.show();
    }

}
```

Figura 3.4-20 Método onSearch

3.4.3 Socket del usuario para la recepción de las coordenadas.

En el caso de la aplicación de usuario se implementa el socket del tipo multicast, el cual consulta el puerto y la dirección de clase d para posteriormente adquirir las coordenadas, las cuales al ser recibidas como una variable String (cadena) debe ser separada en variables independientes (latitud y longitud), mediante la implementación del método nextToken para posteriormente mandarlas a imprimir en pantalla; dicho código es similar al socket del servidor.

```
private class UDP extends AsyncTask<String, Void, String> {

    @Override
    protected String doInBackground(String... params) {
        int puerto = 9010;
        String linea;

        try {
            MulticastSocket socketCliente = new MulticastSocket(puerto);
            InetAddress Host = InetAddress.getByName("228.5.6.7");
            socketCliente.joinGroup(Host);

            while (true) {
                byte[] bufer = new byte[1024];
                DatagramPacket mensajeEntrada =
                        new DatagramPacket(bufer, bufer.length);
                socketCliente.receive(mensajeEntrada);
                linea = new String(mensajeEntrada.getData(),0, mensajeEntrada.getLength());
                traduc = linea;

                StringTokenizer coo=new StringTokenizer(traduc,",");
                String Lat=coo.nextToken();
                String Lon=coo.nextToken();
                trac=Lat;
                suck=Lon;


                socketCliente.close();
                if (mensajeEntrada.equals("Adios")) break;
            }
            socketCliente.leaveGroup(Host);
```

Figura 3.4-21 Socket para la recepción de coordenadas

Para el correcto funcionamiento del socket del usuario se tiene que incorporar un timer en la clase principal, ya que, sin este el proceso de recepción y envió solo se ejecutaría una sola vez, esto garantiza que la información adquirida del GPS se actualizase cada segundo.

```
public void startTimer() {
    //Lanza un nuevo Timer
    timer = new Timer();

    //Inicailiza  el metodo TimerTask
    initializeTimerTask();
    timer.schedule(timerTask, 1000, 1000); //
}

public void initializeTimerTask() {
    //Se define el timer
    timerTask = (TimerTask) () -> {
            try {
                handler.post(() -> {
                        setUpMap();
                        //mMap.clear();
                        new UDP().execute();
                });
            } catch (Exception e) {
                Toast toast = Toast.makeText(getApplicationContext(),
                        e.toString(), Toast.LENGTH_SHORT);
                toast.show();
            }
    };
}
```

Figura 3.4-22 Timer de Socket

3.5. Sumario

El presente capítulo explica cómo fue el desarrollo de la aplicación prototipo para la ubicación del trolebús en tiempo real, de la misma manera se tratan los temas de como se hace la comunicación entre las aplicaciones que se desarrollan y de la misma manera se observa como Android requiere permisos específicos para que dé acceso a la aplicación prototipo a la ubicación de cada dispositivo.

Se muestra como se hace el diseño de la interfaz de usuario de ambas aplicaciones prototipo, de la misma manera se hizo una explicación detallada de cómo se relaciona la interfaz de usuario con el código creado en los archivos xml correspondientes.

Las aplicaciones prototipo de usuario y trolebús quedan listas para proceder a las pruebas y observar las decadencias y aciertos, todo esto se explica en el siguiente capítulo.

Capítulo 4. Pruebas de las aplicaciones prototipo

4.1. Pruebas de la aplicación prototipo del trolebús.

En este capítulo se muestran los resultados del prototipo en funcionamiento. Las pruebas fueron realizadas en la avenida Manuel de Anda y Barredo, desde la parada ubicada en el edificio 4 de ESIME ZACATENCO hasta la parada del edificio 3 del mismo.

Por parte de la aplicación servidor, primeramente, la aplicación comienza buscando las coordenadas del trolebús, que se ilustra en la figura 4.0:1.

Figura 4.0-1Búsqueda Iniciada de las Coordenadas del Trolebús(Servidor)

Posteriormente una vez obtenidos los datos se observan la latitud y longitud pertenecientes al trolebús, que son enviadas a través de un socket multicast, conforme el trolebús avance las coordenadas se irán actualizando, visualizando en pantalla la imagen con las coordenadas de la figura 4.0:2.

Figura 4.0-2 Despliegue de las Coordenadas del Trolebús (Servidor)

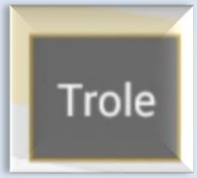
Todo lo anterior, fue realizado por una persona la cual llevaba un celular, mismo que efectuó la acción de servidor. En la siguiente figura se muestra el momento en el que la persona aborda el trolebús y empieza a transmitir coordenadas.

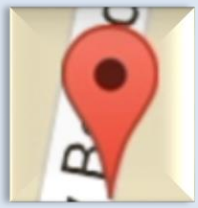
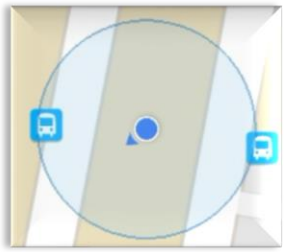
Figura 4.0-3 Momento en el que el Servidor aborda el trolebús

4.2. Pruebas de la aplicación prototipo de usuario.

En cuanto a la aplicación de usuario se tiene la siguiente interfaz:

Tabla 4.2–1 Elementos de la interfaz de usuario

Descripción	**Repesentacion gráfica en la Interfaz**
1.-Un botón que direcciona la aplicación a la ubicación del usuario.	
2.-Un buscador de locaciones, ubicado en la parte inferior izquierda.	
3.-Un botón que direcciona la aplicación a el área de circulación de trolebuses	

4.-Señalizacion de las paradas de los trolebuses	
5.-Marcador de la ubicación del trolebús	
6.-Indicador de la posición del usuario	

Al arranque de la aplicación se muestra la zona de circulación de trolebuses como se muestra en la figura 4.2:1.

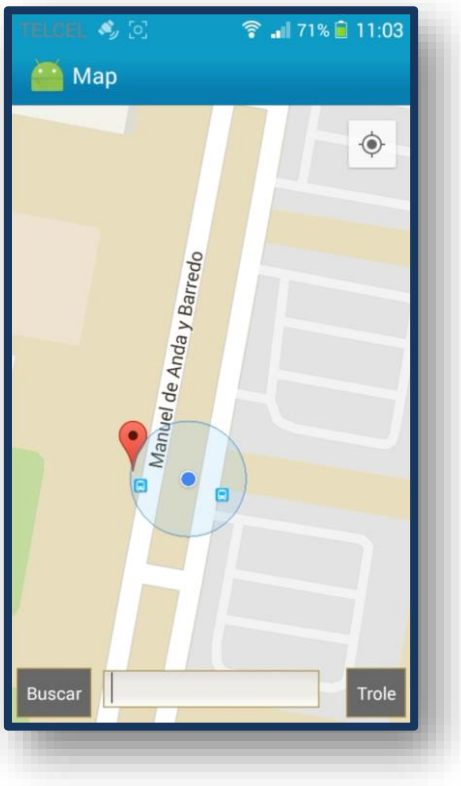


Figura 4.2-1 Circuito de Trolebuses

En seguida la aplicación ubica al usuario y lo señala con un punto azul, junto con el rango de error que se señaliza con un círculo azul claro alrededor del punto (Figura 4.2:2)

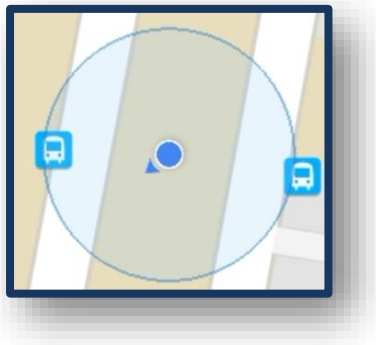


Figura 4.2-2 Ubicación del Usuario

Una vez que se obtienen las coordenadas provenientes del trolebús (servidor) por medio del socket, la aplicación despliega un marcador que indica la posición del colectivo (Figura 4.2:3)

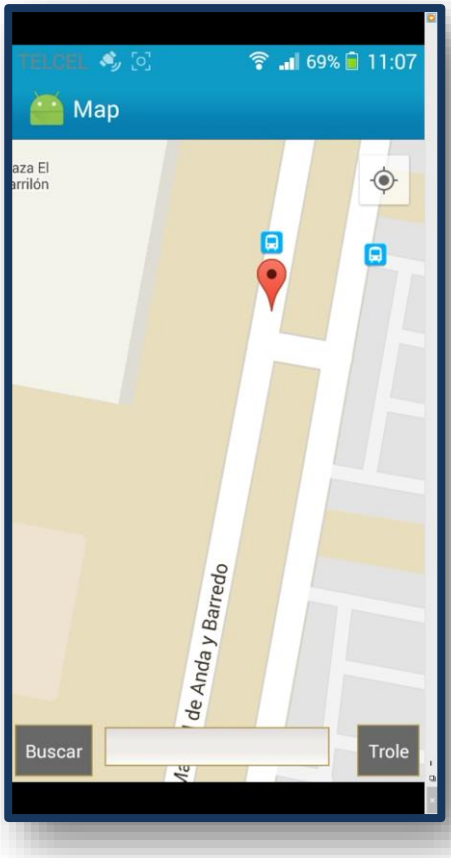


Figura 4.2-3 Ubicación del Trolebús

A su vez, se muestra al transporte en movimiento, dentro de él, una persona con la aplicación servidor enviando las coordenadas.

Figura 4.2-4 Trolebús y la aplicación de servidor dentro él

4.3. Valor agregado de la aplicación (Tiempo de llegada del Trolebús)

Existen diferentes métodos para calcular el tiempo aproximado de llegada del trolebús, uno de ellos es la implementación del lenguaje de programación Java Script, el cual sirve para acceder directamente a los servidores de Google y obtener dicha información.

Otra forma la cual es la queda se implementó como valor agregado en este proyecto es por medio del trazo manual de diversos puntos en el mapa de Google la cual indique la ruta exacta del trolebús y a partir de la misma calcular el tiempo por medio de la formula física:

$$v = \frac{d}{t}$$

Por medio de un despeje se tiene:

$$t = \frac{d}{v}$$

Para esto se tuvo que modificar el método **public void onLocationChanged (Location sat)** de la clase **Location Listener** de la aplicación (servidor), quedando de la siguiente manera:

```
@Override
public void onLocationChanged(Location sat) {
    sat.getLatitude();
    sat.getLongitude();
    double vel= sat.getSpeed()*3.6;
    String Texto = "Mi ubicación actual es: " + "\n Lat = "
            + sat.getLatitude() + "\n Long = " + sat.getLongitude()+"\nVel= "+vel+" Km/hrs";
    String Exito=sat.getLatitude()+","+sat.getLongitude()+","+sat.getSpeed();
    messageTextView.setText(Texto);
    ////////////////////////////////////////////
    this.coor.setLocation(sat);
    posicion = Exito;
}
```

Figura 4.3-1 Código modificado de la aplicación del Trolebús.

Cabe señalar que en esta ocasión la aplicación servidor enviara la información de la velocidad al cliente.

La aplicación del trolebús (Servidor) mostrara la siguiente interfaz:

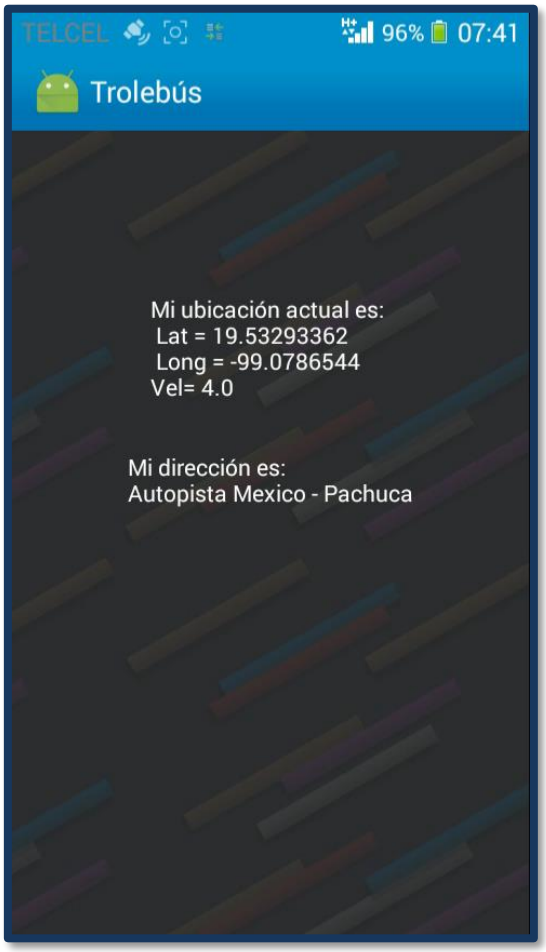


Figura 4.3-2 Interfaz de la aplicación del trolebús mostrando la velocidad

Nota: La velocidad que muestra la aplicación está dada en kilómetros por hora.

Por otra parte, en la aplicación del cliente se tienen que agregar dos métodos para el cálculo del tiempo, el primer método que se declara es la parte de la distancia que hay desde del trolebús hasta la localización del usuario quedando de la siguiente manera:

```
public void Distancia(){
    if(suck!=null&&trac!=null){
        Double latitude = Double.parseDouble(trac);
        Double longitude =Double.parseDouble(suck);
        Location punto=new Location("Location 1");
        Location.distanceBetween(punto.getLatitude(),punto.getLongitude(),latitude,longitude,dis);
        //Location.distanceBetween(19.603363, -99.030541,19.604786, -99.030153,dis);
    }
}
```

Figura 4.3-3 Código para calcular la distancia del trolebús al usuario

El segundo método que se utiliza sirve para calcular el tiempo con la formula física anterior mente mencionada.

```
public void Tiempo(){
    if(puck!=null){
        double vel=Double.parseDouble(puck);
        if(vel>0){
            i++;
            velp=(velp+vel)/i;
        }
        estimado=((dis[0]*1E-7)/velp)/60;
    }
}
```

Figura 4.3-4 Código para calcular el tiempo del trolebús al usuario

La interfaz que muestra la aplicación de usuario con las modificaciones hechas seria la siguiente:

Figura 4.3-5 Interfaz de la aplicación de usuario modificada

En la imagen se puede observar que el tiempo estimado es demasiado grande, esto se debe a la precisión que Android da de forma predeterminada, para corregir el error la distancia se debe de

multiplicar por el siguiente valor 1x10^-7, esto para que la distancia sea en metros y reducir las cifras significativas de la medida, el tiempo en la aplicación está dada en minutos.

Estas pruebas para saber el tiempo aproximado no se hicieron en la zona del trolebús se hicieron en la carretera México-Pachuca, esto no afecta de ninguna manera la aplicación, si en cambio los tiempos arrojados son aproximadamente correctos, pero cabe señalar que para poder ser más preciso en el tiempo que tardara el trolebús en llegar a la ubicación del usuario depende de muchos factores externos aparte de la distancia, esto hace que el error sea cada vez más grande, pero si se desea, la aplicación está abierta a alguna actualización para mejorar dicha precisión.

4.4. Observaciones.

Durante las pruebas realizadas, se pudo percatar que la aplicación de servidor (Trolebús) hizo un gasto alto dentro de la batería. El motivo de ello es el constante uso del GPS, que trae como consecuencia un consumo de energía mayor, en comparación con otras aplicaciones. A continuación, se muestra una imagen en la cual se demuestra el consumo de energía determinado por el sistema operativo Android.

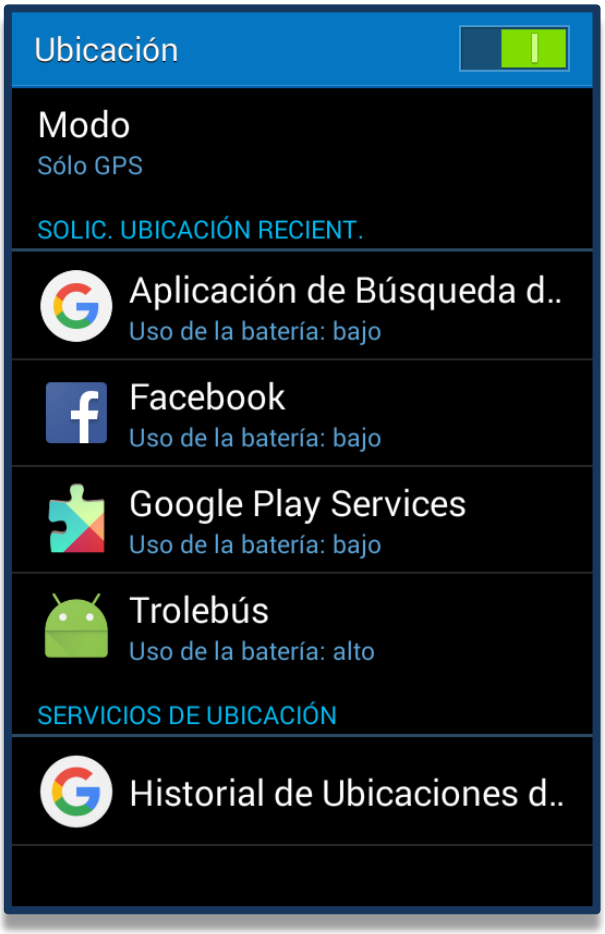


Figura 4.4-1 Consumo de batería de la Aplicación

4.5. Análisis de costos

Los costos generados en este prototipo se contemplan desde el diseño del prototipo, hasta los elementos que integran a este. Es importante recalcar que los costos totales generados son aplicables para la elaboración del primer prototipo.

4.5.1 Costo del dispositivo de la transmisión de datos

La elección del celular como transmisor de las coordenadas se fundamentó principalmente en la parte económica, ya que un kit de Arduino tiene un valor monetario considerablemente mayor, lo cual puede ser apreciado en la tabla siguiente, además de la comparación de algunas otras características relevantes entre ambos dispositivos (Para mayor información acerca del dispositivo móvil y del kit de arduino, revisar los apéndices 7 y 8 respectivamente).

Tabla 4.5–1 Comparativa entre Arduino y Celular

Comparativa Arduino Vs Celular		
Características	Celular Modelo: Pixi 3	Arduino uno kit
Imagen del Dispositivo		
Precio	$900	$1,584
Internet	GSM/HSPA/LTE	GSM
Display	Sí	No Incluye
Memoria	4Gb,Expandible	32Kb
Peso	110g	70g
Bateria	1400mAh	No Incluye
Velocidad de Procesamiento	1GHz,por núcleo	16MHz

4.5.2 Costo por envió de información de las aplicaciones

Se considera que se envía una cadena de 22 caracteres por lo tanto se enviaran 22 bytes de información en la aplicación del trolebús.

Tabla 4.5–1 Costo de la aplicación de Trolebús

Tiempo	Datos en (Mb)	Costo de conexión por (Mb)
1seg	0.000022	0.000022
1min	0.00132	0.00132
1hrs	0.0792	0.0792
1dia	1.9008	2
1mes	57.78432	58
1año	693.41184	693

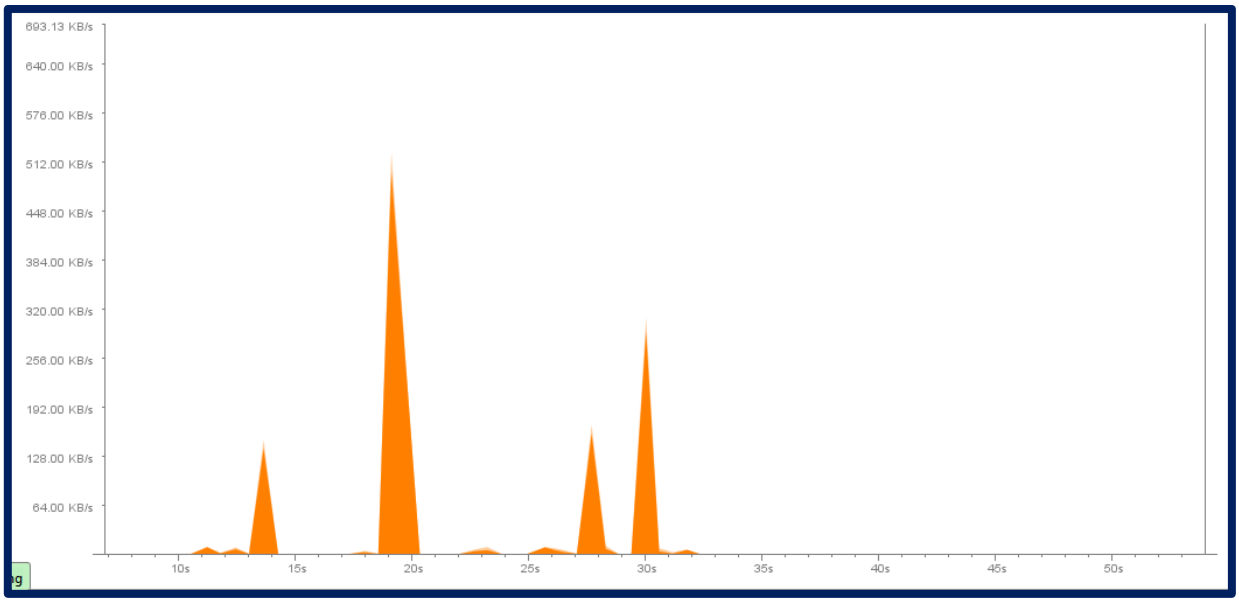


Figura 4.5-1 Costo de la aplicación de Usuario

Con base a la gráfica que el mismo Android Studio muestra acerca de la velocidad de descarga por segundo de la aplicación, se tiene que al arranque de la aplicación se ocupa un mega byte de información, ya que se debe de realizar la descarga de los mapas, posteriormente se ocupa la misma tasa de transferencia de la aplicación del trolebús. La tabla de costo de la aplicación se muestra en la siguiente tabla:

Tabla 4.5–2 Costo de la Aplicación de Usuario

Tiempo	Datos en (Mb)	Costo de conexión po (Mb)
1seg	1.000022	1
1min	1.001342	1
1hrs	1.080542	1
1dia	2.981342	3
1mes	60.765662	61
1año	754.177502	754

4.6. Costos de ingeniería

Para llevar a cabo el desarrollo del prototipo se necesita de personal capacitado en el área del lenguaje de programación, específicamente en el apartado de Java. En este caso se consideró a un Ing. en Control y Automatización, el cual cuenta con una buena base en el área de programación, con 1 año de experiencia, que lo califica para llevar a cabo este proyecto. El costo que se sugiere para el personal esta expresado en la siguiente tabla:

Tabla 4.5–1 Costo de ingeniería

Cantidad	Profesión	Horas de trabajo	Importe
1	Ing. en Control y Automatización	8	$ 13,019
		16%	$ 2,479
		$ 15, 498	

*Los sueldos mostrados están sujetos a cambios, la cotización del sueldo está basado en estadísticas realizadas por una página de orientación laboral mexicana llamada mi salario.org. Para más información *véase Referencias.*

4.7. Costo total.

Una vez que se conoce el costo individual de cada elemento que compone al proyecto, se procede a realizar el cálculo del costo total. En la siguiente tabla se muestra el cálculo realizado:

Tabla 4.5–1 Costo total

Área	Importe
Ingeniería (horas hombre)	$ 15, 080
Costo de envío de información	$ 1, 447
Costo de dispositivo	$ 900
TOTAL	$ 17, 427

4.8. Propuesta para la mejora a largo plazo del prototipo

El prototipo que fue anteriormente detallado, se planea convertirlo en un proyecto más fiable. La razón de hacerlo es el mejorar su desempeño y ser mucho más confiable a la hora de recibir y enviar datos. Las propuestas que se contemplan para su mejora es mediante la adquisición de un kit, que comprende los siguientes elementos:

- Un dispositivo GPS
- Una placa Arduino UNO
- Un módulo GSM.

Esto tendrá como finalidad reemplazar al celular, que ya se había mencionado con anterioridad su implementación para él envió de coordenadas del trolebús.

El kit mencionado, será ubicado en la parte frontal del tablero, de esta manera se eliminará el uso del celular y se integrará de forma permanente al Trolebús. Otra ventaja que ofrecerá el uso de este kit es la nula dependencia de las baterías que en contraste con el celular este kit estará en función de la energía que proporciona el trolebús.

Figura 4.8-1 Trolebús

4.9. Sumario

En el presente capítulo se redactan las pruebas que se realizaron con ambas aplicaciones prototipo observando la eficiencia y la exactitud que tienen para determinar la posición del usuario y del trolebús en tiempo real.

Se hizo un análisis de las interfaces de usuario de cada software determinando cada elemento junto con sus ubicaciones en la pantalla del dispositivo, también se realizó la comparativa entre un microcontrolador y un celular, se anexan los costos de datos móviles que tendrían las aplicaciones.

Conclusiones.

El uso de aplicaciones en los diversos dispositivos electrónicos, hoy en día es muy útil para el ser humano, ya que ayuda a facilitar las tareas o incluso agilizar procesos, como es el caso de la aplicación generada en este proyecto, que gracias a ella se ataca un problema que se tiene en las diferentes instituciones de la unidad académica Zacatenco el cual se describe en el desarrollo de este escrito. Derivado de los resultados obtenidos en este trabajo, se puede concluir lo siguiente:

La programación en Android es puramente la implementación de paqueterías creadas por Google las cuales se anexan al lenguaje de programación Java. La aplicación funciona de manera correcta, siempre y cuando se tenga un dispositivo móvil con acceso a datos (conexión a internet).

Para implementar por completo el proyecto cabe señalar que cada trolebús necesitaría llevar un localizador (teléfono inteligente) para el funcionamiento del proyecto, el prototipo como tal está enfocado para el trolebús, pero viéndolo desde la perspectiva de negoció se puede implementar en diferentes redes de transporte en la Ciudad de México, pudiendo venderse a los dueños de las rutas de los autobuses que lo deseen implementar. El costo de datos dependería de la institución que compre el servicio, cabe resaltar que el costo por la transmisión de datos sería muy bajo ya que solo se envía una cadena de alrededor de 112 bytes de información/seg, por lo tanto, a pesar de ser muy barato también indica que no se necesita como tal una conexión 4G para satisfacer las necesidades de envío y recepción de datos, la aplicación como tal depende directamente de la conexión a internet sin importar que sea Wi-Fi o datos móviles, la aplicación funcionara siempre y cuando haya un punto de acceso a internet.

Derivado de este trabajo, se identificaron limitantes dentro de las cuales la
limitante más agresiva para este prototipo, fue la fuente de energía (eléctrica) para la ubicación del transporte, la cual es tomada de la batería de un teléfono inteligente (Smartphone), que hace depender directamente de ella, en el mismo sentido, la incursión a un nuevo lenguaje de programación (Java) nos impidió potencializar y hacer más eficiente el proyecto.

Glosario.

Handover: Es el sistema utilizado en comunicaciones móviles celulares con el objetivo de transferir el servicio de una estación base a otra cuando la calidad del enlace es insuficiente en una de las estaciones. Este mecanismo garantiza la realización del servicio cuando un móvil se traslada a lo largo de su zona de cobertura.

Red NMT: Nordisk MobilTelefoni o Nordiska MobilTelefoni-gruppen, Telefonía Móvil Nórdica en español) es un sistema de telefonía móvil definido por las autoridades de telecomunicaciones escandinavas.

NTT (Nippon Telegraph and Telephone Corporation) : También conocida como NTT es una empresa de telecomunicaciones líder en el mercado nipón. Compañía estatal hasta su privatización en 1985.

GUI (Graphical User Interface): Es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Google Play: Fue originalmente llamado Android Market, es la tienda oficial de Google para aplicaciones de Android y otro tipo de contenido para smartphones, tablets y dispositivos Android TV.

Timation: Fue un satélite, desarrollado y lanzado por el Laboratorio Naval de Investigación in Washington en 1964.

TCP (Transmission Control Protocol): es uno de los protocolos fundamentales del internet.

UDP (User Datagram Protocol): Es un protocolo del nivel de transporte basado en el intercambio de datagramas.

API (*Application Programming Interface):* Es el conjunto de subrutinas, funciones y procedimientos (o métodos, en la programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

GPS : Sistema de Posicionamiento Global.

IDE (*Integrated Development Environment):* Un **ambiente de desarrollo integrado** o **entorno de desarrollo interactivo**, es una aplicación informática que proporciona servicios integrales para facilitarle al desarrollador o programador el desarrollo de software.

JDK (Java Development Kit): Es un *software* que provee *herramientas de desarrollo* para la creación de *programas* en *Java*. Puede instalarse en una *computadora* local o en una unidad de red.

RAM (Random Access Memory): Memoria de Acceso Aleatorio.

SDK (Software Developers Kit): En español mejor conocido por Kit de Desarrollo de Software. Es generalmente un conjunto de herramientas de desarrollo de software que le permite al programador o desarrollador de software crear aplicaciones para un sistema concreto, por ejemplo ciertos paquetes de software, frameworks, plataformas de hardware, computadoras, videoconsolas, sistemas operativos, etcétera.

Referencias

[1] Jesús Tomas Tirones. Julio 2012. El gran libro de Android. 2° Edición. Alfaomega,.

[2] Victoria López, Introducción a Android. 1 Edición. Madrid. E.M.E Editorial.

[3] Ing. Pablo Augusto Sznajdleder. 2013. Java a Fondo. 2ª Edición. Buenos Aires. Alfaomega.

[4] Support Library. Obtenida el 7 de Octubre de 2015. https://developer.android.com/sdk/index.html

[5] Stephanie Falla Aroche. Curso de Android: Todo lo que necesitas para empezar. Consultado el 3 de Septiembre de 2015. http://www.maestrosdelweb.com/editorial/curso-android

[6] Datagram Socket, Obtenido el 5 de Noviembre de 2015. http://developer.android.com/intl/es/reference/java/net/DatagramSocket.html

[7] Compara tu salario, Wage inidicator, Obtenido el 5 de noviembre 2015. Disponible en: http://www.misalario.org/main/tu-salario/comparatusalario?job-id=2149990000000

Apéndice 1: Código fuente del archivo MainActivity.java de la aplicación del Trolebús

```
package org.tesis.esime.localizaciongps;

/**
 * Created by OrlandoMS on 11/oct/2015.
 */
import java.io.IOException;
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.List;
import java.util.Locale;
import java.util.Timer;
import java.util.TimerTask;

import android.location.Address;
import android.location.Geocoder;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.AsyncTask;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.os.Handler;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;


public class MainActivity extends Activity {

    //Variables declaradas en el programa
    Timer timer;
    TimerTask timerTask;
    final Handler handler = new Handler();
    String posicion;
    TextView messageTextView;
    TextView messageTextView2;

    @Override
    //Se genera el metodo onCreate la cual lanza las actividades de la aplicación
    protected void onCreate(Bundle savedInstanceState) {
        //Se manda a llamar el contructor de el metodo
        super.onCreate(savedInstanceState);
        //Se identifica el nombre de la actividad
        setContentView(R.layout.activity_main);
        /*Se asignan los nombres a las variables messageTextView, las cuales sirven
para imprimir
        * en la pantalla de el dispositivo cualquier mensaje*/
        messageTextView  = (TextView) findViewById(R.id.message_id);
        messageTextView2 = (TextView) findViewById(R.id.message_id2);

        /*Mandamos a declarar la varible mloListener de tipo Trolocalización la
        cual invoca el contructor de dicha clase*/
        Trolocalizacion mlocListener = new Trolocalizacion();
        /*La valiable antes creada invoca el metodo Localización y la rellena con si
misma*/
        mlocListener.Localizacion(this);
```

```
        startTimer();
    }
//Timer de refrescamiento
    public void startTimer() {
        //La variable timer lanza el metodo Timer
        timer = new Timer();

        //Inizializa el metodo Timer
        initializeTimerTask();

        /*Schedule  es una metodo que repite deforma periodica una tarea y se
ejecuta despues de un
        retardo especifico despues de 1000ms el TimerTask correra cada 1000ms */
        timer.schedule(timerTask, 1000, 1000); //
    }
//El metodo detiene el timer
    public void stoptimertask(View v) {
        //stop the timer, if it's not already null
        if (timer != null) {
            timer.cancel();
            timer = null;
        }
    }
// Se hace un metodo el cual hara las tarea del timer task.
    public void initializeTimerTask() {
        //El timerTask manda a llamar el metodo TimerTask.
        timerTask = new TimerTask() {
            @Override
            //Con el siguiente método ejecuta las tareas deseadas
            public void run() {
                try {
                    /*La siguente linea handler perite enviar y procesar mensajes
                    * ejecutables como un metodo Runnable()*/
                    handler.post(new Runnable() {
                        //Inicia la ejecución de la parte activa del codigo
                        public void run() {
                            /*Se ejcuta el contructor del metodo de la clase
Socket_UDP y rellena
                            * la variable propia coordenada con los valores de
posición*/
                            new Socket_UDP(posicion).execute("");
                        }
                    });
                }
                //En caso de que se produsca un error el codigo ejecuta una fase de
error
                catch (Exception e){
                    Toast toast = Toast.makeText(getApplicationContext(),
                            e.toString(), Toast.LENGTH_SHORT);
                    toast.show();
                }
            }
        };
    }

    public void setLocation(Location loc) {
        //Obtener la dirección de la calle a partir de la latitud y la longitud
        if (loc.getLatitude() != 0.0 && loc.getLongitude() != 0.0) {
            try {
                Geocoder geocoder = new Geocoder(this, Locale.getDefault());
                List<Address> list = geocoder.getFromLocation(
                        loc.getLatitude(), loc.getLongitude(), 1);
                if (!list.isEmpty()) {
```

```
                    Address address = list.get(0);
                    messageTextView2.setText("Mi dirección es: \n"+
address.getAddressLine(0));
                }

            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    /* Clase Trolocalización es la encargada de obtener las cordenadas */
    public class Trolocalizacion implements LocationListener {
        /*Se generan vairables de tipo LocationManager que permitiran acceder a la
        caracterisitcas de los LocationListener*/
        MainActivity coor;
        LocationManager mlocManager,manejador;

        //Se genera el metodo localización la cual contiene una variable locate del
tipo MainActivity
        public void Localizacion(MainActivity locate) {
            /*Se da la orden de cada variable para poder hacer usos de los
            servicios de lozalización*/
            mlocManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
            manejador = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
            /*En ambas condiciones if se puede observar el tipo de provedor al cual
            se dara el acceso para obtener las coordenadas*/
            if(mlocManager.isProviderEnabled(LocationManager.GPS_PROVIDER)) {
                mlocManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
                        0,0, this);
            }


            if(manejador.isProviderEnabled(LocationManager.NETWORK_PROVIDER)) {
                manejador.requestLocationUpdates(LocationManager.NETWORK_PROVIDER,
                        0,0, this);
            }

            this.coor= locate;
            messageTextView.setText("LocationListener agregado\nBuscado
coordenadas...");
            messageTextView2.setText("");

        }

        @Override
        /*Este metodo se ejecuta cada vez que el GPS recibe nuevas coordenadas
        * debido a la deteccion de un cambio de ubicacion*/
        public void onLocationChanged(Location sat) {
            /*Los metodos getLatitud() y getLongitude() son los encargados de
obtener las
            coordenadas de longitud y latitud respectivamente*/
            sat.getLatitude();
            sat.getLongitude();
            /*Se mandan a imprimir las coordenadas en la pantalla del dispositivo*/
            String Texto = "Mi ubicación actual es: " + "\n Lat = "
                    + sat.getLatitude() + "\n Long = " + sat.getLongitude();
            String Exito=sat.getLatitude()+","+sat.getLongitude();
            messageTextView.setText(Texto);
            ////////////////////////////////////////
            this.coor.setLocation(sat);
```

```
            posicion = Exito;
        }

        @Override
        public void onProviderDisabled(String provider) {
            // Este metodo se ejecuta cuando el GPS es desactivado
            messageTextView.setText("GPS Desactivado");
        }

        @Override
        public void onProviderEnabled(String provider) {
            // Este metodo se ejecuta cuando el GPS es activado
            messageTextView.setText("GPS Activado");
        }

        @Override
        public void onStatusChanged(String provider, int status, Bundle extras) {
            /* Este metodo se ejecuta cada vez que se detecta un cambio en el
               estatus del proveedor de localizacion (GPS)
               Los diferentes Status son:
               OUT_OF_SERVICE -> Si el proveedor esta fuera de servicio
               TEMPORARILY_UNAVAILABLE -> Temporalmente no disponible pero se
               espera que este disponible en breve
               AVAILABLE -> Disponible*/
        }

    }/* Fin de la clase Trolocalizacion */
public class Socket_UDP extends AsyncTask<String, Void, String> {
    String coordenada;


    public Socket_UDP(String coordenada){

        this.coordenada=coordenada;
    }
```

Se explica el program por medio de los comentarios que se hacen el mismo

```
    @Override
            protected String doInBackground(String... params) {
                //Se genera una variable de tipo entero que nos dara el
              número de puerto a enviar los datos
                int puerto = 9010;
                //Se ejecuta una instrucción try la cual tendrá el código del
              socket
                try {
                    //en este punto la variable str obtiene las corrdenadas del
                       LocationListener
                    String str =coordenada ;
                    /*En la siguente linea se crea el obeto Socket cliente la
    cual genera elsocketMulticast*/
                    MulticastSocket socketCliente = new MulticastSocket(puerto);
                    /*La variable eniviar se rellena la varible de tipo cadena
                       str la cual se obtinen los byts para ser enviada*/
                    byte[] enviar = str.getBytes();
                    /*La variable host se rellena con el numero de IP de clase d
                    a la cual sera enviado el paquete*/
                    InetAddress Host = InetAddress.getByName("228.5.6.7");
                    /* La variable send_packet es la encargada de prepara el
                       paquete para ser enviado*/
                    DatagramPacket send_packet = new DatagramPacket(enviar,
    str.length(), Host, puerto);
```

```
                    /*La siguente instruccion es la que nos permite enviar
                    el paquete a la IP y puerto asignado*/
                    socketCliente.send(send_packet);

                } catch (SocketException e) {
                    e.printStackTrace();
                } catch (UnknownHostException e) {
                    e.printStackTrace();
                } catch (IOException e) {
                    e.printStackTrace();
                }catch (Exception e){

                }
                ////////////////////
                return null;
            }

            @Override
            protected void onPostExecute(String result) {
            }

            @Override
            protected void onPreExecute() {
            }

            @Override
            protected void onProgressUpdate(Void... values) {
            }
        }

    }
```

Todo el código anterior mente descrito es del archivo MainActivity.java el cual se encuentra en la ruta AndroidStudioProjects**\LocalizacionGPS**\app\src\main\java**\org\tesis\esime\localizaciongps**, las partes que se encuentra en negritas puede varias dependiendo del nombre del proyecto que se haya generado

Apéndice 2: Código fuente del archivo AndroidManifest.xml de la aplicación del Trolebús

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="org.tesis.esime.localizaciongps">
    <!--Perimsos de Localizazion-->
    <uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-
permission>
    <uses-permission
android:name="android.permission.ACCESS_LOCATION_EXTRA_COMMANDS"></uses-permission>
    <uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
    <!--Permisos de Internet para que el socket funcione de forma correcta-->
    <uses-permission android:name="android.permission.LOCATION_HARDWARE"></uses-
permission>
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-
permission>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-
permission>
    <!--En esta parte se detallan las caracteristicas de la aplicación como el icono
o tema de presentación-->
    <application android:allowBackup="true" android:label="@string/app_name"
        android:icon="@mipmap/ic_launcher" android:supportsRtl="true"
        android:theme="@style/AppTheme">
    <!--En esta sección se dan de alta todas las actividades usadas ya que de no ser
así la aplicación marcaria un errror-->
        <activity android:name=".MainActivity"
            android:label="@string/app_name">
            <!--Las siguientes lineas sirvan para que la aplicacion se instale en
los smathphones, estas lines son en general las mismas para cualquier aplicación-->
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

El archivo AndroidManifest.xml está ubicado en la siguiente ruta AndroidStudioProjects**LocalizacionGPS**\app\src\main la parte resaltada depende del nombre que se le haya dado al proyecto

Apéndice 3: Código fuente del archivo activity_main.xml de la aplicación del Trolebús

```
<!--Se declara el tipo de Layout que se quiere usar-->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >
    <!--Se definen las caracteristicas de el primer TextView como la posicion los
margenes y el nombre-->
    <TextView
        android:id="@+id/message_id"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text=""
        android:layout_alignParentTop="true"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="100dp"
        />
    <!--Se definen las caracteristicas de el primer TextView como la posicion los
margenes y el nombre-->
    <TextView
        android:id="@+id/message_id2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/message_id"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="30dp"
        android:text="" />

</RelativeLayout>
```

La ruta de este archive es: AndroidStudioProjects**LocalizacionGPS**\app\src\main\res\layout, como se explica en el desarrollo del proyecto el layout es la interface de usuario, la parte resaltada del proyecto depende del nombre que se le a dado al proyecto

Apéndice 4: Código fuente del archivo MapsActivity.java de la aplicación del Usuario

```
package com.example.luis18.mtp;

import android.location.Address;
import android.location.Geocoder;
import android.os.AsyncTask;
import android.os.Bundle;
import android.os.Handler;
import android.support.v4.app.FragmentActivity;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import com.google.android.gms.maps.CameraUpdateFactory;
import com.google.android.gms.maps.GoogleMap;
import com.google.android.gms.maps.SupportMapFragment;
import com.google.android.gms.maps.model.LatLng;
import com.google.android.gms.maps.model.MarkerOptions;

import java.io.IOException;
import java.net.DatagramPacket;
import java.net.InetAddress;
import java.net.MulticastSocket;
import java.net.SocketException;
import java.net.UnknownHostException;
import java.util.List;
import java.util.StringTokenizer;
import java.util.Timer;
import java.util.TimerTask;

public class MapsActivity extends FragmentActivity {

    private GoogleMap mMap;

    Timer timer;
    TimerTask timerTask;
    final Handler handler = new Handler();

    String traduc,trac,suck;

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


        setUpMapIfNeeded();
        startTimer();
        trolezona();
    }
//////////////////////////////////////////Timer//////////////////////////////////////
///////////////
    public void startTimer() {
        //Lanza un nuevo Timer
        timer = new Timer();

        //Inicailiza  el metodo TimerTask
        initializeTimerTask();
/*schedule sirve para determinar el tiempo a ejecutar por primera vez, el timer que
será después de 1000ms el segundo numero indica cada cuando se vuelve a ejecutar el
timer 1000ms*/
```

```
        timer.schedule(timerTask, 1000, 1000); //
    }

    /*public void stoptimertask(View v) {
        //Para el timer
        if (timer != null) {
            timer.cancel();
            timer = null;
        }
    }*/

    public void initializeTimerTask() {
        //Se define el timer
        timerTask = new TimerTask() {
            @Override
            public void run() {
                try {
                    //En este punto se pone que proceso se va a ejecutar
                    handler.post(new Runnable() {
                        public void run() {
                            setUpMap();
                            //mMap.clear();
                            new UDP().execute();
                        }
                    });
                } catch (Exception e) {
                    Toast toast = Toast.makeText(getApplicationContext(),
                            e.toString(), Toast.LENGTH_SHORT);
                    toast.show();
                }
            }
        };
    }
////////////////////////////////////////Ubicacion////////////////////////////////////
/////////////////
    public void onSearch(View view) {


//Se guarda la dirección ingresada
        EditText location_tf = (EditText)findViewById(R.id.TFaddress);

//Se cambia el texto ingresado a una variable tipo String
        String location = location_tf.getText().toString();
        List<Address> addressList;
        if(!location.equals("")){

//Se crea el objeto que  permitirá obtener las coordenadas de la dirección ingresada
            Geocoder geocoder = new Geocoder(this);
            try {

//Las coordenadas obtenidas se guardan en el objeto adress
                addressList = geocoder.getFromLocationName(location, 1);
                Address address = addressList.get(0);

//Se transfieren las coordenadas al objeto latLng
       LatLnglatLng = new LatLng(address.getLatitude(),address.getLongitude());

//Se crea el marcador con las coordenadas recibidas
 mMap.addMarker(newMarkerOptions().position(latLng).title(location));
```

```
//Se define el valor del acercamiento de la cámara
 mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));

            } catch (IOException e){
                e.printStackTrace();
            }
        }else {
            Toast toast=Toast.makeText(this,"Inserte locación",Toast.LENGTH_SHORT);
            toast.show();
        }

    }

    public void on_Trole(View view) {

//Se declaran las coordenadas de la zona de circulación de trolebuses
            Double latitude = 19.501654935817502;
            Double longitude = -99.13990433471832;

//Se crea la variable que contiene las coordenadas
            LatLng latLng = new LatLng(latitude, longitude);

//Se realiza un acercamiento en el mapa de acuerdo  a la variable con las coordenadas
            mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));

//Se define el valor del acercamiento de la cámara
            mMap.animateCamera(CameraUpdateFactory.newLatLng(latLng));
            mMap.animateCamera(CameraUpdateFactory.zoomTo(15));
        }

    @Override
    protected void onResume() {
        super.onResume();
        setUpMapIfNeeded();
    }

    private void setUpMapIfNeeded() {

    // Se realiza una condición para revisar que el mapa allá cargado
        if (mMap == null) {

            //Se obtiene el mapa de SuportFragment.            mMap =
((SupportMapFragment) getSupportFragmentManager().findFragmentById(R.id.map))
                    .getMap();

            // Se revisa que la obtención del mapa sea exitosa.
            if (mMap != null) {
                setUpMap();
            }
        }
    }

    private void setUpMap() {

//Se habilita la ubicación del usuario, la cual se muestra en el mapa junto con un
botón de retorno
        mMap.setMyLocationEnabled(true);
        if(trac!=null&&suck!=null){

            mMap.clear();
```

```
//Se realiza el cambio de tipo de variable, de String a Double
            Double latitude = Double.parseDouble(trac);
            Double longitude =Double.parseDouble(suck);

//Se crea el marcador con las coordenadas recibidas, para mostrar la ubicación del
trolebus.
            mMap.addMarker(new MarkerOptions().position(new LatLng(latitude,
longitude)).title("Ubicación Trolebús!"));

        }

    }

    private void trolezona(){

//Se declaran las coordenadas de la zona de circulación de trolebuses

        Double latitude= 19.501654935817502;
        Double longitude = -99.13990433471832;

//Se crea la variable que contiene las coordenadas
        LatLng latLng = new LatLng(latitude, longitude);

//Se realiza un acercamiento en el mapa de acuerdo  a la variable con las
coordenadas
        mMap.moveCamera(CameraUpdateFactory.newLatLng(latLng));

//Se define el valor del acercamiento de la cámara
        mMap.animateCamera(CameraUpdateFactory.zoomTo(15));
    }
////////////////////////////////////////UDP
Socket//////////////////////////////////////////////////////

    private class UDP extends AsyncTask<String, Void, String> {

        @Override
        protected String doInBackground(String... params) {
            int puerto = 9010;
            String linea;

            try {
                MulticastSocket socketCliente = new MulticastSocket(puerto);
                InetAddress Host = InetAddress.getByName("228.5.6.7");
                socketCliente.joinGroup(Host);

                while (true) {
                    byte[] bufer = new byte[1024];
                    DatagramPacket mensajeEntrada =
                            new DatagramPacket(bufer, bufer.length);
                    socketCliente.receive(mensajeEntrada);
                    linea = new String(mensajeEntrada.getData(),0,
mensajeEntrada.getLength());
                    traduc = linea;

                    StringTokenizer coo=new StringTokenizer(traduc,",");
                    String Lat=coo.nextToken();
                    String Lon=coo.nextToken();
                    trac=Lat;
                    suck=Lon;


                    socketCliente.close();
                    if (mensajeEntrada.equals("Adios")) break;
```

```
                }
                socketCliente.leaveGroup(Host);

            } catch (SocketException e) {
                e.printStackTrace();
            } catch (UnknownHostException e) {
                e.printStackTrace();
            } catch (IOException e) {
                e.printStackTrace();
            } catch (Exception e) {

            }
            return null;
        }
        @Override
        protected void onPostExecute(String result) {
        }

        @Override
        protected void onPreExecute() {
        }

        @Override
        protected void onProgressUpdate(Void... values) {
        }
    }
}
```

Código

Código del archivo MapsActivity.java el cual se encuentra en la ruta AndroidStudioProjects **\Aplicacion-Trolebus**\app\src\main\java**\org\tesis\esime\usuario** (Las partes remarcadas en negritas pueden variar dependiendo del nombre del proyecto que se haya generado).

Apéndice 5: Código fuente del archivo AndroidManifest.xml de la aplicación del Usuario

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.luis18.mtp" >

    <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" />

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <meta-data
            android:name="com.google.android.gms.version"
            android:value="@integer/google_play_services_version" />
        <meta-data
            android:name="com.google.android.maps.v2.API_KEY"
            android:value="@string/google_maps_key" />

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

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

</manifest>
```

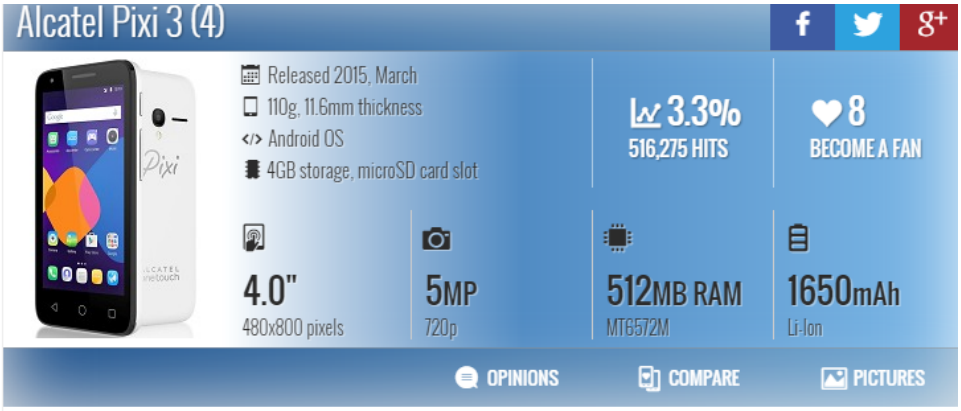
Código del archivo AdroidManifest.xml el cual se encuentra en la ruta AndroidStudioProjects **\Aplicacion-Trolebus**\app\src\main (Las partes remarcadas en negritas pueden variar dependiendo del nombre del proyecto que se haya generado).

Apéndice 6: Código fuente del archivo activity_maps.xml de la aplicación del Usuario

```
<RelativeLayout
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:id="@+id/lay1">


    <fragment xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:tools="http://schemas.android.com/tools" android:layout_width="400dp"
        android:layout_height="510dp" android:id="@+id/map"
tools:context=".MapsActivity"
        android:name="com.google.android.gms.maps.SupportMapFragment"
        android:layout_alignParentTop="true"
        android:layout_alignParentStart="true" />

    <EditText
        android:layout_width="184dp"
        android:layout_height="wrap_content"
        android:id="@+id/TFaddress"
        android:layout_alignTop="@+id/Bsearch"
        android:layout_centerHorizontal="true" />

        <Button
            style="?android:attr/buttonStyleSmall"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Buscar"
            android:id="@+id/Bsearch"
            android:onClick="onSearch"
            android:layout_alignParentBottom="true"
            android:layout_alignParentStart="true" />
        <Button
            style="?android:attr/buttonStyleSmall"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/Role"
            android:id="@+id/Btrole"
            android:layout_gravity="right"
            android:onClick="on_Trole"
            android:layout_alignParentBottom="true"
            android:layout_alignParentEnd="true" />

</RelativeLayout>
```

Código del archivo activity_maps.xml el cual se encuentra en la ruta AndroidStudioProjects **\Aplicacion-Trolebus**\app\src\main\res\layout (Las partes remarcadas en negritas pueden variar dependiendo del nombre del proyecto que se haya generado).

Apéndice 7: Hoja de especificaciones del dispositivo móvil Alcatel Pixi 3(4)



NETWORK	Technology	GSM / HSPA / LTE EXPAND
LAUNCH	Announced	2015, January
	Status	Available. Released 2015, March
BODY	Dimensions	121.3 x 64.2 x 11.6 mm (4.78 x 2.53 x 0.46 in)
	Weight	110 g (3.88 oz)
	SIM	Optional Dual SIM Mini Sim (3G model), Micro SIM (4G model)
DISPLAY	Type	TFT capacitive touchscreen, 16M colors
	Size	4.0 inches (~58.5% screen-to-body ratio)
	Resolution	480 x 800 pixels (~233 ppi pixel density)
	Multitouch	Yes, up to 2 fingers
PLATFORM	OS	Android OS, v4.4.2 (KitKat)/ Planned upgrade to v5.0 (Lollipop) - EMEA model
	Chipset	Mediatek MT6572M (3G model) Qualcomm MSM8909 (4G model)
	CPU	Dual-core 1 GHz Cortex-A7 (3G model) Quad-core 1.1 GHz (4G model)
MEMORY	Card slot	microSD, up to 32 GB
	Internal	4 GB, 512 MB RAM
CAMERA	Primary	2 MP / 5 MP, LED flash
	Features	Geo-tagging, panorana. HDR
	Video	720p@30fps
	Secondary	VGA
SOUND	Alert types	Vibration; MP3, WAV ringtones
	Loudspeaker	Yes
	3.5mm jack	Yes
COMMS	WLAN	Wi-Fi 802.11 b/g/n, Wi-Fi Direct, hotspot
	Bluetooth	v4.0, A2DP
	GPS	Yes, with A-GPS
	Radio	FM radio
	USB	microUSB v2.0
FEATURES	Sensors	Accelerometer
	Messaging	SMS(threaded view), MMS, Email, IM
	Browser	HTML
	Java	No
		- MP3/AAC+/WAV player - MP4/H.264 player - Document viewer - Photo editor
BATTERY		Li-Ion 1400 mAh battery (EMEA)
	Stand-by	Up to 466 h (2G) / Up to 350 h (3G)
	Talk time	Up to 8 h 20 min (2G) / Up to 7 h 40 min (3G)
		Li-Ion 1650 mAh battery (4G model)
	Stand-by	
	Talk time	

Apéndice 8: Hoja de especificaciones del dispositivo Arduino uno kit

Technical specs

Microcontroller	ATmega328P
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	14 (of which 6 provide PWM output)
PWM Digital I/O Pins	6
Analog Input Pins	6
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	32 KB (ATmega328P) of which 0.5 KB used by bootloader
SRAM	2 KB (ATmega328P)
EEPROM	1 KB (ATmega328P)
Clock Speed	16 MHz
Length	68.6 mm
Width	53.4 mm
Weight	25 g