UACM

Universidad Autónoma de la Ciudad de México

Nada humano me es ajeno

COLEGIO DE CIENCIA Y TECNOLOGÍA

LICENCIATURA EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

“Asistencia Social en Sistema Operativo Android para el Sistema de Transporte Metrobús de la Ciudad de México”

TRABAJO RECEPCIONAL

PARA OBTENER EL TÍTULO DE LICENCIADO EN INGENIERÍA EN SISTEMAS ELECTRÓNICOS Y DE TELECOMUNICACIONES

PRESENTA:

Israel Alba Mejía

Director del trabajo recepcional

Ing. David Velázquez Suárez

México, D.F. diciembre, 2015.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

CA Coordinación Académica

SISTEMA BIBLIOTECARIO DE INFORMACIÓN Y DOCUMENTACIÓN

UNIVERSIDAD AUTÓNOMA DE LA CIUDAD DE MÉXICO
COORDINACIÓN ACADÉMICA

RESTRICCIONES DE USO PARA LAS TESIS DIGITALES

DERECHOS RESERVADOS©

Agradecimientos

Primero quiero agradecer a mi director de tesis, el ing. David Velázquez Suárez, quien fue un profesor que se preocupó porque adquiriéramos el método científico y el conocimiento práctico que requerimos todos los estudiantes.

Segundo, quiero agradecer a mi madre la Sra. Patricia Mejía García, quien me apoyo dentro de sus limitaciones a que pudiera terminar la carrera.

Tercero, quiero agradecer al Dr. Leonardo Álvarez Paque, quien no sólo ha sido un excelente amigo para mí, sino que también es uno de los mejores médicos que he conocido en mi vida y gracias a él y a su diagnóstico oportuno es que pude continuar mis estudios y superarme en la vida.

Y a todos los maestros que han sido mis guías a lo largo de mi vida profesional.

Síntesis

En el presente documento se describe una aplicación de asistencia social para los usuarios del sistema de transporte metrobús usando el SO[1] Android.

El objetivo principal de esta aplicación es ofrecer al usuario del sistema de transporte público metrobús, información relacionada con la cantidad de usuarios que hay en una estación, de esta forma se le permitirá al usuario tomar decisiones en sus traslados en el uso del sistema de transporte metrobús de la Ciudad de México, para optimizar el tiempo de traslado de los usuarios. La aplicación debe ser fácil de usar y de bajo costo al momento de usar la red de telefonía celular, en caso que la aplicación sea ejecutada desde un punto de acceso de red celular.

Un aspecto muy importante que debe cumplir la aplicación es la rapidez de ejecución, es decir, el usuario no debe de tardar para obtener la información de una estación en particular. Esto se debe a que los usuarios requieren la información lo más rápido posible para transportarse. La aplicación fue desarrollada tomando en cuenta a los usuarios que a diario utilizan el sistema de transporte metrobús y que conocen no sólo este medio de transporte público. Para el desarrollo de la aplicación fue necesario

[1] SO (Sistema Operativo).

un dominio de Internet y un servidor externo. Esto se debe a que los usuarios deben acceder a una base de datos independientemente de la red en que se encuentren para poder intercambiar información con otros usuarios. Así los usuarios se conectarán a una base de datos en donde se encuentran los registros de aquellos pasajeros que se encuentran en la estación y podrán ser contabilizados para saber la cantidad de usuarios que hay en una determinada estación.

La aplicación detecta la cercanía de un usuario con alguna estación mediante el receptor de *GPS* integrado en el dispositivo móvil, mediante una serie de cálculos y conexiones a un servidor externo se determina la cantidad de usuarios que están ejecutando la aplicación que se encuentran en una estación del metrobús para que la información pueda ser guardada en un servidor externo y posteriormente pueda ser consultada por algún usuario.

La aplicación comienza realizando una conexión a un servidor externo, en el servidor se encuentran alojados los archivos *PHP* y las bases de datos a las que la aplicación accede cada vez que el usuario selecciona una estación, el servidor envía al dispositivo Android (cliente) el resultado de la petición realizada por el usuario y finalmente, el usuario recibe en la pantalla de su dispositivo móvil un mensaje que contiene el resultado de la petición.

El cliente es un dispositivo móvil, ya sea un “*Smartphone*” (*teléfonos inteligentes*), Tablet o cualquier dispositivo que cuente con los requerimientos mínimos para ejecutar la aplicación. Posteriormente se abordarán los requerimientos mínimos que deben de cumplir los dispositivos móviles para hacer uso de la aplicación. El servidor externo entrega toda la información necesaria con la que debe de contar el cliente (dispositivo Android) para que se puedan realizar los cálculos y conexiones a la base de datos.

El dispositivo móvil recibe del servidor externo los datos (*Estacion*[2] y *línea*) de la tabla *Estaciones* en una codificación de tipo *JSON* (JavaScript Object Notation), el cual es visualizado por el usuario para que pueda seleccionar desde la pantalla del dispositivo móvil la estación de su interés. Una vez que el usuario hace su selección, el dispositivo móvil proporciona al servidor externo las variables longitud y latitud obtenidas por el *GPS* del dispositivo Android. Cuando el servidor recibe las variables longitud, latitud, estación y línea obtenidas por el dispositivo móvil, realiza los cálculos para determinar cuál es la línea que se encuentra cerca del usuario, y determinar en qué estación se encuentra el usuario. Por otro lado si el usuario se encuentra en una distancia mínima previamente establecida dentro del servidor, se agregan dentro de una base de datos las variables que identificarán a cada usuario. El dispositivo Android continúa enviando registros de longitud y latitud cada minuto hasta que llegue a un máximo de 200 registros o se cierre la aplicación, con la finalidad de que el usuario sea contabilizado dentro de la base de datos puesto que aún se encuentra dentro del sistema de transporte metrobús.

Cuando el usuario realiza una selección de estación y línea, automáticamente se realiza un *Query*[3] dentro del servidor incluyendo las variables *Estación, línea, fecha* y *hora*. Estas dos últimas variables son definidas por el servidor cuando el cliente o dispositivo Android realiza una petición, esto da la seguridad de que no se consulten desde el dispositivo móvil registros que tengan una antigüedad mayor a un minuto. El servidor envía al cliente una suma de todos los registros que coincidan con su petición, así el dispositivo móvil muestra en pantalla la cantidad de usuarios que se encuentran en ese momento en una estación.

[2] Para evitar errores de compilación las variables no deben de incluir caracteres especiales como acentos, tal es el caso de las variables "*Estación"* y "*línea", entre otras variables*.
[3] Palabra de origen inglés que en lenguaje de programación hace referencia a una consulta.

Resumen

Capítulo 1. Introducción

En este capítulo se hace una introducción al problema de transporte público en la Ciudad de México, haciendo un breve antecedente del mismo se justifica el propósito del presente trabajo, objetivos generales y particulares del proyecto.

Capítulo 2. Desarrollo teórico

En este capítulo se presenta el marco teórico general, la hipótesis, las variables del proyecto y la metodología para llegar a la solución propuesta.

Capítulo 3. Desarrollo práctico

En este capítulo se explica la creación del dominio, permisos, bases de datos, implementación de la aplicación, el cálculo de distancias y la obtención de peticiones para el usuario.

Capítulo 4. Presentación de datos

En este capítulo se presentan los resultados que se obtuvieron del servidor, resultados recibidos por el usuario, interpretación de los mismos e interfaz del usuario.

Capítulo 5. Conclusiones, trabajos futuros, a*lcances y metas.*

Referencias.

Contenido

Capítulo 1

1.1 Introducción

En la actualidad la Ciudad de México presenta un gran problema de transporte, las causas del problema son múltiples, las soluciones buscadas y algunas de ellas implementadas, resultan paliativas y no tienden a resolverlo. Si bien hacen falta más sistemas de transporte público también es necesario hacer uso eficiente de las opciones actuales con las que ya cuenta la Ciudad de México.

En muchas ciudades del mundo, principalmente europeas, han sustituido el uso de vehículos particulares por sistemas de transporte público, e incluso han implementado transporte alternativo como lo es el uso de la bicicleta para aligerar la carga vehicular.

Dentro de los efectos que este problema causa en la sociedad, es el daño de la calidad de vida, por los tiempos de traslado, que incrementa con la creciente afluencia vehicular en las llamadas “horas pico”.

Si el usuario del transporte público metrobús pudiera saber la cantidad promedio de personas que se encuentran en una estación antes de llegar a ella, el usuario podría determinar si le es conveniente llegar a dicha estación, utilizar otro sistema de transporte u otras formas de trasladarse. De tal forma al darle al

usuario la opción de saber el “estado” de una estación se podrían reducir tiempos de traslado y al mismo tiempo podría lograrse reducir la cantidad de usuarios que se encuentran en las estaciones, e incluso podrían evitarse gastos económicos superfluos, porque muchos pasajeros al no poder ingresar a las estaciones debido a que los autobuses están saturados, llegan a abandonar las instalaciones al no poder acceder al autobús y esto representa además de pérdidas económicas para los usuarios pérdidas de tiempo.

Por otro lado, en la actualidad, el uso casi generalizado de un dispositivo móvil con acceso a Internet, da motivo al desarrollo de la presente aplicación. El presente proyecto pretende hacer uso de los medios de transporte ya disponibles para los usuarios y darles opciones para que se beneficien del sistema de transporte metrobús. Esto se pretende llevar a cabo, creando una aplicación que permita a los usuarios saber la cantidad aproximada de pasajeros que se encuentran en determinada estación a determinada hora. De esta forma el usuario tendrá la libertad de decidir si le es conveniente esa ruta, línea, transporte o incluso podría determinar el tiempo de traslado que le llevaría al hacer uso del sistema metrobús, tomando en cuenta que el *GPS* puede calcular velocidades y también es posible calcular la distancia que hay entre un usuario y cada una de las estaciones, fácilmente se puede obtener la relación de tiempo estimado por medio de los parámetros velocidad y distancia.

Aunque la aplicación se enfoca al sistema de transporte metrobús de la Ciudad de México, la aplicación puede ser implementada para otros sistemas de transporte público, porque contando con los sistemas de comunicación actuales es posible realizar implementaciones de este tipo y nuevas aplicaciones para distintos sistemas de transporte.

Esta aplicación desarrollada en sistema operativo (SO) Android, crea una conexión de datos entre un cliente (dispositivo Android) y un servidor, se registra en una base de datos la cantidad de

usuarios que hay en una estación, para que dicha información posteriormente pueda ser consultada desde un dispositivo Android por algún usuario.

La aplicación debe de cumplir con las siguientes características, para que sea agradable y útil al usuario del dispositivo móvil.

- La aplicación debe de ejecutarse en el menor tiempo posible para que los usuarios obtengan su información rápido.
- La aplicación debe ocupar poco espacio en memoria del dispositivo Android.
- Usar lo menos posible recursos de redes móviles, esto se debe al cobro que hacen las compañías proveedoras del servicio de telefonía celular.
- La aplicación debe estar disponible para el mayor número de usuarios, por eso se eligió a Android como el SO, en el cual se ejecutará la aplicación.
- La aplicación debe ser sostenible (que pueda funcionar aun cuando haya cambios en el sistema metrobús) y expandirse al aparecer nuevas líneas.
- Interfaz gráfica simple e intuitiva.

Por lo anterior se decidió que la aplicación debe realizar los cálculos que resulten necesarios remotamente (en el servidor), esto no sólo ayudará al mejor desempeño de la aplicación sino que también reduce el tiempo que tarda la aplicación para ejecutarse. De igual forma algo que hay que tomar en cuenta es que, para que la aplicación pueda realizar los cálculos localmente es necesario contar con más información de la base de datos, al tener que descargar más información del servidor el usuario deberá pagar más por su acceso a Internet.

1.2 Usuarios

La aplicación fue desarrollada para funcionar con versiones de Android 2.0 a 6.0 por lo cual cualquier persona que cuente con algún dispositivo móvil Android podrá utilizarla. Hay que recordar que se debe tener acceso a una red de datos ya sea mediante redes celulares o Wi-Fi. La aplicación se decidió que fuese desarrollada en SO Android porque en la actualidad es el sistema operativo más utilizado en móviles, además que los dispositivos Android suelen ser más económicos a comparación de su competidor más cercano iOS.

Con el creciente desarrollo de la Ciudad de México han surgido nuevas metas, una de ellas es el desarrollo de sistemas de transporte público que cubran las necesidades que demanda esta gran urbe, si recordamos que la ciudad de México tiene una cantidad de población de 8,851,080. de habitantes[4], sin contar con la cantidad de personas que todos los días llegan de otros estados a laborar a la Ciudad de México, la ciudad se enfrenta a un gran problema de transporte y tráfico vehicular. En la actualidad es muy común que las personas utilicen algún sistema de comunicación móvil, los cuales ya cuentan con distintos sensores electrónicos que dan mayores servicios a los usuarios. Haciendo uso de la tecnología con la que ya cuentan los dispositivos móviles, es posible explotar el potencial de uso y aplicaciones de los llamados Smartphones.

1.3 Hipótesis

La mejor forma de realizar la implementación de la aplicación es que la aplicación se conecte a un servidor con una *IP (Internet Protocol)* pública y que todos los cálculos se lleven a cabo en el mismo

[4] Dato obtenido en la página oficial del INEGI, del censo de población y vivienda 2010.

servidor. El servidor debe de contener una base de datos con la información de las estaciones y la posición geográfica de cada una de las estaciones.

La posición actual del usuario puede ser obtenida mediante el sistema interno de *GPS* con el que cuenta cada dispositivo móvil, de esta manera se podrá determinar la distancia que hay entre un dispositivo móvil y una estación. Aunado a esto, es necesario introducir a la base de datos la información proporcionada por cada usuario, para contar el número de usuarios en cada estación.

1.4 Justificación

El desarrollo de esta aplicación en Android será útil para dar alternativas a los problemas de transporte que presenta en la actualidad la Ciudad de México, además, la base de datos obtenida podría arrojar un histórico de datos que se pueden aprovechar para realizar mejoras en el sistema, por ejemplo, saber con exactitud en qué estaciones a determinadas horas e incluso días, presentan mayor afluencia de usuarios; de esta forma el sistema de transporte metrobús podría implementar medidas para evitar retrasos o saturación en las estaciones. La información dentro de la base de datos se conservará hasta que se borre directamente en el servidor.

El usuario podrá determinar la mejor opción de transporte al saber el estatus de cada estación antes de ingresar al sistema. Con estas medidas no sólo el usuario estaría ahorrando tiempo, también dinero cuando un pasajero tiene que abandonar la estación debido a que hay un exceso de pasajeros, lo cual implica que tiene que volver a pagar para poder ingresar al sistema de transporte metrobús.

Es importante mencionar que actualmente existen dos aplicaciones en el mercado, disponibles en ***play store***[5] para el sistema de transporte metrobús llamadas: "*Metro y Metrobús México*" y "*Metro y*

[5] Tienda de descarga de aplicaciones y contenido de Android.

Metrobús de México"[6], estas aplicaciones existentes, no son fáciles de usar y sólo trazan rutas o indican la cercanía con alguna estación; no dan información del estado de la estación en concreto. Parecen estar orientadas a personas que usan el transporte por primera vez y no ayudan al usuario a tomar la mejor decisión al momento de tomar una ruta en particular.

1.5 Objetivos

1.5.1 Objetivo General

Diseñar e implementar una aplicación móvil que ayude a determinar la densidad de usuarios, densidad en ruta, tiempos, distancia a estaciones, entre otros, ayudar al usuario a que pueda decidir la viabilidad de abordar determinada estación o línea.

1.5.2 Objetivos

Los objetivos del proyecto son los siguientes:

- Generar una base de datos que contenga la información de las estaciones y que pueda ser consultada remotamente.
- Creación de un dominio de Internet para que se pueda acceder a la base de datos remotamente.
- Generar un entorno de conexión local y externo para que la aplicación acceda a la información de la base de datos y al mismo tiempo la aplicación pueda ingresar información a la base de datos.
- Realizar los cálculos necesarios ya sea local o remotamente para determinar la cercanía del usuario con una estación.

[6] Información obtenida de la play store en día 25 de Junio de 2014.

- Obtención de resultados en tiempo real y de forma rápida.
- Reducir al máximo los costos (económicamente hablando) que se puedan generar al usuario al ejecutar la aplicación.

1.6 Características de la aplicación.

Entre las ventajas con las que cuenta la aplicación están las siguientes:

- **Fácil uso**. El usuario sólo debe abrir la aplicación y seleccionar la estación de la cual quiere saber su estado, de tal forma que usuarios poco expertos en dispositivos móviles podrán usar la aplicación fácilmente.
- **Disponible para cualquier versión de Android** (a partir de la 2.0 *Eclair*).
- **Aplicación gratuita** No incluye el acceso a Internet, los costos pueden variar dependiendo del proveedor deservicio de telefonía celular.
- **Puede ser ejecutada desde cualquier red** incluso Wi-Fi por lo cual el usuario podrá saber el estatus de una estación antes de llegar a ella.
- **Aplicación sostenible**, quiere decir que la aplicación está diseñada para poder ser implementada fácilmente en caso de que surjan nuevas estaciones, cambien de nombre las estaciones, etc., sin necesidad de liberar otra versión de la aplicación.
- **Aplicación única en el mercado**, ya que en la actualidad no hay una aplicación que calcule la cantidad de usuarios que se encuentran en alguna estación.
- **Base de datos en *MySQL***, de esta forma se podrá obtener información importante de los usuarios y a su vez será posible realizar cálculos estadísticos que pueden ayudar a la toma de decisiones y nuevas implementaciones al sistema de transporte metrobús.
- **Totalmente desarrollada en "*Código Abierto"*** por lo cual cualquier persona que quiera

realizar alguna actualización podrá hacerlo.

1.7 Ventajas de un servidor externo

La aplicación demuestra un mejor desempeño funcionando con un servidor externo, las ventajas del uso de este servidor son:

1. **Ahorros económicos**, esto se debe en gran medida a que sólo requiere de la mitad de registros de la base de datos, porque sólo se descargan del servidor el listado de estaciones y no se requiere del listado de posiciones geográficas de las estaciones para realizar los cálculos necesarios para determinar distancias y debido a que se requiere descargar menos información del servidor externo, el usuario no tendrá que pagar más por su acceso a Internet.
2. **Mayor control de la base de datos**, representa una gran ventaja, porque no requiere desarrollar una nueva versión de la aplicación cada vez que se requiera actualizar la misma.
3. **La velocidad de respuesta y procesamiento de los datos es mucho mayor**, aunque el dispositivo tiene la capacidad de realizar los cálculos de distancias entre estaciones y usuarios, el dispositivo móvil tarda más tiempo en obtener los resultados y consume más rápidamente su batería.

1.8 Recursos

Para la realización de la aplicación se utilizaron los siguientes materiales y software:

- Computadora con 4 GB de RAM y 300 GB en disco duro.
- Android SDK.
- Java JDK versión 1.8.0.
- Eclipse IDE Mars versión 4.5.0.

- Un dominio de Internet.
- Un servidor externo, que cuenta con servicios de *MySQL* y *PHP*.
- Dispositivo Android con acceso a Internet y sensor de *GPS*.
- Photoshop CS6.

Capítulo 2

2.1 Marco teórico

En este capítulo se abordan los tópicos importantes para el mejor entendimiento del desarrollo de la aplicación.

2.1.1 GPS

El "*Global Positioning System*" o *GPS* consiste en un conjunto de 24 satélites que circundan la tierra y envían señales de radio frecuencia a su superficie. Un receptor de *GPS* es un dispositivo electrónico pequeño que permite a quienes los utilizan recibir las señales de los satélites. El receptor utiliza señales de radio frecuencia para calcular su posición.

El principio utilizado por el *GPS* para determinar la posición se basa en la medición de la distancia entre el "receptor" *GPS* (en este caso el Smartphone) y varios satélites. La posición de cada satélite en el espacio permanece conocida con extrema precisión; cada satélite transmite permanentemente su posición exacta con respecto a la tierra. También indica la hora exacta de la transmisión del mensaje. Calculando el tiempo requerido por las señales para llegar al receptor, se establece la distancia del receptor al satélite. Con la distancia y la posición del satélite, puede trazarse un círculo imaginario en la superficie de la tierra dentro de la cual hay un receptor, la intersección de varios círculos permite conocer la posición exacta del receptor, la

exactitud depende directamente con la cantidad de satélites que estén en línea de vista, sino hay la cantidad necesaria de satélites se pueden tener errores de posicionamiento de hasta 10 metros. En la Figura 1 se muestra el principio de posicionamiento *GPS*.

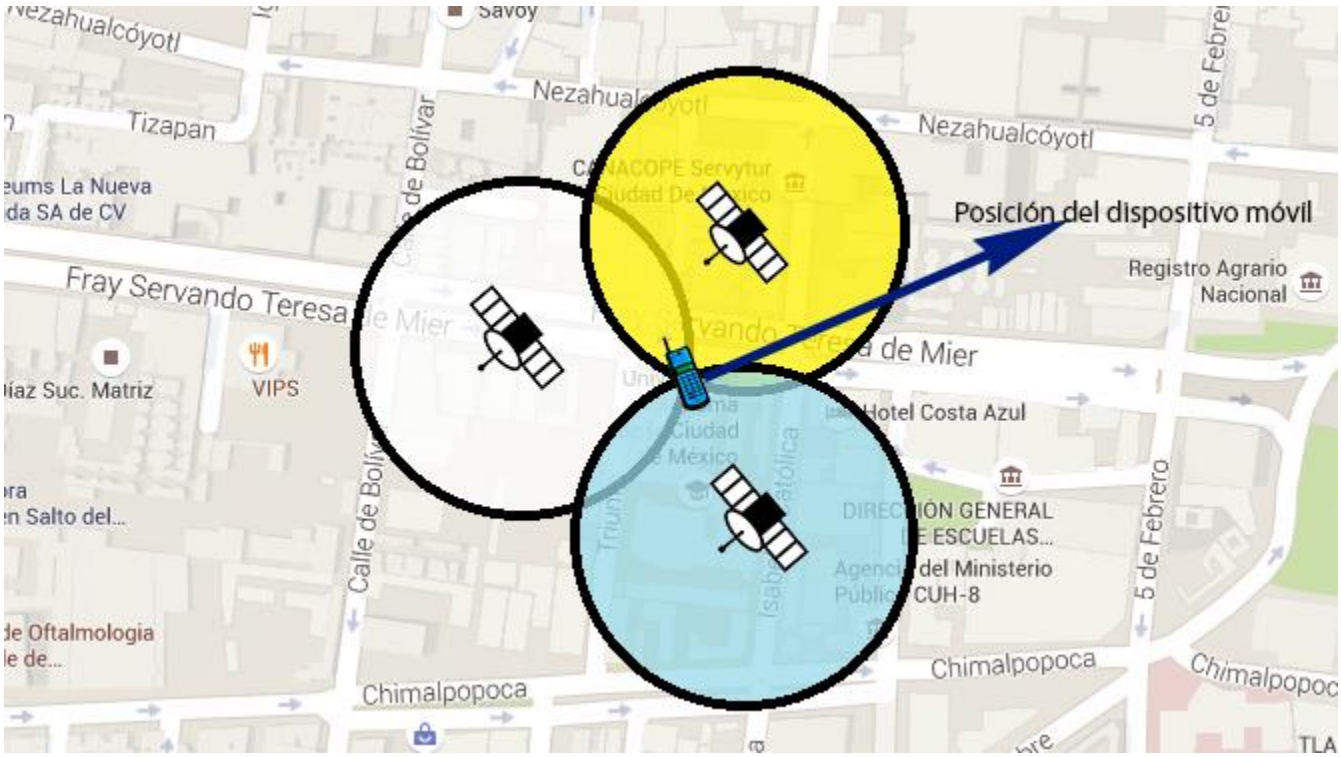


Figura 1: Método de posicionamiento GPS.

2.1.2 Principio de la posición por satélite

El Sistema de Posicionamiento Global está basado en tecnologías satelitales. La técnica del *GPS* es medir el alcance entre el receptor y algunos satélites “visibles”. La posición de los satélites es prevista y transmitida a lo largo de la señal *GPS* del usuario. A través de varias posiciones conocidas (de los satélites) y la distancia entre el receptor y el satélite, la posición del receptor puede ser determinada. La más importante aplicación del *GPS* es la navegación y el posicionamiento. El *GPS* ha sido

utilizado en muchas áreas tanto en tierra, mar y aire, estática y posición cinemática, monitoreo de vuelos y agrimensura[7].

El sistema de posicionamiento global *GPS* fue diseñado y construido por el departamento de defensa de los Estados Unidos, el primer satélite *GPS* puesto en órbita fue en 1978 y el sistema fue completamente terminado a mediados de los años 90. La constelación de satélites de *GPS* consiste de 24 satélites en seis orbitas planas con cuatro satélites en cada plano. A esta constelación de satélites se le llama *NAVSTAR* (Navigation Satellite Timing and Ranging) [1].

Los satélites de *GPS* son monitoreados por cinco estaciones base en Colorado Springs y otros cuatro localizados en la isla Ascensión (Océano Atlántico), Diego García (Océano Indico), Kwajalein y Hawái (Océano Pacífico).

Cada satélite de *GPS* transmite datos en tres frecuencias L1 a 1575.42 MHz, L2 a 1227.2 MHz y L5 a 1176.45 MHz, las portadoras de las frecuencias L1, L2 y L5 son generadas multiplicando la frecuencia fundamental por 152, 120 y 115, respectivamente, por códigos de ruido pseudoaleatorio (PRN).

La medición de la distancia que separa a un satélite del receptor se basa en la propagación de las ondas electromagnéticas. El tiempo empleado por una señal para llegar al receptor es directamente proporcional a la distancia recorrida. Las señales se propagan a una velocidad del orden de 300,000. (Km/seg). El tiempo que tarda en llegar la señal de un satélite oscila entre los 67 milisegundos.

[7] La palabra *agrimensura* tiene como raíz dos palabras latinas *ager,* que significa "campo" y *mensura,* que significa "medida". De tal forma que Agrimensura se define como la ciencia que se ocupa de la determinación o medida de las superficies de los terrenos.

2.1.3 Cobertura de los satélites.

Los satélites no siempre tienen la cobertura prevista, esto se debe en gran medida a variaciones en los campos magnéticos.

La disposición de los satélites hace posible tener, en el 99.9% de los casos, un mínimo de 4 satélites visibles a 5 o más por encima del horizonte con un PDOP (Disposición de precisión) igual o inferior a 6, en cualquier lugar sobre la superficie terrestre, esto significa que no siempre el *GPS* puede indicar una posición lo suficientemente confiable, esto se puede deber en casos en los que la zona no está lo suficientemente despejada, es decir, zonas en las que no puede verse desde el cielo. Estos lugares son raros, la cobertura no es segura en más del 3% de los casos, que indica un promedio de 45 minutos por día [2].

En la Figura 2 se muestra la cantidad de satélites visibles a lo largo de un día.

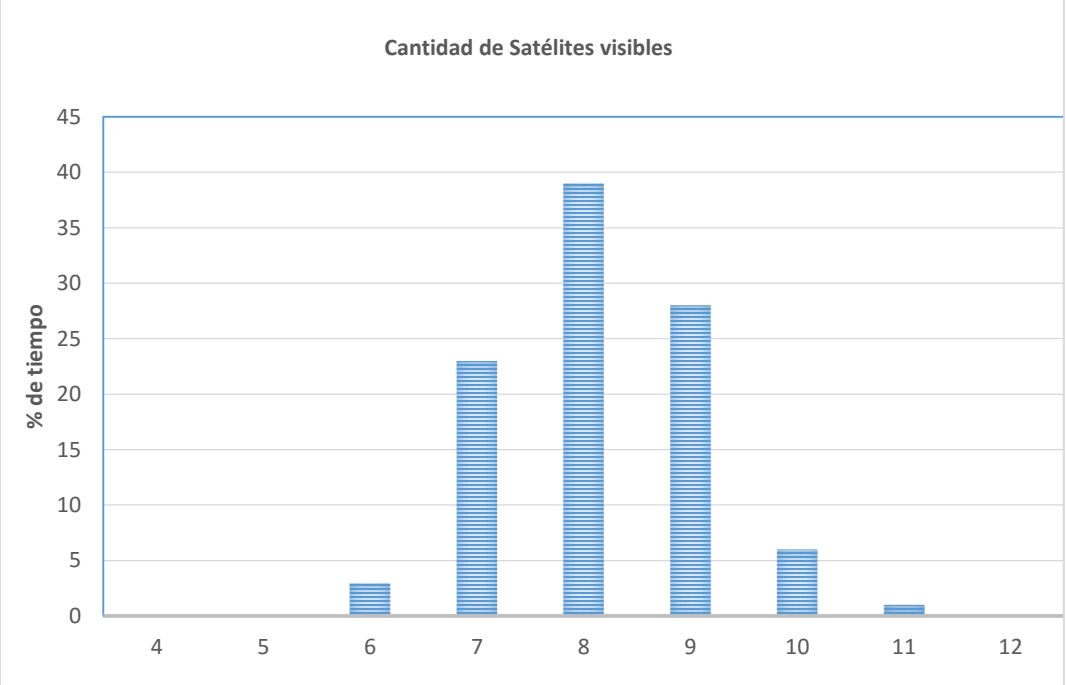


Figura 2: Cantidad de satélites visibles a lo largo de un día en promedio son 8 [3].

2.1.4 Control del GPS

El *GPS* es totalmente realizado, financiado y controlado por el departamento de defensa de los Estados Unidos de Norte América (DoD), El departamento de defensa es el único que decide en las implementaciones del *GPS* y el único que controla el funcionamiento del sistema.

Las señales *GPS* pueden ser utilizadas libremente por todo el mundo, con fines prácticos, de igual forma los receptores de *GPS* pueden ser fabricados y vendidos libremente en todo el mundo salvo en casos que el receptor de *GPS* funcione a una altura superior de 18,300. (m) y a una velocidad superior de 1,670. (km/h).

2.1.5 Sistema de satélites de navegación global (GNSS)

(GNSS) *Global Navigation Satellite System* es un sistema o tecnología, con cobertura global que permite a un teléfono hacer mediciones de señales que han sido transmitidas desde satélites para calcular con precisión su propia localización.

Los satélites trasmiten señales que pueden ser detectadas por un receptor y permiten al receptor determinar el alcance de cada uno de los satélites. El satélite también puede transmitir información que permite que el receptor pueda calcular la localización del mismo, la cantidad de efemérides de mediciones y obtener alguna información acerca de otros satélites en la constelación, puede ser calculado usando otras tecnologías de localización o puede venir desde una localización previamente almacenada dentro del teléfono.

Los datos de asistencia ayudan a que el teléfono bloquee señales de satélite rápidamente de varias maneras. Esto anulará la necesidad de que el teléfono demodule y module la transmisión del modelo

de transmisión; representa una ventaja significativa porque toma 30 segundos decodificar el historial de cada satélite o más aun cuando hay bits de errores de datos o señales con poca potencia.

La principal limitante del *GPS* autónomo es la cantidad de tiempo que toma en descargar el mensaje de navegación aunado a que el teléfono está restringido a usar señales lo suficientemente potentes para permitir que el mensaje de navegación pueda ser modulado. Esto puede ocasionar problemas en ambientes en donde las señales tienen poca potencia o en situaciones donde se requiere una localización rápida. Desde el servidor se provee la información de satélites que están en línea de vista desde la localización aproximada.

2.1.6 Modelo de asistencia.

El LS (Location Server) es el componente en una red inalámbrica que recupera información de referencia del *GPS* de un *GRS (GNSS Reference Server)* y provee datos de asistencia para un teléfono con *GPS*. En la figura 2 se muestra la arquitectura de *A-GPS (GPS Asistido).* El servidor de referencia *GNSS* (GRS) se conecta a una red dispersa de receptores *GNSS* que se extienden ya sea a través de la zona de cobertura de la red inalámbrica o geográficamente sobre la tierra de cobertura global. En la Figura 3 se muestra el modelo de asistencia *GPS*.

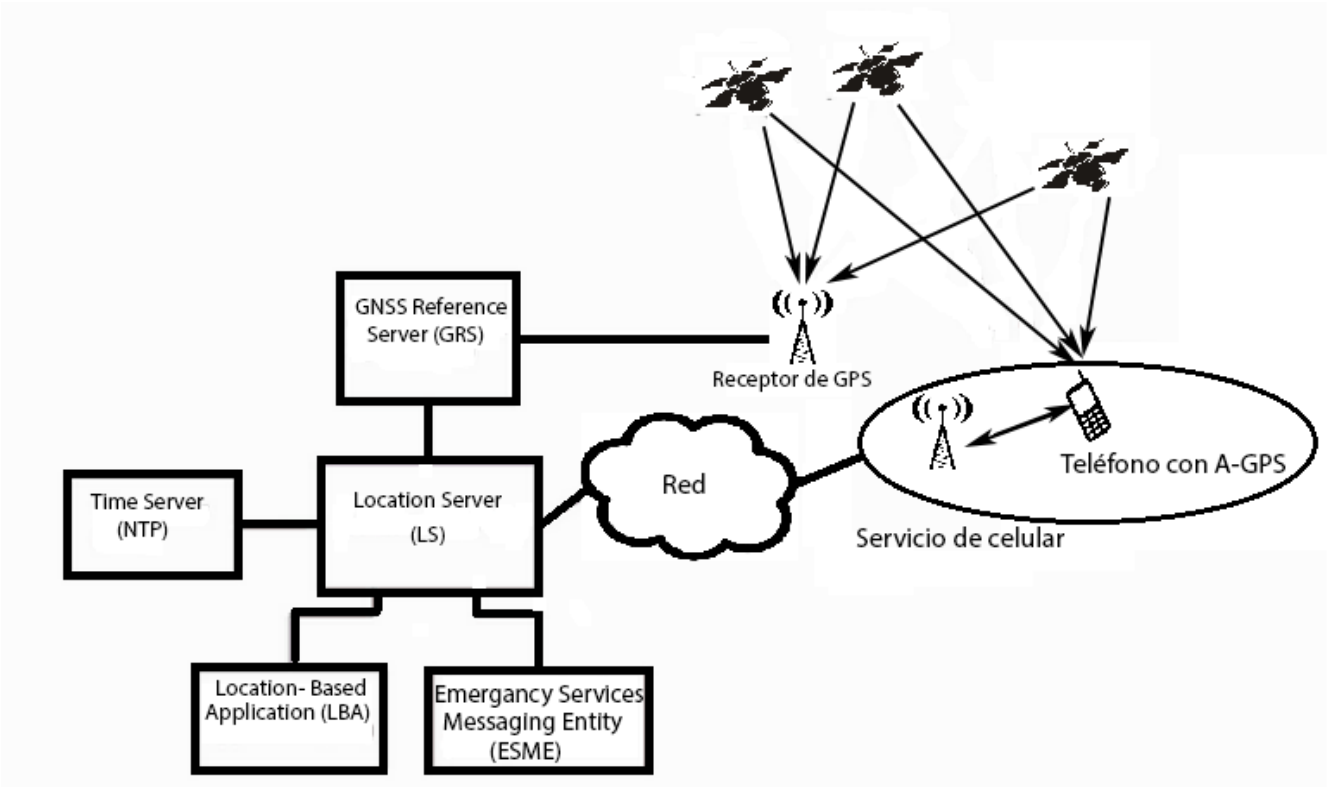


Figura 3: Modelo de asistencia [3].

2.1.6 Arquitectura de GPS Asistido (A-GPS)

La tarea básica del servidor *GNSS* es proveer datos a un dispositivo móvil *GNSS* habilitado en orden para mejorar el *TTFF* (*Time to First Fix*) y el rendimiento, a menudo provee la funcionalidad del cálculo de la posición en orden de conservar la energía de la batería del teléfono, reduce la cantidad de datos de asistencia que necesitan ser enviados en la red y permite la integración de otras mediciones disponibles en la red [3] .

El *A-GPS* es para mejorar el *TTFF* y el rendimiento comparado con el *GPS* autónomo para remover la necesidad de demodular el mensaje de navegación de la señal transmitida. Esto significa que el

teléfono no necesita esperar por el mensaje de navegación y puede ser usado con señales con menos potencia. Además que el servidor del *A-GPS* puede realizar cálculos y significa que el teléfono tiene menos que hacer y puede conservar su batería, también los enfoques de apoyo de datos se centran en el teléfono y puede ver a los satélites mucho más rápido.

2.2 SO Android

2.2.1 Orígenes de Android

Android es un SO libre que permite tener todos los dispositivos móviles funcionando como computadoras y más aún, los conecta a una red. Fue desarrollado en sus inicios por Google. Android es una versión del sistema operativo libre Linux pero adaptado para una computadora que cuenta con pocos botones (a diferencia de las computadoras que tienen un teclado alfa-numérico) y pantallas táctiles.

En la actualidad el SO Android tiene nuevas funciones que le permiten el reconocimiento de voz y es capaz de realizar búsquedas en la red con tan sólo decir lo que se desea buscar y obtener búsquedas de lugares cercanos mediante el *GPS*.

Google adquirió Android en el año 2005 que en un principio se trataba de una pequeña compañía encargada de crear aplicaciones móviles. Ese mismo año se comenzó a trabajar con la creación de la máquina virtual Java optimizada para móviles (Dalvik VM).

Para el año 2007 se creó el consorcio Handset Allianse 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. Su objetivo principal es promover el diseño y la difusión de la plataforma Android.

En diciembre de 2007 se lanzó la primera versión de Android ***SDK***, para el siguiente año apareció el primer móvil con SO Android. (T-Mobile G1). En octubre del 2008 Google liberó el código fuente de Android principalmente bajo licencia de código abierto Apache. De igual forma ese mismo mes se abrió **Android Market**, para la descarga de aplicaciones. En abril del 2009 Google lanzó la versión del *SDK* que incorpora nuevas características como teclado en pantalla (para que los usuarios puedan introducir información directamente desde una pantalla táctil). A finales del 2009 se lanzó 2.0 y durante el 2010 las versiones 2.1, 2.2, y 2.3.

En el 2012 Google cambió la **Android Market** por **Play Strore**, aquí en un sólo portal se pueden descargar aplicaciones como contenidos (videos, música, libros, etc.). En ese mismo año aparecen las versiones de Android 4.1 y 4.2 del *SDK*. Es debido a este crecimiento que hasta este momento ocupa el 75% del mercado.

Existen varias plataformas móviles iPhone, Linux Mobile, BlackBerry, Palm, Java Mobile, pero Android tiene cualidades que lo hacen diferente, Android tiene las siguientes cualidades [4].

- **Plataforma realmente abierta.** Es una plataforma de desarrollo libre basada en Linux y de código abierto, una de sus grandes ventajas es que puede usar y personalizar el sistema sin pagar derechos de autor.
- **Adaptable a cualquier tipo de hardware**. No ha sido diseñado sólo para el uso de teléfonos y tabletas. Actualmente se pueden encontrar un sinnúmero de dispositivos como cámaras, electrodomésticos, etc. que utilizan el SO Android. Esto difiere con su competidor directo ya que en iOS es necesario desarrollar una aplicación distinta para iPhone y para iPad.
- **Arquitectura basada en componentes inspirados en Intel**. Esto se debe a que la interfaz de usuario se realiza en *XML* (*Extensible Markup Language*) lo cual permite que la misma aplicación se ejecute en un móvil de pantalla reducida o en una TV.

- **Portabilidad asegurada.** Las aplicaciones son desarrolladas en Java lo cual asegura que podrán ser ejecutadas en cualquier CPU. Se puede lograr mediante máquinas virtuales.
- **Gran cantidad de servicios incorporados.** *GPS*, bases de datos con *SQL*, reconocimiento y síntesis de voz, etc.
- **Optimizado para usar poca potencia y poca memoria**. Android utiliza una máquina virtual Dalvik. Es una implementación de Google de máquina virtual de Java optimizada para dispositivos móviles.

2.2.1 Requerimientos mínimos para la ejecución de Android SDK.

Android no requiere de dispositivos poderosos, de hecho puede ser ejecutado mediante emuladores. En la Tabla 1 se mencionan los requerimientos mínimos con los que debe contar el equipo de desarrollo.

	Windows	**Linux**	**Mac OS X**
Versión del SO	Windows XP (32 bits)	Ubuntu, RedHat y otros	OS X (10.4.9 +)
Espacio en Disco Duro	25 GB	25 GB	25 GB
Memoria RAM	3 GB	2 GB	4 GB
Procesador	Dual Core +	Dual Core +	sólo x86
USB	USB 2.0 +	USB 2.0 +	USB 2.0 +

Tabla 1: Requerimientos mínimos para utilizar Android SDK [5].

2.2.2 Java.

La plataforma de desarrollo de Android está construida en un entorno estándar de Java, las aplicaciones de Android son construidas con base a una plataforma de java, es por eso que es necesario instalar java para poder desarrollar en Android. Se debe de asegurar que antes de iniciar a programar en Android, se tenga instalado en *JDK*. El *JDK* tiene el compilador, depurador y otras

herramientas que se requieren en el desarrollo. Para ello es necesario descargar la versión más reciente del *JDK*[8].

2.2.3 SDK de Android.

El SDK[9] de Android es la colección de bibliotecas, herramientas y documentación que son requeridos para correr las aplicaciones de Android. No contiene todo el entorno de desarrollo, sólo contiene las herramientas básicas requeridas y el resto de las herramientas se deben descargar mediante el ***SDK Manager***. En la Figura 4 se muestra una imagen del Android SDK Manager.

2.2.4 Componentes del SDK de Android

El Android *SDK* tiene una estructura modular, lo cual significa que la mayor parte de los componentes del *SDK* están colocados en paquetes separados. Esto hace fácil instalar sólo los componentes que se van a utilizar en una aplicación. Los paquetes se instalan según la versión del SO a la que están orientadas. Es importante tomar esto en cuenta ya que cada plataforma requiere al menos de 100MB de espacio en disco duro.

[8] La página oficial para descargar el JDK es: http://www.oracle.com/technetwork/es/java/javase/downloads/index.html
[9] La página oficial para descargar el Android SDK es: https://developer.android.com/sdk/index.html

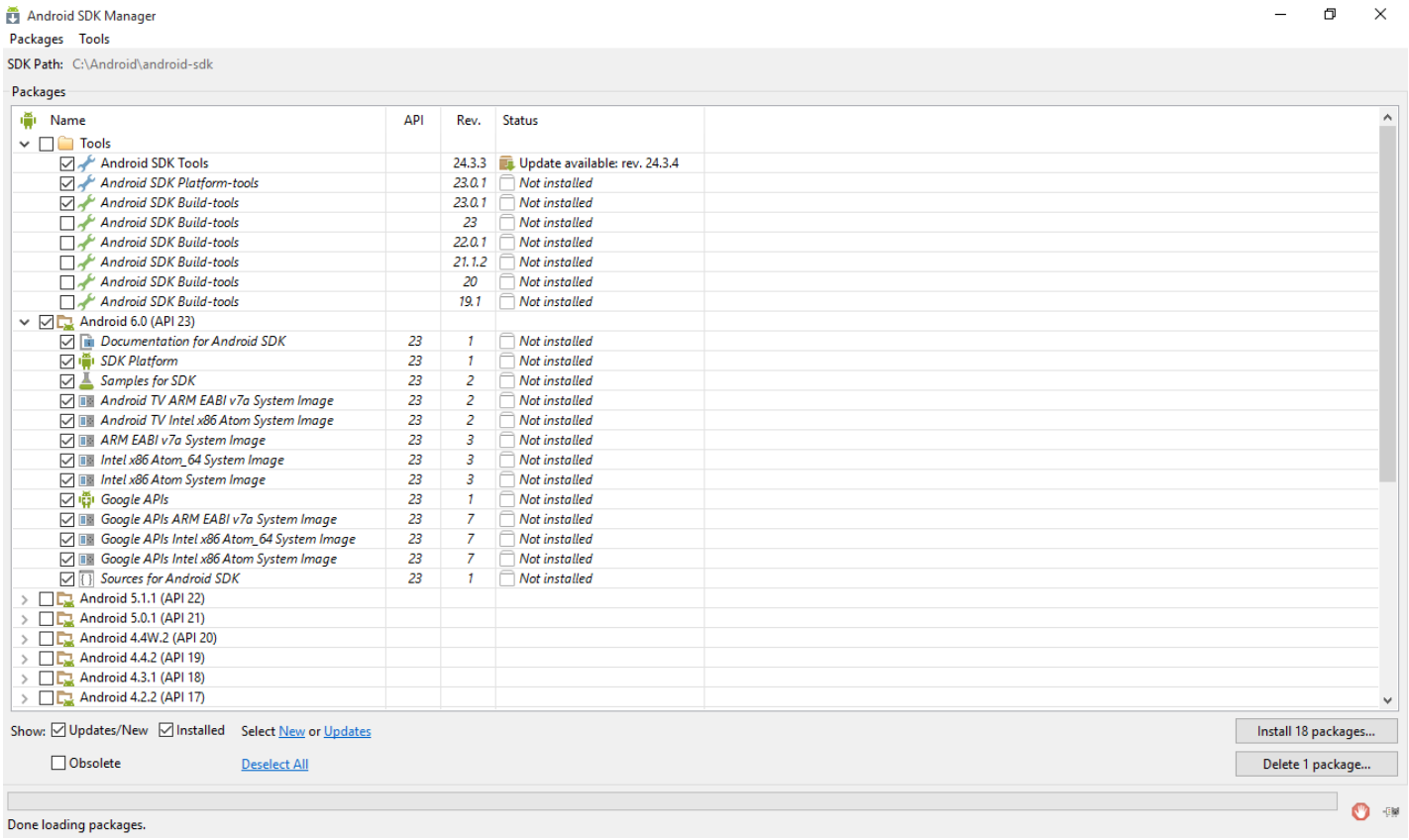


Figura 4: SDK Manager.

2.2.5 Herramientas del SDK de Android

Incluye varias utilidades para desarrollar las aplicaciones. Son herramientas esenciales para todos los desarrolladores. Se puede pensar en ellas como el corazón de las herramientas del sistema de la plataforma. Esto incluye el DDMS (*Dalvik Debug Monitor Server*). Se encuentra en el directorio *android-sdk/tolos*.

2.2.6 Herramientas de plataforma de Android

Son las herramientas adicionales que están desarrolladas en el núcleo de la plataforma, son actualizadas cada vez que es liberada una versión de la plataforma, esto incluye *ADB* (Android Debug Bridge), *fastboot* (protocolo de diagnóstico incluido en el *SDK*), *AIDL* (Android Interface Definition Language), entre otros.

2.2.7 Google APIs

Son las bibliotecas necesarias para usar servicios específicos de Google tales como Google Cloud, Messaging o los mapas.

2.2.8 Controladores (Drivers)

Contiene los archivos de controladores que permite a un dispositivo Android trabajar con una computadora, un ejemplo en el Google USB driver que depende del SO en el cual se esté desarrollando la aplicación.

2.2.9 Dispositivo virtual de Android

Para poder hacer pruebas de las aplicaciones en SO Android sin necesidad de disponer de un dispositivo móvil Android, el SDK incluye la posibilidad de definir un dispositivo virtual de Android ***ADV*** (Android Virtual Device). Este dispositivo emula una terminal con SO Android [6].

2.2.10 Dispositivo físico para el desarrollo de la aplicación

Para muchas actividades, es muy importante no depender completamente del emulador, es mejor probar la aplicación en un dispositivo físico, en particular cuando se quiere probar el procesamiento de gráficos avanzados, al utilizar servicios de posicionamiento o al hacer uso de sensores avanzados. Como es el caso del desarrollo de la aplicación, ya que se hará uso de los sistemas de posicionamiento GPS, A-GPS e Internet.

2.2.11 Configuración de un dispositivo físico para la implementación

Casi todos los dispositivos Android pueden ser usados para el desarrollo de aplicaciones. Para configurar un dispositivo, es necesario activar los ajustes de aplicaciones → Opciones de desarrollo → y activar la opciones de depuración USB, tal como se muestra en la Figura 5.

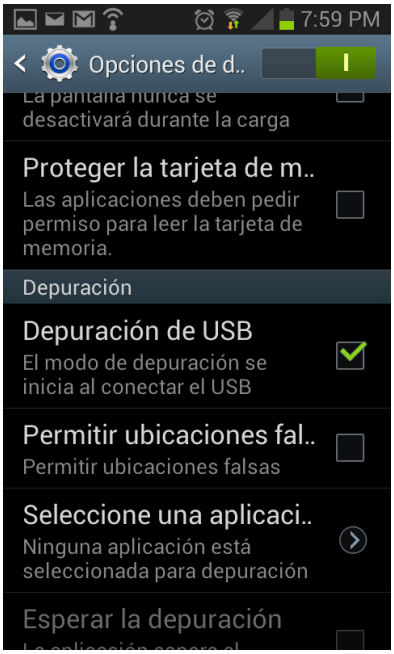


*Figura 5: Configuración de un dispositivo Android para la implementación (*Imagen obtenida desde un dispositivo Samsung S3 mini*).*

2.2.12 Android y GPS.

Actualmente hay muchas aplicaciones en el mercado que hacen uso de la localización del usuario para hacer uso de múltiples servicios como por ejemplo Google Maps. También es posible obtener la localización de un dispositivo Android utilizando las antenas de telefonía celular o mediante puntos de acceso Wi-Fi cercanos y cada uno de ellos tiene una velocidad, precisión y consumo de recursos del sistema distintos [6].

Posteriormente se hablará de la clase *GPSTracker.java*, que es la encargada de crear los métodos necesarios para poder usar el posicionamiento GPS y A-GPS.

2.2.13 Selección de plataforma de desarrollo

Antes de seleccionar una plataforma de desarrollo es necesario listar todas las características que demandará del sistema la aplicación y agregar, si se requiere, alguna característica especial.

2.2.14 Usuarios y sistema operativo Android

Se decidió que el sistema operativo Android es el adecuado debido a que la gran mayoría de usuarios cuentan con algún dispositivo móvil con sistema operativo Android y son más económicos.

En la Figura 6 se muestran los resultados obtenidos de un estudio que realizó la empresa Gartner relacionado a la cantidad de ventas de diferentes SO [7]:

En la Figura 7, se muestra un comparativo de los diferentes sistemas operativos existentes en telefonía celular en el mercado y porcentaje de uso en el año 2014.

	Venta de dispositivos en miles en los años		
Sistema Operativo	**2012**	**2013**	**2014**
Android	503,69	877,885	1,102,572
Windows	346,272	327,956	359,855
iOS/Mac	213,69	266,769	344,206
RIN	344,581	24,019	15,416
Chrome	185	1,841	4,793
Total	1,408,418	1,498,470	1,826,842

Figura 6: Datos obtenidos por la compañía Gartner[10]

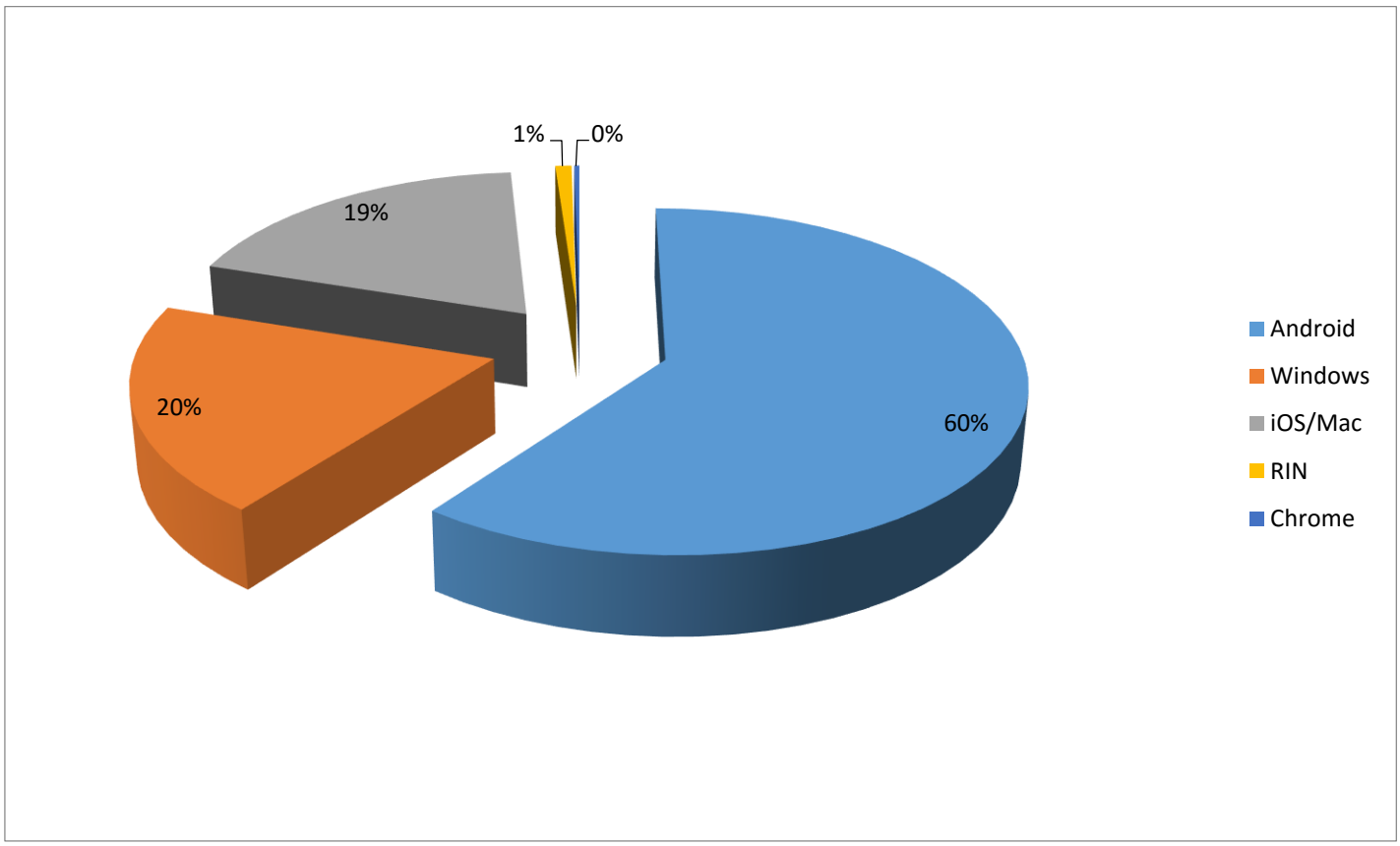


Figura 7: Porcentajes de SO más usados en el mundo en el año 2014.

[10] Gartner, es una compañía líder en investigación de tecnología de información mundial, fundada en 1979 con oficinas centrales en Stamford, Connecticut, en Estados Unidos [8].

2.3 PHP

Es un lenguaje que se interpreta como "*del lado del servidor*", es de open source. Se caracteriza por su potencia, versatilidad, robustez y modularidad. *PHP* es eficiente para tareas complejas de programación y es ampliamente usado para desarrollos web y puede ser embebido en *HTML*. Fue creado por el programador Rasmus Lerdorf como un conjunto de comandos para mantener su página web, que él liberó como (*Personal Home Page Tools PHP Tools*) versión 1.0, el 8 de junio de 1995 se extendió a una versión 2 liberada en 1997, y el nombre cambió a *PHP* (Hypertext Preprocessor). La versión 3 apareció tres años después. En la actualidad está instalado en más de 20 millones de sitios web [9].

En el caso de la aplicación del presente desarrollo *PHP* funciona como un gestor de conexiones cliente-servidor, realiza consultas a bases de datos, realiza cálculos de distancias geográficas entre el dispositivo móvil y el servidor externo, hace inserciones a la base de datos y envía los resultados de los cálculos realizados en el servidor al dispositivo móvil.

2.4 Open Source

Debido a que la aplicación fue desarrollada en su totalidad con software libre (open source), a continuación se definirá el concepto de ***open source.***

Open source es una filosofía y formas de trabajar que implican distintos factores como repercusiones tecnológicas, sociales y económicas, a continuación se hace un análisis más detallado del concepto.

Open source busca dar la libertad a los usuarios en utilizar el software, esto implica poner en completa disposición el código fuente, que en la mayor parte de los casos, puede ser copiado, modificado y redistribuido sin restricciones.

El movimiento open source tiene importantes consecuencias económicas, relacionadas con las sociales, las cuales favorecen a los usuarios. Todo esto se debe a que altera la forma tradicional de producción de software, puesto que no hay una organización empresarial con fines de lucro propietaria del software que se haya desarrollado. Este modelo está formado por desarrolladores y usuarios cuya finalidad es obtener un software de calidad.

Otro punto es que open source genera competencia y reduce efectos negativos en monopolios, lo cual obliga a las empresas a ser más competitivas y ofrecer productos de mejor calidad [10].

2.5 MySQL

La aplicación requiere del uso de *MySQL* porque fue necesario la ceración de una base de datos dentro del servidor y para poder obtener información de la base de datos del servidor externo, fueron necesarios códigos de MySQL dentro de los scripts de *PHP* .Por esta razón es importante en este proyecto.

Los códigos en MySQL que se utilizaron para el desarrollo de la aplicación son tanto de consulta, registro e inserción de datos dentro de la base de datos externa. A continuación se hace una breve definición de MySQL.

MySQL es el software más popular en el mundo de bases de datos, es utilizado para gestionar datos almacenados y es descrito como Data Base Management Software (DBMS) o Relational Data Base Management Software (RDBMS). *MySQL* fue creado por Michael Wildenius y David Axmark,. *SQL* significa (Structured Query Language). *MySQL* es un software gratuito. Es importante reconocer que *MySQL* y *PHP* son tecnologías “server-side”, es decir que se encuentran dentro de un servidor Web.

2.5.1 Ventajas de MySQL

Algunos de los principales competidores de *MySQL* son *postgreSQL*, *Microsoft SQL* server y *Oracle SQL*. Lo que hace fuerte a MySQL es [11]:

- **Desempeño**. MySQL es muy rápido para procesar la información.
- **Bajo costo.** MySQL está disponible sin costo, bajo una licencia de open source, o por un costo muy bajo por una licencia comercial.
- **Fácil de usar**. También es fácil de instalar.
- **Portabilidad.** MySQL puede ser usado en diferentes sistemas Unix y también en Windows®.
- **Código fuente abierto**. Al igual que php también se puede modificar el código fuente de *MySQL*.

2.6 Eclipse

En la actualidad Android cuenta con su propia plataforma de desarrollo (Android Studio), pero Android como es open source permite que su código fuente pueda ser utilizado en otras plataformas como Eclipse.

Eclipse es un entorno de desarrollo open source que permite trabajar debido a la arquitectura plugins con las bibliotecas de java *JDK*, *c++*, *xml*, etc. Para el caso en particular del desarrollo de la aplicación se instaló el *SDK* de Android dentro de Eclipse. Pero es indispensable tener instalado el *JDK* para poder trabajar con el *SDK* de Android debido a que requiere de bibliotecas de java.

Una vez que se ha instalado el SDK de Android en eclipse, en la plataforma de desarrollo de Eclipse se podrá visualizar los íconos correspondientes al SDK de Android así como la ADV Manager que sirve para emular los programas, pero debido a que en el caso de la aplicación se requerían

características especiales con las que no cuenta el ADV Manager fue necesario correr la aplicación en un dispositivo físico.

2.7 Relación cliente-servidor.

Con la finalidad de comprender mejor el proceso que se lleva a cabo dentro del servidor, a continuación se definirán los conceptos cliente y servidor.

Todos los servicios de Internet, se basan en una relación cliente-servidor. Esta relación es indispensable para el funcionamiento de lenguajes como *PHP*. En la red se denominan de la siguiente manera a los dispositivos conectados.

- **Servidores**: Computadoras que ofrecen servicios a todos los demás equipos conectados, por lo general tienen una presencia estable en la red, o dicho en otras palabras tener una dirección *IP* fija, en ellos se pueden alojar varios servicios como por ejemplo, las páginas web.

  Para este caso en particular el servidor gratuito está localizado físicamente en Alemania con una URL: http://www.awardspace.com/

- **Clientes**: Son equipos que los usuarios utilizan para conectarse a la red y solicitar servicios a los servidores. Generalmente los proveedores de acceso a Internet asignan a estos equipos una dirección *IP* durante su conexión, pero esa dirección es variable, es decir, cambia conexiones (IP dinámica) [12].

  En este caso el cliente es cualquier dispositivo con SO Android que cuente con los requerimientos mínimos (SO 2.0 o superior, acceso a Internet, sensor de *GPS* y espacio libre en memoria de 6MB).

Capítulo 3

Metodología

En este capítulo se explican los códigos en *Java*, *PHP* y *MySQL*, que se desarrollaron para la aplicación.

3.1 Flujo de ejecución de la aplicación

1. El usuario del dispositivo Android debe abrir la aplicación y posteriormente el dispositivo móvil obtiene la posición actual del usuario mediante *A-GPS* o *GPS*. A su vez mediante cualquier servicio de Internet disponible descargará del servidor externo el listado de estaciones que estén almacenadas en el servidor.
2. Ya que se obtuvo el listado completo de estaciones, el usuario deberá de seleccionar una estación de la cual quiere saber su estatus (cantidad de usuarios que se encuentran en la estación).
3. El dispositivo móvil realiza una petición al servidor para obtener los registros de todos los pasajeros que han ingresado a la estación elegida en un tiempo no mayor a un minuto (incluye a todos aquellos que aún siguen en la estación o que se encuentran dentro del autobús en dicha estación).

De esta forma el usuario con sólo dos toques de pantalla obtendrá los resultados. Este flujo de ejecución se muestra en la Figura 8:

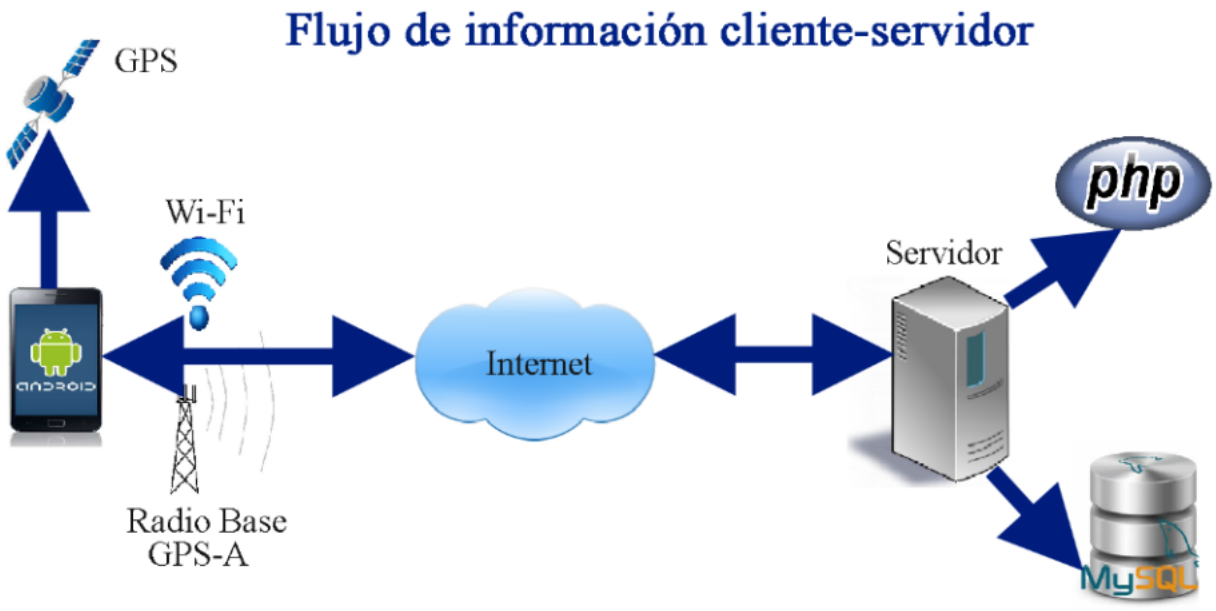


Figura 8: Petición del dispositivo Android al servidor.

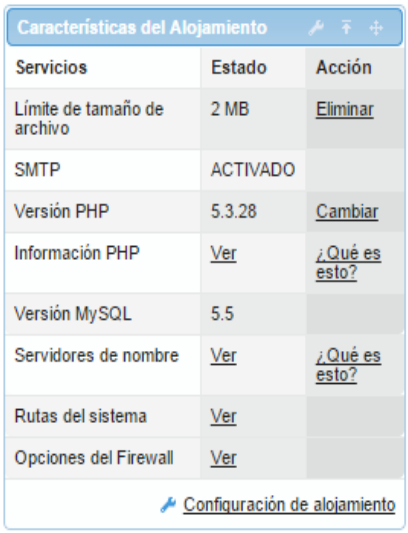
3.1 Creación de un dominio y obtención de un servidor

Lo primero que se llevó a cabo para el desarrollo de la aplicación es la creación de un dominio y la adquisición de un servidor con los servicios de *PHP* y *MySQL*.

El dominio que se creó para el desarrollo es el siguiente: http://www.proyectouacmmetrobus.tk el cual estará vigente por un año a partir del 1 de Diciembre de 2014 y se encuentra alojado en el servidor de awardspace.com.

3.2 Características del servidor

El servidor donde se alojó el dominio cuenta con las características técnicas mostradas en las Figuras 9 y 10:

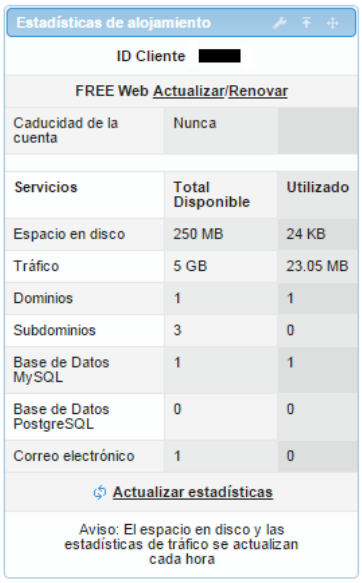


Características del Alojamiento

Servicios	Estado	Acción
Límite de tamaño de archivo	2 MB	Eliminar
SMTP	ACTIVADO	
Versión PHP	5.3.28	Cambiar
Información PHP	Ver	¿Qué es esto?
Versión MySQL	5.5	
Servidores de nombre	Ver	¿Qué es esto?
Rutas del sistema	Ver	
Opciones del Firewall	Ver	

Configuración de alojamiento

Figura 9: Características y servicios del servidor



Estadísticas de alojamiento

ID Cliente [redacted]

FREE Web Actualizar/Renovar

Caducidad de la cuenta	Nunca	
Servicios	**Total Disponible**	**Utilizado**
Espacio en disco	250 MB	24 KB
Tráfico	5 GB	23.05 MB
Dominios	1	1
Subdominios	3	0
Base de Datos MySQL	1	1
Base de Datos PostgreSQL	0	0
Correo electrónico	1	0

Actualizar estadísticas

Aviso: El espacio en disco y las estadísticas de tráfico se actualizan cada hora

Figura 10: Características y servicios del servidor

Se debe de poner especial atención a la información de los servicios de *PHP* y *MySQL*, ya que son los servicios principales para desarrollar la aplicación.

El servidor awardspace.com limita al cliente con cuentas gratuitas con una sola base de datos *MySQL* con 10MB de espacio total y un máximo permitido de consultas de 3,600/h y se inhabilitará si no hay actividad en 12 meses.

3.3 Creación de la tabla Estaciones

Dentro de la base de datos se encuentra una tabla llamada *Estaciones* y es en ésta donde se encuentran alojadas todas las estaciones con las que cuenta (cinco líneas o rutas y 204 estaciones)[11]. La creación de la tabla se realizó con los comandos en *MySQL,* que se muestran en el Código 1.

```
CREATE TABLE `Estaciones` (
`id_estacion` int(11) NOT NULL AUTO_INCREMENT,
`Estacion` text COLLATE utf8_spanish_ci NOT NULL,
`longitud` text COLLATE utf8_spanish_ci NOT NULL,
`latitud` text COLLATE utf8_spanish_ci NOT NULL,
`linea` text COLLATE utf8_spanish_ci NOT NULL,
PRIMARY KEY (`id_estacion`),
FULLTEXT KEY `Estacion` (`Estacion`)
) ENGINE=MyISAM  DEFAULT CHARSET=utf8
COLLATE=utf8_spanish_ci AUTO_INCREMENT=1 ;
```

Código 1: Tabla Estaciones.

[11] Cantidad de estaciones y líneas disponibles hasta el 23 de junio de 2014. En el anexo se indica la base de datos completa con todo el listado de líneas y estaciones.

La tabla cuenta con cinco campos el primero es "*id_estacion"* es de tipo entero auto incrementable y sirve para identificar a cada estación dentro de la base de datos, el segundo campo es "*Estación"* de tipo texto; es el nombre de cada estación, la tercera es "*longitud"* y es de tipo *double*; indica el parámetro de posición *"longitud",* el cuarto campo es la *latitud* de tipo *double; indica* el parámetro de posición *"latitud"* y por último el campo "*línea"* de tipo texto que identifica a cada estación de cada ruta, principalmente útil si hay más de una estación con el mismo nombre pertenecientes a distintas rutas, por ejemplo: estación *circuito* perteneciente a la línea 1 y estación *circuito* perteneciente a la línea 3.

3.4 Inserción de datos en MySQL.

Cabe mencionar que el sistema de transporte público metrobús proporcionó un listado en formato Excel con todas las estaciones y la ubicación geográfica de cada una, los datos de longitud y latitud se encontraban en coordenadas sexagesimales "en grados y minutos" y para realizar los cálculos necesarios con ellas es necesario que se expresen en coordenadas decimales.

Para hacer los cálculos de distancias es necesario que las coordenadas sexagesimales sean convertidas en coordenadas decimales, la forma de realizar la conversión de grados sexagesimales a decimales se realiza con la Ecuación 1.

$$Grados + \left(\frac{minutos}{60}\right) + \left(\frac{segundos}{3600}\right) \rightarrow Ecuación\ (1)$$

Ecuación 1: Conversión de grados sexagesimales a decimales

Ahora sólo se determina el signo dependiendo en los meridianos en los que se encuentren.

En el caso de las coordenadas en México para la latitud son positivos y para la longitud son negativos.

En el Anexo se muestra la inserción de todos los datos de la tabla “Estaciones” con los valores de longitud y latitud en decimal.

3.5 Implementación local y con base de datos externa para la aplicación

3.5.1 Implementación local usando SQLite

Durante la implementación de la aplicación fue necesario realizar algunas pruebas y se propusieron dos posibles soluciones, para ello se escogió la más conveniente y para cuestiones prácticas, cabe mencionar que ambas soluciones son útiles y funcionales. En un principio se planteó que la aplicación obtuviera del servidor externo la información necesaria y posteriormente se llevaran a cabo todos los cálculos y procesos aritmético-lógicos dentro del dispositivo Android, pero se optó por realizar todos los cálculos en el servidor externo y aunque los resultados fueron muy semejantes, había que tomar en cuenta los resultados para poder determinar cuál de las dos opciones era la más adecuada.

Antes de desarrollar la implementación local, hay que recordar que es necesario que el dispositivo móvil tenga acceso a una base de datos, la cual también en su momento se planteó implementarla de forma local haciendo uso de la versión de *SQL* con la que cuentan todos los dispositivos móviles.

En este apartado sólo se mencionará la posibilidad de una base de datos interna en ***SQLite***, no se abordará extensamente ya que no fue la mejor de las opciones para poder realizar la aplicación.

SQLite es un sistema de gestión de base de datos como lo es *SQL* o *MySQL**,*** la diferencia radica principalmente en el pequeño espacio que ocupa, es por eso que los dispositivos móviles cuentan con este sistema de gestión de bases de datos. El problema principal que representa el usar *SQLite* es que la base de datos se genera de forma interna en la memoria del dispositivo Android, el hecho

que la base de datos se encuentre dentro del dispositivo móvil implica que para poder realizar alguna modificación dentro de la base de datos del dispositivo móvil, sería necesario realizar una nueva versión de la aplicación, en otras palabras hay que programar desde Android *SQLite*, lo cual resulta ser muy poco práctico en caso de que se requiera realizar actualizaciones o ajustes a la base de datos.

Una vez que se analizó el caso de tener una base de datos interna, se determinó que, la base de datos estuviera en un servidor del cual el cliente (el dispositivo Android) pudiera obtener toda la información necesaria para la aplicación, uno de los motivos principales por lo cual se decidió esta solución fue porque la actualización de una base de datos interna en la misma aplicación, tendría como desventaja la creación de una nueva base de datos cuando la información del sistema de transporte metrobús cambie, como por ejemplo la aparición de una nueva línea o ruta, esto significa que se tendría que crear una nueva aplicación que contenga la nueva línea. Por otro lado si la base de datos está alojada en un servidor externo, sólo se debe de actualizar la información en la base de datos y el usuario no requeriría descargar una actualización de la aplicación.

3.5.2 Permisos en el dispositivo Android.

Cada vez que se haga uso de características específicas de algún dispositivo Android es necesario declarar y asignar los servicios con los que contará la aplicación, es decir, cada vez que se quiera acceder a alguno de los sensores con los que cuenta el dispositivo móvil es necesario declarar los recursos a los cuales tendrá acceso la aplicación. La aplicación requiere acceder a tres recursos, el acceso a Internet, posicionamiento *GPS* y posicionamiento *A-GPS*.

A continuación se muestran la forma de declarar los permisos y los servicios con los que contará la aplicación.

Dentro del archivo ***AndroidManifest.xml*** se asignaron los siguientes permisos, tal como se muestra en la Tabla 3.

Permiso	código en Android
Acceso a Internet	Android.permission.INTERNET
Acceso a localización por GPS	Android.permission.ACCESS_FINE_LOCATION
Acceso a localización por A-GPS	Android.permission.ACCESS_COARSE_LOCATION

Tabla 3: Permisos necesarios para la aplicación.

3.6. Interfaz visual.

La interfaz visual se llevó a cabo mediante los recursos gráficos con los que cuenta Android.

Para la actividad principal ***Activity_main.xml*** se hizo uso de un *LinearLayout*, un *TextView* y un *ListView*, los cuales tienen las siguientes características.

3.6.1 Actividad principal visible para el usuario (Activity_main.xml)

La actividad que es la encargada de realizar la interfaz visual es la actividad *Activity_mail.xml* y consta con un *LinearLayout* el cual contiene un *textview* y un *listview*, en este último se va desplegando el listado de estaciones que verá el usuario.

LinearLayout es el diseño que se eligió para la aplicación y es lineal porque los elementos dentro de él van seguidos uno de otro. Es uno de los diseños con los que cuenta Android, el cual sólo coloca los componentes visuales uno tras de otro.

Textview sólo le indica un mensaje impreso al usuario dentro del *layout* en este caso el mensaje es “Verificar el estatus de la estación”. Es un componente del tipo cuadro de texto con los que cuenta Android.

Listview es el encargado de mostrar los elementos deseados en forma de lista, en este caso es un listado de estaciones con su respectiva línea. Es un componente en Android capaz de mostrar un conjunto de elementos de datos, como tipo tabla.

3.6.2 TextView visibles en la aplicación (List_v.xml)

El archivo ***List_v.mxl*** sólo consta de dos objetos *TextView*, la cantidad de objetos *TextView* que se encuentren en el archivo *List_v.xml* serán a su vez la cantidad de datos que se quieran visualizar, en este caso sólo se desea ver la Estación y la línea.

Para la aplicación los dos objetos tipos *TextView* creados están ligados a un *id* o por una identificación de tal forma que un *TextView* tiene un ID llamado Estación1 y el otro línea1 y corresponden a las estaciones y a las líneas respetivamente. Son los *TextView* que se visualizarán cuando sean llamados por el objeto del tipo *ListView,* al momento que la aplicación obtenga los registros de la base de datos.

Cuando se realiza una aplicación en Android se deben de definir los tamaños de pantallas en las que estará disponible la aplicación, esto se debe hacer para evitar problemas de ejecución y garantizar la disponibilidad de la aplicación en cualquier tamaño de pantalla. Para la aplicación se hicieron las consideraciones necesarias para lograr que se ejecute en cualquier tipo de pantalla.

Para crear una versión de la aplicación diferente, para cada tamaño de pantalla, es necesario que dentro del *SDK* de Android se localice una carpeta llamada “*res”,* la cual contiene el layout actual y

una vez dentro de ella se crean cinco carpetas más con los siguientes nombres *layout-land, layout-large, layout-small, layout-xlarge y layout-xlarge-land.* Cada una de las carpetas representa un tamaño predefinido por Android y una vez creadas se deben de crear los mismos archivos *XML* mencionados con anterioridad, con los mismos objetos y nombres. Esto es sumamente importante porque posteriormente serán llamados por la actividad principal y si uno de sus objetos no tiene el mismo nombre que los definidos en el *layout* principal no se ejecutarán correctamente. En la figura 11 se muestra cómo se deben de crear las carpetas que contienen los archivos *XML* para distintos tipos de pantallas.

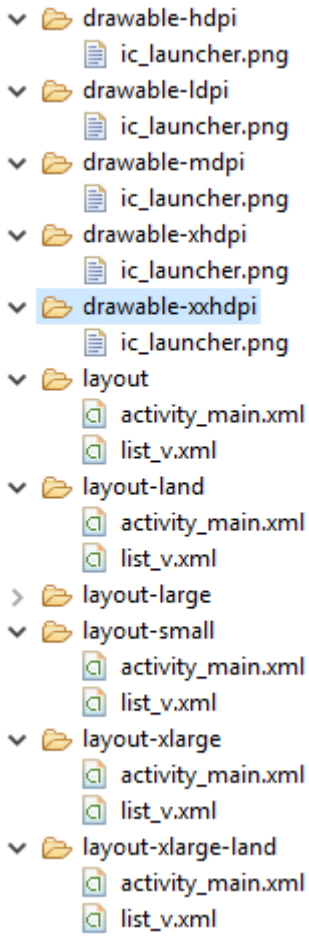


Figura 11: Todos los archivos xml dentro de las carpetas layout tienen los mismos nombres y objetos que la carpeta layout principal.

La implementación de cada una de las formas de visualización dependerá en gran medida del programador o diseñador, en este caso sólo se abordó el tamaño estándar de pantalla, aunque si se implementaron todos los tamaños de pantalla posibles.

3.7 Archivos PHP y clases de java para el desarrollo de la aplicación.

En este apartado se detallará cada uno de los archivos en *PHP* y las clases en java necesarios para la aplicación. Para un mejor entendimiento del código fuente de la aplicación, los archivos *PHP* y las clases en java se explicarán mediante pseudocódigos o diagramas de flujo.

Para que Android pueda conectarse a la base de datos en *MySQL*, se requirió crear cuatro archivos en *PHP*, los cuales garantizan la conexión entre el servidor y el cliente, de igual forma, permiten realizar consultas o inserciones a la base de datos. A continuación se explicarán más a fondo cada uno de los archivos en *PHP*.

3.7.1 Crear conexión con la base de datos (Conectar.PHP)

El script ***Conectar.php*** tiene como función crear los permisos necesarios para acceder a la base de datos, en él se encuentran los usuarios, contraseñas, el nombre de la base de datos y tablas a las que se debe conectar la aplicación. A continuación se muestra el código 2 Conectar.php.

```
<?php

header( 'Content-Type: text/html;charset=utf-8');

define('HOST','fdb5.awardspace.net');

define('USER', '#######');

define('PASS', '#######');

define('DB', '##########);

class conectar{

public static function con(){

$con = mysql_connect(HOST,USER,PASS); mysql_query("SET NAMES:
utf-8"); mysql_query(DB); return $con;     } } ?>
```

Código 2 Conectar.php..

3.7.2 Consulta a una base de Datos MySQL desde PHP (consultaDataBase.PHP).

Con este script se puede realizar una consulta completa de la base de datos, obteniendo así el listado de estaciones y su posición geográfica. Un hecho importante y que no se debe perder de vista es que en la consulta se realiza una codificación tipo *JSON*, es importante tenerla en mente ya que esta información en este tipo de codificación será consultada por el dispositivo Android y se tendrán que hacer las clases de java necesarias para que el dispositivo tenga acceso a la información. A continuación en el Pseudocódigo 1 se describe el archivo ***consultaDataBase.php.***

Programa **consultaDataBase.php**

1. Llamar a *conectar.php.*
2. Iniciar un arreglo.
3. *Consulta = $query*.
4. Mientras (Devolver un arreglo y mover a la siguiente posición){
   []Estaciones= $query.
   }
5. Realizar codificación JSON a Estaciones.

Pseudocódigo 1: consultaDataBase.php.

Sí se consulta el archivo ***consultaDataBase.php*** desde algún explorador de Internet la información se visualizará como se muestra en la Figura 12.

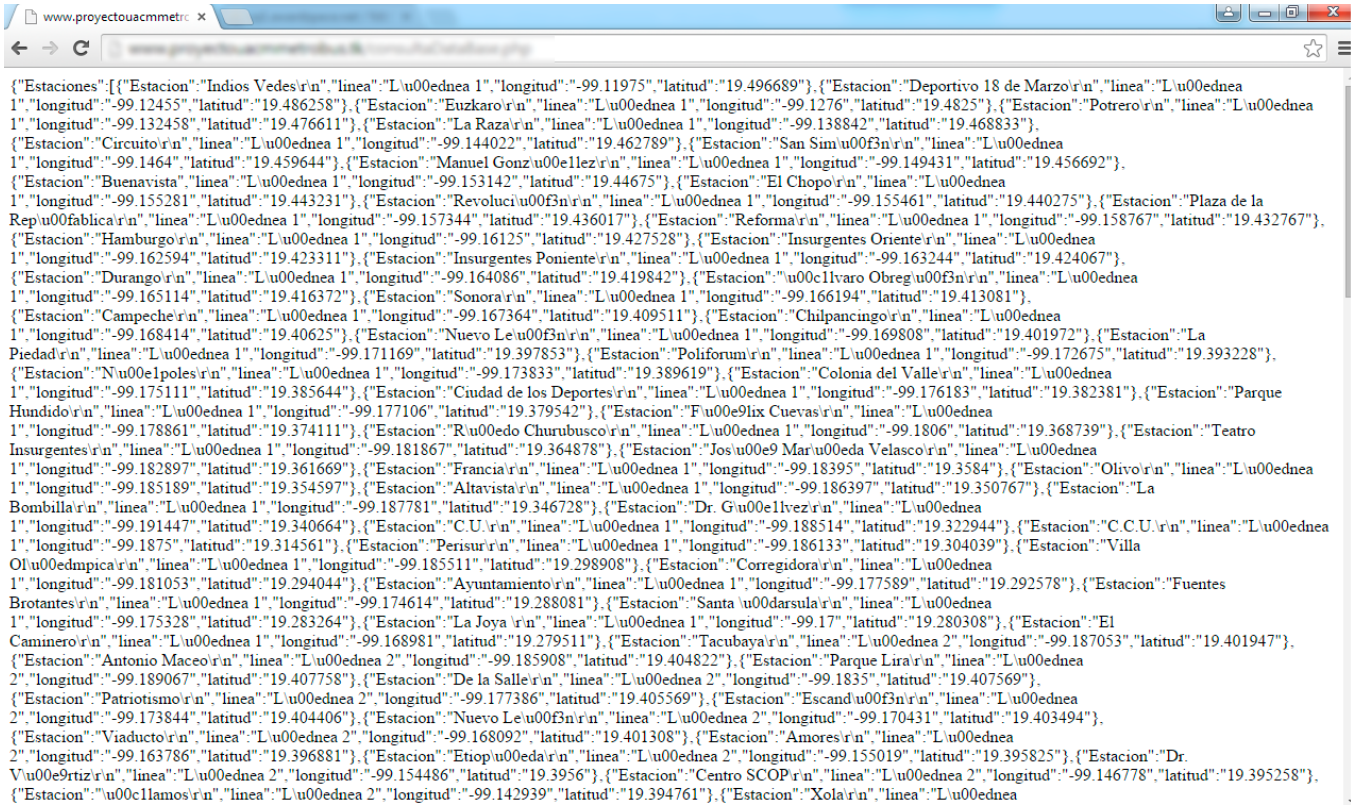


Figura 12: Vista de todos los registros dentro de la tabla Estaciones en formato json obtenida con el navegador de Internet Google Chrome.

3.7.4 Condición de distancia mínima entre el usuario y una estación (MinDistancia.PHP).

Para determinar la distancia que hay del usuario a cada una de las estaciones de forma remota es necesario crear un archivo *PHP*, que obtendrá como parámetros de entrada la longitud y latitud del dispositivo móvil y posteriormente realizar una *Consulta* en la base de datos para obtener el listado de todas las longitudes y latitudes de las estaciones dentro de la base de datos.

Una vez que se obtuvo un *Arreglo* que contenga las distancias del usuario a cada una de las estaciones, se determina cuál es la más cercana, “arbitrariamente se determinó una distancia en particular de 450 (m)” y de cumplirse esta condición de distancia se agregará un usuario más a la base de datos, para que pueda ser contabilizado.

A continuación se explicarán las funciones que lleva a cabo el archivo *MinDistancia.php*.

1. Inicia la conexión con la base de datos que contiene todos los datos de las estaciones del metrobús.
2. Obtiene los parámetros fecha y hora, los cuales como su nombre lo dice, obtienen la fecha y hora actual utilizando como zona horaria la Ciudad de México.
3. Convierte grados a radianes, ya que se trabajará con funciones trigonométricas.
4. Obtiene la dirección *IP* del dispositivo que realice la consulta a la base de datos, con la finalidad de poder identificar a un usuario en particular.
5. Realiza una *Consulta* que haga una selección de todas las estaciones que se encuentran en la base de datos.
6. Obtiene del dispositivo móvil las variables “*longitud”* y “*latitud”*, ya que con ellas se podrá determinar la cercanía del dispositivo móvil con cualquier estación.
7. Con la Ecuación 2 se obtiene la distancia del dispositivo móvil a todas las estaciones y los resultados son guardados dentro de un *Arreglo* de distancias.

8. Define una condición de distancia mínima que para este caso fue de 450 (m).
9. En caso de que sea cumplida la condición de distancia mínima, se obtiene la posición que ocupa el valor de distancia mínima dentro del *Arreglo* de distancias y posteriormente se determina cuál es la estación más cercana, realizando una *Consulta* a la tabla *Estaciones* y se compara el valor con la posición del Arreglo de distancias más uno, con el valor del *id* de la estación, es decir, si el lugar del menor valor dentro del *Arreglo* de distancias es de la forma: [0,1,**2**,3,4,5,…n] y los id_estacion son: [1,2,**3**,4,5,6,…n+1], debido a que los *id* de las estaciones inician en 1 y las posiciones del Arreglo de distancias inician en 0, para que coincidan es necesario sumar uno al valor de la posición del Arreglo de distancias.
10. Realiza una *Consulta* para ingresar a la tabla "*DatosUsuarios",* los datos de los usuarios que hayan cumplido con la condición de distancia mínima. Los datos ingresados a la tabla *DatosUsuarios* son: el identificador del usuario, la estación y línea, así como la fecha, hora y dirección *IP* del dispositivo móvil.

La fórmula para calcular la distancia de un punto a otro en una esfera (se debe recordar que la tierra tiene una forma muy parecida a una esfera) está determinada mediante la Ecuación 2.

$$\boldsymbol{d = R\,cos^{-1}\left(cos\,\lambda_1\,cos\,\lambda_2\,cos(\,\phi_2 - \phi_1) + sin\;\lambda_1 sin\;\lambda_2\;\right)} \rightarrow\; Ecuación\,(2)$$

Donde:

R = radio de la esfera (radio de la tierra).

λ_1 *= latitud del dispositivo móvil.*

λ_2 *= latitud de la estación.*

ϕ_1 = *longitud del dispositivo móvil.*

ϕ_2 = *longitud de la estación.*

Se puede acceder a la información guardada en el arreglo mediante un índice, para este caso el primer elemento del arreglo es 0:

0	1	2	3	4	5	6	7	8	9	n

Posición de un elemento dentro de un Arreglo.

En el caso del **id_estacion** de la base de la tabla estaciones comienza desde 1.

1	2	3	4	5	6	7	8	9	10	m

Orden de una tabla por id.

3.7.5 Archivo PHP (Peticion.PHP) encargado de realizar una consulta a la base de datos MySQL desde un cliente Android.

Este archivo en *PHP* tiene la función de recibir la petición del usuario, es decir, es el encargado de enviar los datos finales que requiere el usuario que en este caso es la cantidad de usuarios que hay en una determinada estación. La selección de la estación se explicará más adelante en la clase ***MainActivity.java***, ya que para el entendimiento de esta clase es necesario examinar cómo funcionan todas las clases y archivos *PHP* que dependen de ella.

En el siguiente cuadro se muestra el pseudocódigo 3 correspondiente al archivo *Peticion.php.*

Programa **Peticion.php**

1. Definir como zona horaria la Ciudad de México.
2. Entorno: fecha = fecha actual y hora = hora actual.
3. Entorno: recibir del dispositivo Android Estación seleccionada = $Est y Línea Seleccionada = $Lin.
4. Iniciar conexión con la base de datos.
5. Entorno variables tipo String (Estación y línea).
6. Realizar una consulta a la tabla DatosUsuarios, donde *Estacion = $Esta* y *línea = $Lin* y *fecha = fecha de consulta* y *hora = hora de consulta*.
7. Consulta = con.
8. Imprimir con.

Pseudocódigo 3: Archivo Peticion.php.

3.8. Clases en java para el desarrollo de la aplicación en Android.

En el siguiente apartado se analizarán los códigos necesarios en Android para el desarrollo de la aplicación.

Todos los códigos que se presentarán a continuación fueron implementados en el entorno de desarrollo de Eclipse IDE y el *SDK* de Android. Cabe mencionar que el desarrollo de la aplicación se realiza de forma muy similar a una aplicación de java, es por eso que la aplicación tiene una estructura basada en archivos de java llamados clases, los cuales tienen una función específica y aunque se pudo haber programado todo el proyecto en una sola clase de java, se decidió que tuviera una estructura basada en clases individuales para cada función de la aplicación para poder reutilizar las funciones que se encuentren dentro de una determinada clase y al mismo tiempo facilitar la localización de cada función así como facilitar la lectura del código.

3.8.1 Codificación JSON (JSONParser.java).

La Clase ***JsonParser.java*** tiene como objetivo realizar una codificación de tipo *JSON*, que recordando, el archivo ***consultaDataBase.php*** arroja un listado completo de estaciones con su ubicación correspondiente en formato *JSON*, es por eso que Android debe ser capaz de realizar la consulta de dichos datos en codificación *JSON*, de esta manera se garantiza que los datos podrán ser consultados sin problema por el dispositivo Android. En la Figura 13 se muestra el diagrama de flujo de la clase *JSONParser.java.*

En el siguiente cuadro se muestra el Pseudocódigo 2 de la clase JSONParser.java.

Programa **JSONPareser.java**

1. Clase JSONParser.
2. Crear una función JSONObjet (String url)
3. Entorno inicial JSONObject *jObj* = null;
4. Entorno Iniciar String *json* = "";
5. Leer caracteres en iso-8859-1 //para caracteres latinos.
6. Variable para concatenar sb.
7. String line = null;
8. Mientras (La lectura de la line != null **)**{
   Anexar todas las cadenas a line\n
9. }
10. Json = sb.line.
11. Regresar valores en la variable jObj.

Pseudocódigo 2: Clase JSONParser.java

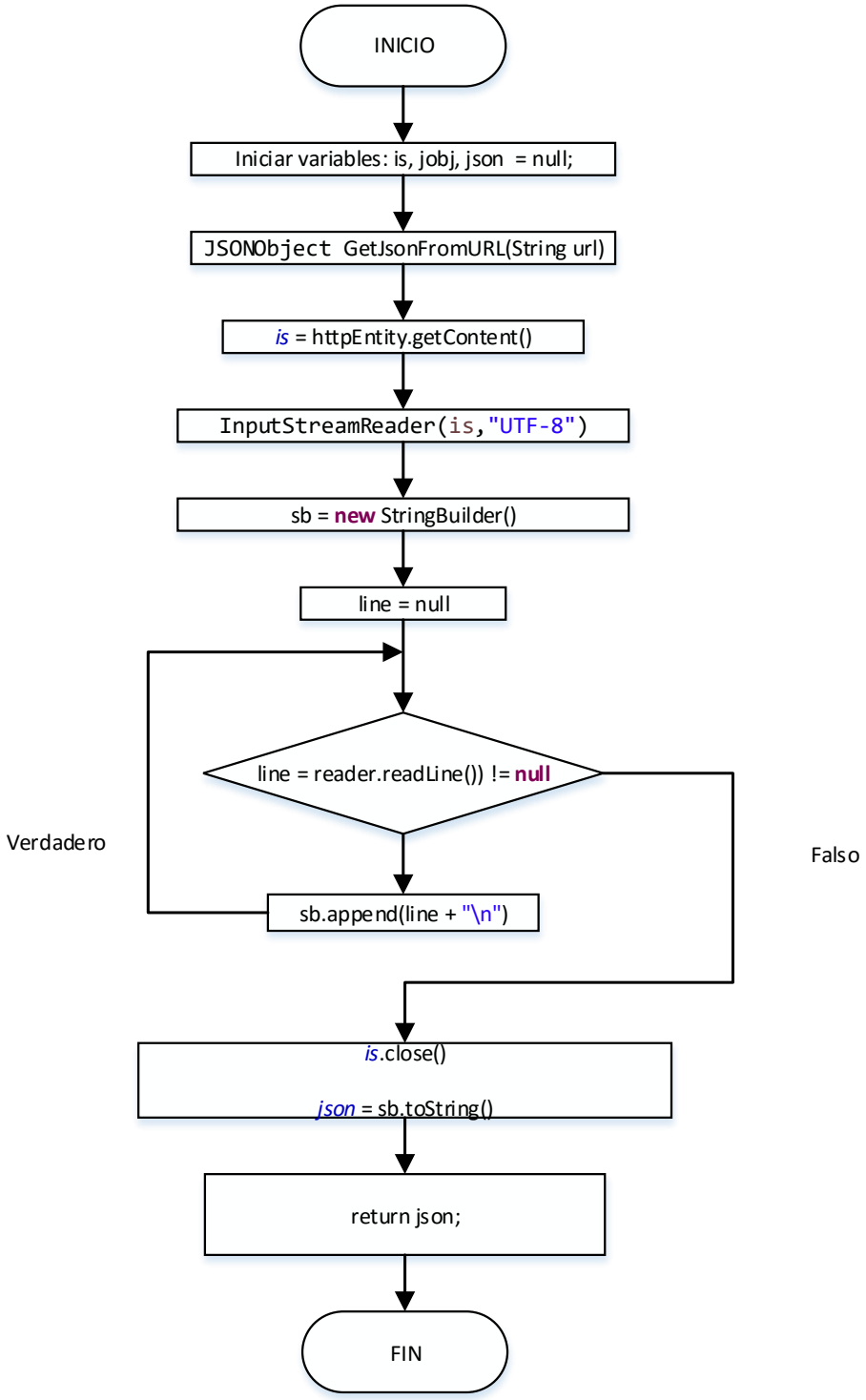


Figura 13: Diagrama de flujo de la clase JSONParser.java.

3.8.2 Implementación de la clase en java encargada de gestionar los servicios de GPS (GPSTracker.java).

La aplicación requiere de la posición del usuario, por eso se implementó una clase en java para poder acceder a los datos con los que cuenta el dispositivo de *GPS*.

La posición del *GPS* debe actualizarse cada determinado tiempo y cada vez que el usuario cambie su posición, hay que tomar en cuenta que entre más actualizaciones realice el dispositivo más rápido se consumirá la batería del dispositivo, en este caso se puso como parámetro que no importa la distancia de actualización pero que el *GPS* debe de actualizarse cada 20 (s). El tiempo de actualización del *GPS* se determinó que fuera de 20 (s) porque los sensores de *GPS* con los que cuentan los dispositivos móviles demandan más energía al dispositivo, lo cual implica que entre más actualizaciones realice el *GPS* de las coordenadas, más energía consumirá del dispositivo. A continuación se muestra el Código 2 encargado de realizar actualizaciones de las coordenadas de *GPS*.

```
private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 0;


private static final long MIN_TIME_BW_UPDATES = 1000 *20;
```

Código 2. Actualiza cada 20 (s) las coordenadas de GPS sin importar la distancia recorrida por el dispositivo.

En la Figura 14 se muestra el diagrama de flujo de la clase ***GPSTracker.java*** con la finalidad de tener mejor entendimiento de dicha clase.

Explicación del diagrama de flujo de la Figura 14:

1. Se verifica si es posible obtener la posición mediante *A-GPS*, de no ser así, se obtiene la posición mediante *GPS*.
2. Después se obtiene la longitud y la latitud del dispositivo móvil.
3. En caso de no ser posible obtener los datos de posicionamiento porque no estén activados en el dispositivo móvil, se mostrará un cuadro de diálogo para dar la opción de activarlos.

A continuación en el Código 3 se muestran las funciones creadas en java para obtener la posición geográfica.

Función que obtiene la latitud.

```
public double getLatitude(){
if(location != null){
latitude = location.getLatitude();
}
```

Función que obtiene la longitud.

```
public double getLongitude(){
if(location != null){
longitude = location.getLongitude();
}
return longitude;
```

Código 3: Funciones para obtener la posición geográfica en Android.

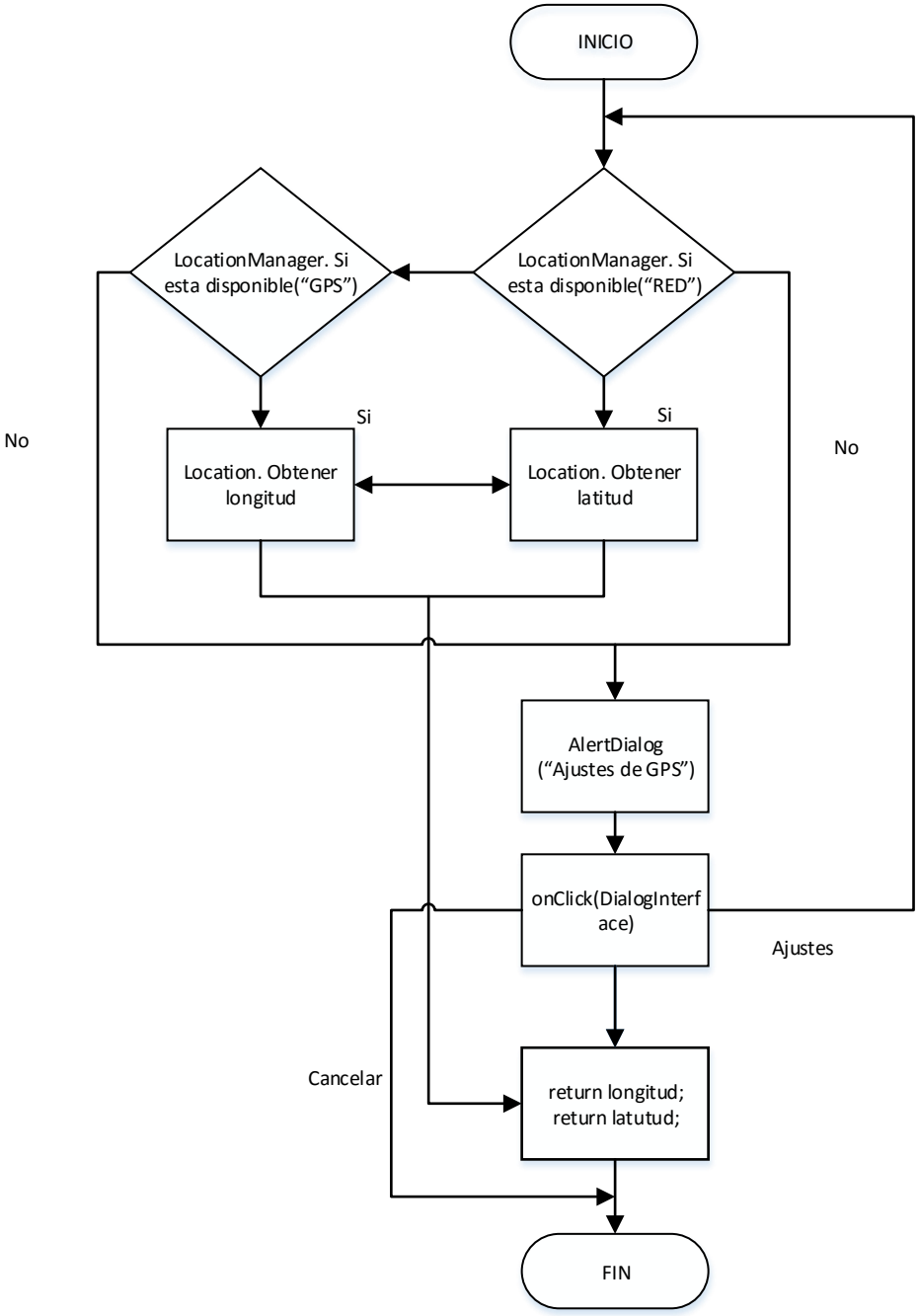


Figura 14: Diagrama de flujo de la clase GPSTracker.java

Posteriormente se debe de asegurar que estén activados los servicios de posicionamiento en el dispositivo móvil porque es necesario obtener la longitud y latitud del dispositivo móvil. En caso de que los servicios de posicionamiento no estén activados en el dispositivo móvil, se deberá dar al usuario la opción de activarlos o no, como se muestra en las Figuras 15, 16 y 17.

El siguiente método está encargado de desplegar una pantalla para poder activar los servicios de posicionamiento. Para eso se creó un cuadro de diálogo que permite activar los servicios de posicionamiento. En el Código 4 se muestra cómo se creó el cuadro de diálogo para activar los servicios de GPS.

Se colocó el título del cuadro de diálogo.

```
alertDialog.setTitle("Ajustes de GPS");
```

Se programó un mensaje de alerta de la siguiente forma:.

```
alertDialog.setMessage("Los servicios de posicionamiento no están disponibles. \n¿Quieres ir a ajustes de GPS?");
```

Código 4: Cuadro de diálogo para los ajustes de GPS.

En la Figura 15 se muestra la visualización del cuadro de diálogo en caso que no estén activados los servicios de posicionamiento.

Para el caso en el que él usuario haya presionado el botón "*Ajustes*" se visualizará el menú de la Figura 16 (El menú de opciones puede variar dependiendo el modelo de dispositivo móvil con el que cuente el usuario). En la Figura 17 se muestran todos los servicios con los que cuenta el dispositivo móvil estando activados.

Figura 15: Mensaje de alerta para indicar disponibilidad de los servicios de posicionamiento.

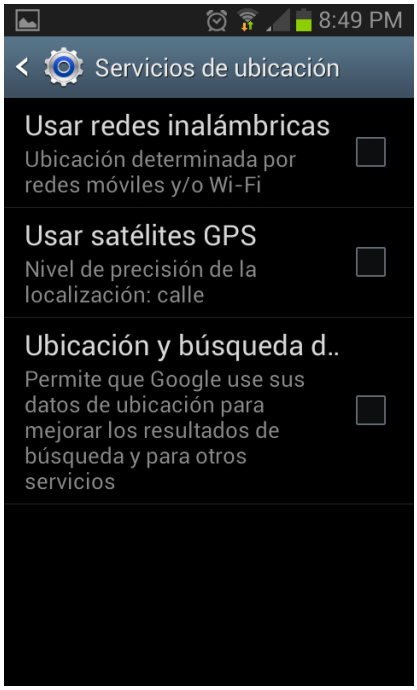


Figura 16: Métodos de posicionamiento con los que cuenta el dispositivo móvil y están desactivados.[12]

[12] Visualización obtenida de un dispositivo móvil Android 4.1.2 Jelly Bean Dispositivo "SAMSUNG S3 MINI GT- I8190L", la visualización puede variar dependiendo el modelo de dispositivo móvil así como versión de SO Android

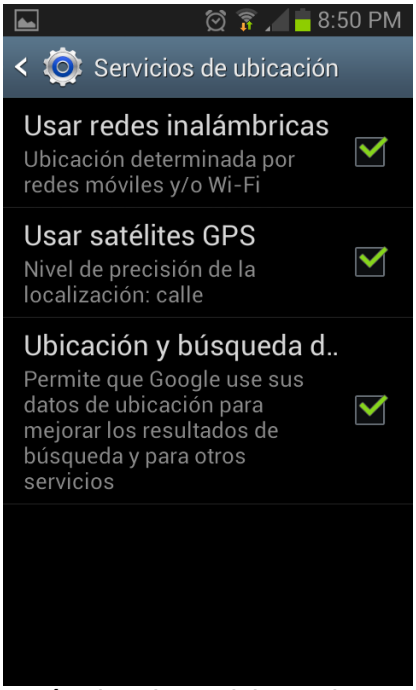


Figura 17: Métodos de posicionamiento activados.

Cuando se implementa la interfaz ***LocationListener*** en Android, se agregan automáticamente 4 métodos que tienen distintas funciones, y aunque no se implementaron en el desarrollo de la aplicación es importante conocerlos. Estos métodos son:

1. **onLocationChanged().** Sirve para realizar una acción cada vez que cambie la localización del dispositivo móvil.
2. **onProviderDisabled().** Se llama cuando el proveedor de posicionamiento ha sido desactivado por el usuario.
3. **onProviderEnabled().** Se llama cuando el proveedor de posicionamiento haya sido habilitado por el usuario.
4. **onStatusChanged().** Se llama cuando el estado del proveedor de posicionamiento cambia.

3.8.3 Inserción a la base de datos en MySQL (InsertarDatos.java).

La clase ***InsertarDatos.java*** es la clase asignada para enviar al servidor los datos que se requieren, en este caso los datos que se obtendrán del dispositivo móvil son la longitud y la latitud actual del usuario y posteriormente serán requeridos por el archivo ***MinDistancia.php***, (del cual se habló en la sección 3.7.4) ya que es el archivo encargado de determinar la cercanía del usuario con una estación. En la Figura 18 se muestra el diagrama de flujo de la clase ***InsertarDatos.java***.

La clase *InsertarDatos.java* consta de una función llamada *insert* que recibe como parámetros de entrada dos variables del tipo *String* que son la longitud y la latitud de la estación que seleccione el usuario.

Posteriormente se abre la conexión para conectarse a una url, en este caso la url es la dirección donde se encuentra alojado el archivo *MinDistancia.php*, se obtienen los datos en formato UTF-8, mientras no haya valores nulos.

3.8.4 Insertar una Estación y una Línea dentro de la base de datos MySQL (InsertEstacionLinea.java)

El principio de esta clase es el mismo que el de la clase ***InsertarDatos.java*** sólo cambia la dirección del método *httppost* (Solicitud para tener acceso a Internet por ejemplo una *URL*) y en los datos que se van a obtener, en este caso son la "*estación"* y la "*línea"* que haya seleccionado el usuario del dispositivo móvil, tal como se muestra en el diagrama de flujo de la Figura 19.

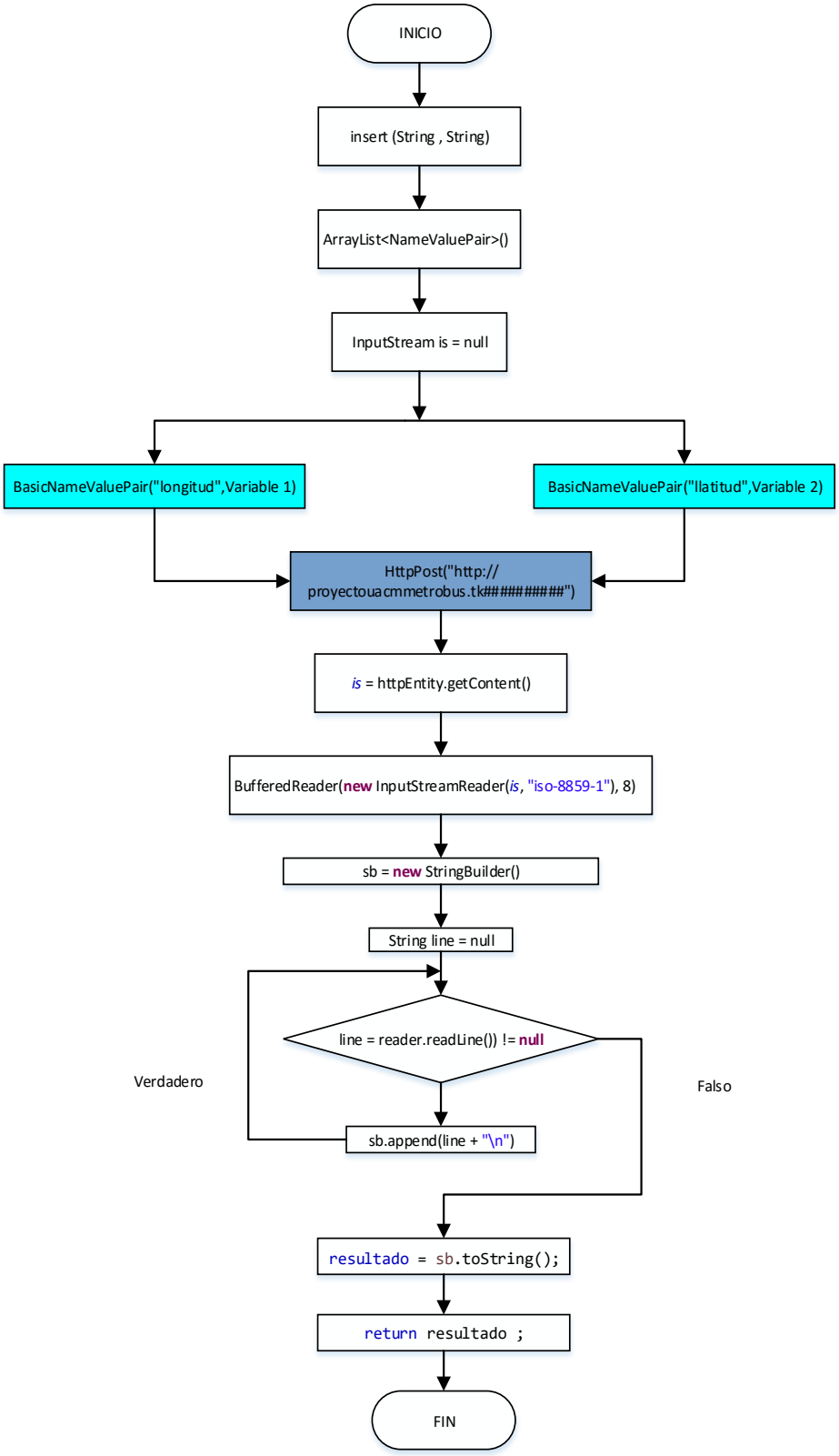


Figura 18: Diagrama de flujo de la clase InsertarDatos.java

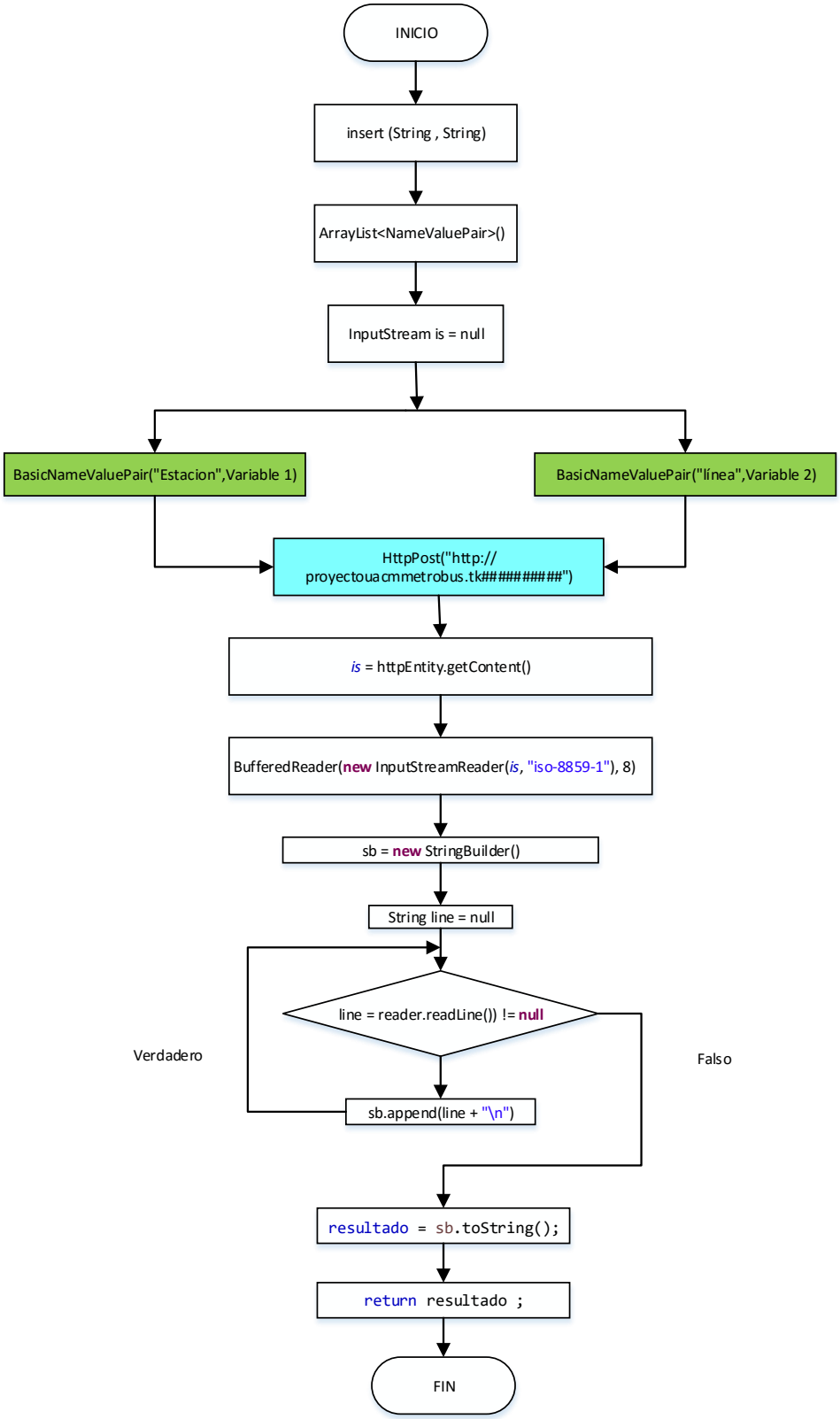


Figura 19: Diagrama de flujo de la clase InsertEstacionLinea.java

3.8.5 Actividad principal en Android (MainActivity.java).

En todo desarrollo en java es necesario tener una clase principal, es la encargada de ejecutar las demás clases si se cuenta con más de una.

A continuación en el Pseudocódigo 4 se explica la clase principal (*MainActivity.java*) para la aplicación que realiza una consulta a un servidor externo y posteriormente se hará el análisis de la aplicación en forma local.

Programa **MainActivity.java**

1. Entorno *gps = obtener posición del dispositivo.*
2. *Thread = crear una tarea asíncrona y repetitiva.*
3. Llamar a las clases *InsertarDatos.java y GPSTrasker.java.*
4. Ejecutar un thread cada minuto y llamar a la función *insert* de la clase *InsertarDatos.java.*
5. Entorno Sting URL.
6. Llamar a la clase JSONParser.java.
7. Crear un objeto *ProgressDialog.*
8. Para (i =0 , Hasta que se tenga el tamaño de la tabla Estaciones del servidor externo, aumenta 1){
    - Guardar objetos del tipo Hashmap (objeto, identificador) = mapa.
    - Entorno *String* (Estación y Línea).
    - Recibir *String* de la selección de mapa.
    - Llamar a la función *insert* (selección del mapa) = x.
    - Mostrar mediante un objeto *Toast* "Hay x cantidad de usuarios en la estación".
    - Llamar mensajes de alerta de GPS si no está disponible.

    } Cerrar el ciclo *for*.
9. Llamar al Thread.

Pseudocódigo 4.

En el punto número 4 del Pseudocódigo 4 de la clase *MainActivity.java* se puede observar que cada determinado tiempo se ingresa un registro dentro de la base de datos *MySQL* por medio de un *Thread*, el cual entra dentro de un ciclo ***for*** y se ejecuta cada minuto 200 veces. Eso quiere decir

que la aplicación cada minuto enviará a la base de datos la longitud y latitud del usuario. A continuación se muestra el Código 5 correspondiente al método *miThread* localizado en la clase principal "*MainActivity.java*" que contiene el *Thread,* con la finalidad que haya un mejor entendimiento.

El *Thread* o *hilo* es un proceso secuencial que puede llevar a cabo varias tareas al mismo tiempo, se le pueden asignar tiempos de ejecución o prioridades [8].

Para el caso de la aplicación sólo lleva a cabo una sola tarea la cual consiste en obtener del *GPS* la posición del usuario y llamar a la función "insert" de la clase "InsertarDatos.java", la cual se encarga de enviar al servidor externo el valor en tipo String de la posición del usuario.

Lo anterior se hace con la finalidad de que el usuario sea agregado a la base de datos cada vez que pasa un minuto y la aplicación ingresará a la base de datos, la estación, la línea, la hora y la fecha actual. Como ya se comentó previamente la distancia que se ha escogido para poder realizar las pruebas necesarias para la aplicación y determinar la cercanía de una estación es de 450 (m), pero una vez que la aplicación sea liberada al mercado la condición de distancia mínima deberá de ser reducida a 50 (m) debido a que la longitud de una estación de metrobús es aproximadamente 50 (m).

Se llama al objeto *ArrayList* que se almacenará dentro de la variable *"oslis",* es aquí donde se almacenarán los datos deseados de la base de datos *MySQL*. También se llama al archivo **consultaDataBase.php** el cual ya se encuentra en un servidor.

Se realiza una tarea asíncrona en un *Thread* nuevo, esto se hace para que la tarea no sea bloqueada por otro proceso mientras esta tarea se ejecuta.

```
final Handler handle = new Handler();

protected void miThread(){
Thread t = new Thread(){
public void run(){
try{
for (int i=0 ; i<= 200;i++){// ingresara 200 registros

datos.insert(String.valueOf(gps.getLongitude()),
String.valueOf(gps.getLatitude()));
System.out.println("Dato ingresado a la Base de Datos: "
+gps.getLatitude() +" y " + gps.getLongitude());

Thread.sleep(60*1000);
}
}catch(InterruptedException e){
	e.printStackTrace();
}
	handle.post(proceso);
}
};
	t.start();
}

final Runnable proceso = new Runnable(){
	public void run(){


}
};
```

Código 5: Thread en MainActivity.java.

También se asignan los objetos que verá el usuario (dos *TextView)*, de igual forma se mostrará un objeto de tipo *ProgressDialog* (Dialogo de progreso) con la finalidad de que el usuario visualice que la aplicación se está ejecutando y debe de esperar a que realice la consulta a la base de datos.

Una vez que se obtuvo la selección del usuario, se realiza una consulta al servidor y se obtiene una nueva variable "*k*", que es la que contiene la cantidad de personas que han ingresado en la base de datos y que cumplen con los criterios previamente establecidos en el archivo ***Peticion.php.***

El usuario recibe en pantalla el resultado de la consulta realizada al servidor, obteniendo así, la cantidad de usuarios que están usando la aplicación y se encuentran en la estación que ha seleccionado.

En el Código 6 se muestra el algoritmo creado para desplegar en la pantalla del dispositivo, los resultados generados con base a la petición del usuario de la aplicación.

```
Toast.makeText(MainActivity.this,"\t\t\t\tHay "+ k +"usuarios en la estación\n "
+oslist.get(+position).get("Estacion") , Toast.LENGTH_SHORT).show());
```

Código 6.

3.9 Desarrollo de la aplicación con funciones locales

En un principio se propuso que los cálculos que requería la aplicación se realizaran localmente, pero finalmente se decidió que, la aplicación debería obtener los datos de un servidor externo; dentro de las razones de cambio están, que la aplicación requería de muchos recursos, se acababa rápido la batería y que dependiendo del dispositivo, podía tardar hasta seis veces más de lo que tarda el servidor externo.

Aunque esta idea de tener localmente la base de datos era buena no cumplía con uno de los objetivos de la aplicación que es la obtención de datos rápido. No obstante se pudo corregir ese problema y reducir los tiempos de ejecución de la aplicación.

Como la propuesta de realizar los cálculos localmente fue parte fundamental para el desarrollo de la aplicación, en este apartado se abordará el desarrollo de la aplicación en el caso de realizar los cálculos localmente. Se explica a continuación con la finalidad de documentar el trabajo realizado y en su momento se pueda aplicar esta solución.

Para iniciar hay que recordar que la información que se obtiene del servidor se encuentra codificada en *JSON*; para que Android pueda trabajar con ella sin problemas, es necesario decodificarla. Se

tuvieron que implementar dos clases que realizan la decodificación, una de ellas decodifica las longitudes de las estaciones y otra las latitudes.

JSON (JavaScript Objet Notation) es un intercambiador de datos muy ligero y su formato de texto es completamente independiente pero muy utilizado por programadores de lenguajes de la familia de ***C*** [13].

Un ejemplo de una codificación en JSON es la siguiente:

{"Objeto": [{"Nombre a_1 ": "Valor a_2", "Nombre b_1": "Valor b_2" …. "Nombre m_n": "Valor m_n"}, {"Nombre $2a_1$ ": "Valor $2a_2$", "Nombre $2b_1$": "Valor $2b_2$" …. "Nombre $2m_n$": "Valor $2m_n$"}, {… … …}]}

Como Android sólo puede realizar operaciones con *Arreglos* en un formato normal de *Arreglo* de java, es necesario realizar una decodificación para poder trabajar con los datos que se obtuvieron del servidor en formato *JSON.* En la sección 3.8.1 se muestra cómo se realizó la decodificación de los archivos en formato *JSON*.

3.9.1 Cálculo de distancias localmente en Android

Android cuenta con un método propio con el cual se pueden realizar cálculos de distancias geográficas y se encuentra dentro del paquete de Android o API "*Android.location.Location".* Recibe cuatro parámetros de entrada que son la longitud inicial, la latitud inicial así como la longitud final y latitud final. Tiene como ventaja que no hay que crear un método especial para poder realizar cálculos de distancias y regresa un resultado de tipo *flotante* o *float*.

La Figura 20 muestra el diagrama de flujo del método que se utilizó para realizar los cálculos de distancias localmente.

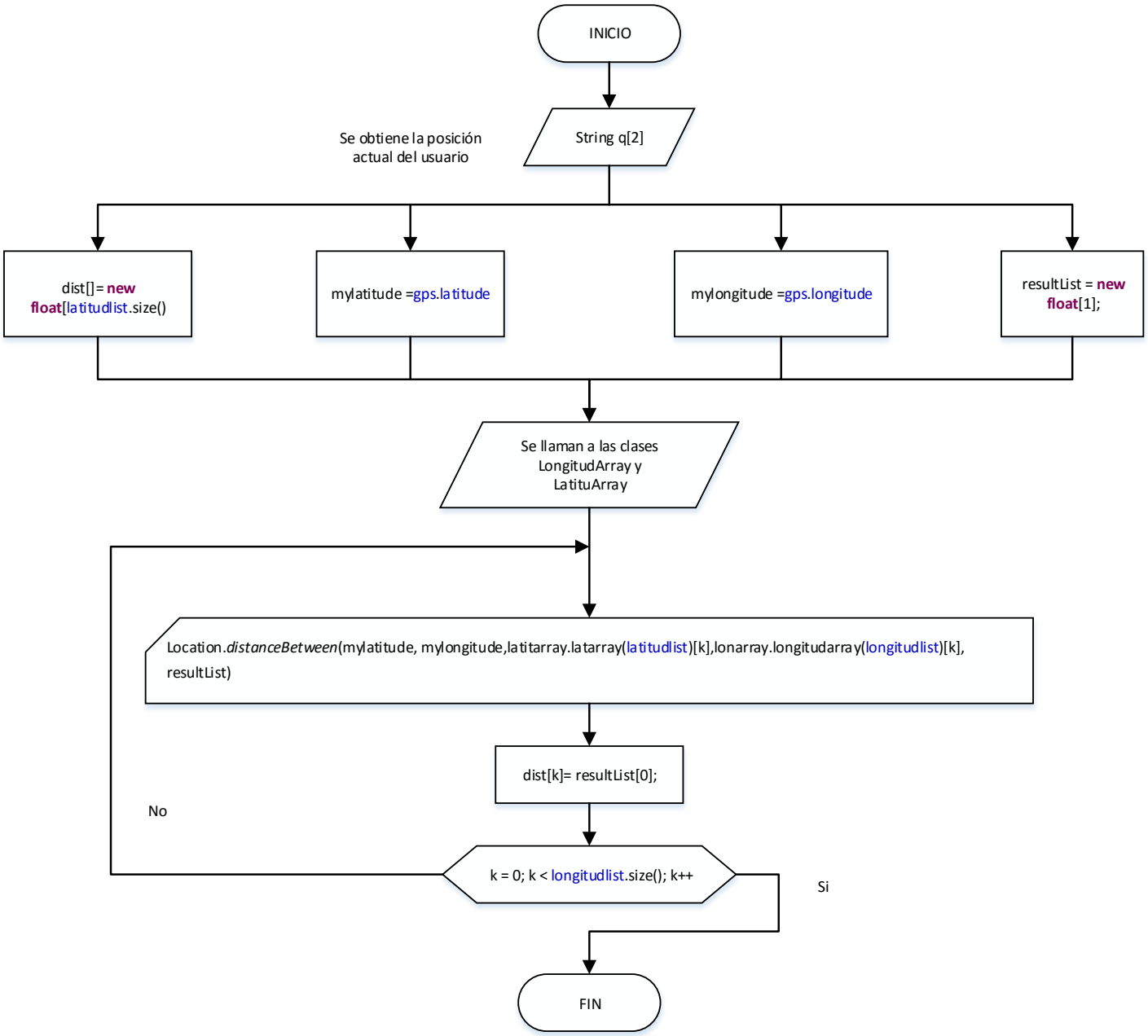


Figura 20: Diagrama de flujo del método para calcular un Arreglo de distancias usando el método propio de Android.

En este caso la función propia de Android ***Location.distanceBetween()[13],*** se encarga de realizar los cálculos para obtener la distancia en metros que hay entre el usuario y un punto dado. La función que se encuentra dentro de un ciclo ***for***, la cual se ejecuta hasta que se haya realizado la cantidad de iteraciones correspondientes al tamaño del arreglo de posiciones de las estaciones obtenidas del servidor externo, después se sustituyen las llaves "{" y "}" por espacios vacíos y finalmente se guardan en un nuevo vector en un formato de arreglo normal de java, con la finalidad de que se puedan realizar cálculos matemáticos con ellos.

Hay que tomar en cuenta que siempre se tendrá la misma cantidad de elementos dentro del objeto *ArrayList* para la longitud y la latitud, es por eso que sólo se toma en cuenta el tamaño de uno de ellos para determinar la cantidad de elementos que contiene. Los resultados que se van obteniendo ya en un formato de Arreglo se almacenan dentro de la variable ***dist[]*** y son del tipo ***flotante****.*

3.9.2 Conversión del Arreglo de latitudes en formato JSON a un Arreglo de java (LatitudArray.java)

Se obtienen todos los registros de la latitud los cuales son llamados desde la clase ***MainActivity.java*** y están almacenados dentro de la variable *latitudlist,* el tipo de datos se cambian a *double,* ya que los datos obtenidos son de tipo *String*, se reemplazan los "{ }" por espacios vacíos[14], de esta forma se obtiene un *arreglo* de latitudes tal como se muestra en la Figura 21.

---

[13] *La forma de utilizar la función es: distanceBetween (**double** Latitud inicial, **double** Longitud inicial, **double** Latitud final , **double** Longitud final, **float[]** resultado) [14].*

[14] La función en java para reemplazar valores es **remplace**(*valor actual*, *valor a reemplazar*).

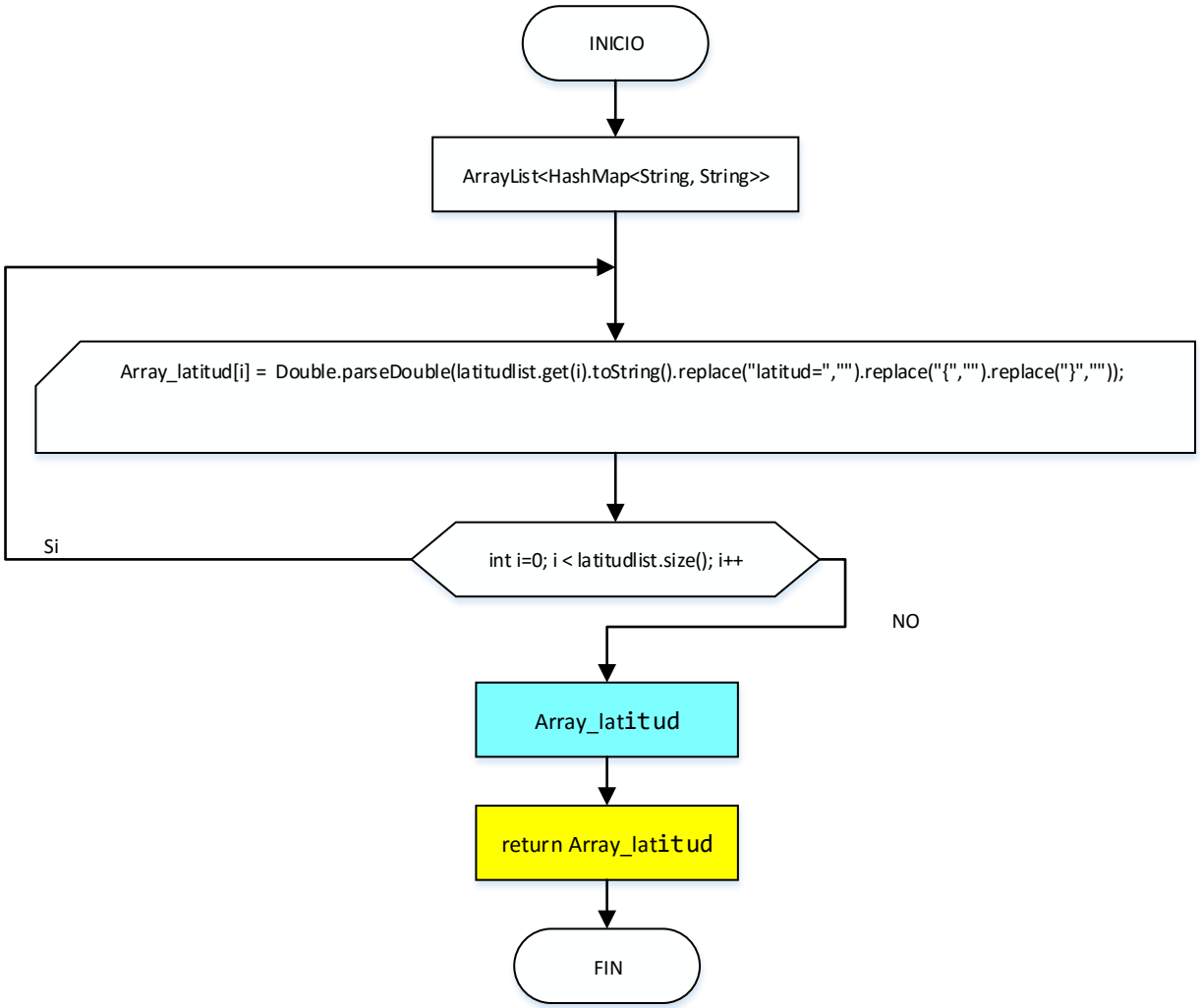


Figura 21: Diagrama de flujo de la clase LatitudArray.

3.9.3 Conversión del Arreglo de longitudes en formato JSON a un Arreglo de java (LongitudArray.java)

Los criterios son los mismos que en la clase ***LatitudArray.java,*** sólo que en este caso se obtienen todos los valores de las longitudes obtenidos de la variable *longitulist* y se obtiene un *arreglo* de longitudes.

En la Figura 22 se muestra la clase *LongitudArray.java*, la cual convierte un arreglo con codificación JSON a un arreglo de java.

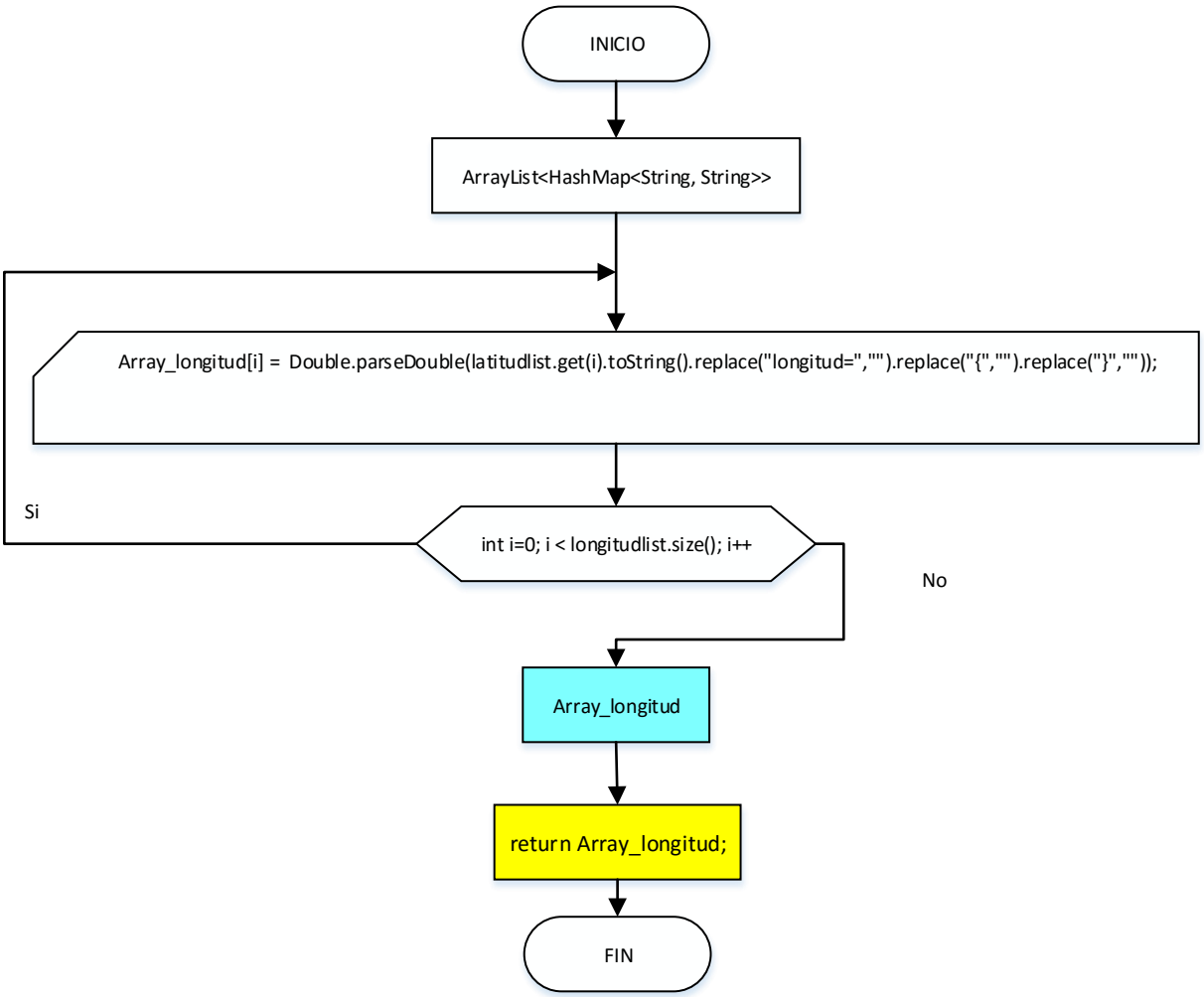


Figura 22: Diagrama de flujo de la clase LongitudArray.

3.9.4 MainActivity.java en forma local

Se creó un objeto del tipo ***ArrayList*** para poder trabajar la longitud y la latitud independientemente y se asigna el objeto que adquiere la longitud y la latitud. Citando el Pseudocódigo de la p. (57), dentro del ciclo ***for*** de la línea 8, se agregan las siguientes instrucciones en el Pseudocódigo 5.

Programa ***Cálculos locales en MainActivity.java***

- *Para (….){*Entorno Hashmap (nombre, clave) = listado de posiciones,
- }Cerrar el ciclo for
- Llamar a las clases *LongitudArray*.java y *LatitudArray.java*
- Llamar a la funciones Array_Longitud y Array_Latitud de las clases LongitudArray.java y LatitudArray.java.
- Obtener el valor más pequeño dentro del Arreglo = distmin.
- Si(distmin < 450){Conectarse al archivo Peticion.php}.

Pseudocódigo 5.

3.9.5 Cálculo de valor mínimo en un Arreglo y envío al servidor externo

Cuando ya se obtuvieron todos los elementos del *arreglo* se debe de calcular el valor mínimo dentro del *Arreglo* de distancias, esto dará como resultado la estación más cercana al usuario. Para determinar el valor mínimo dentro de un *arreglo* se usó un ciclo ***for*** que obtiene uno a uno los elementos dentro del *Arreglo* y a su vez realiza un comparativo dentro de un condicional ***if*** entre todos ellos para determinar el valor menor dentro del *Arreglo*.

La Figura 23 muestra el diagrama de flujo para calcular el valor mínimo de un arreglo y la inserción de un dato dentro de la base de datos.

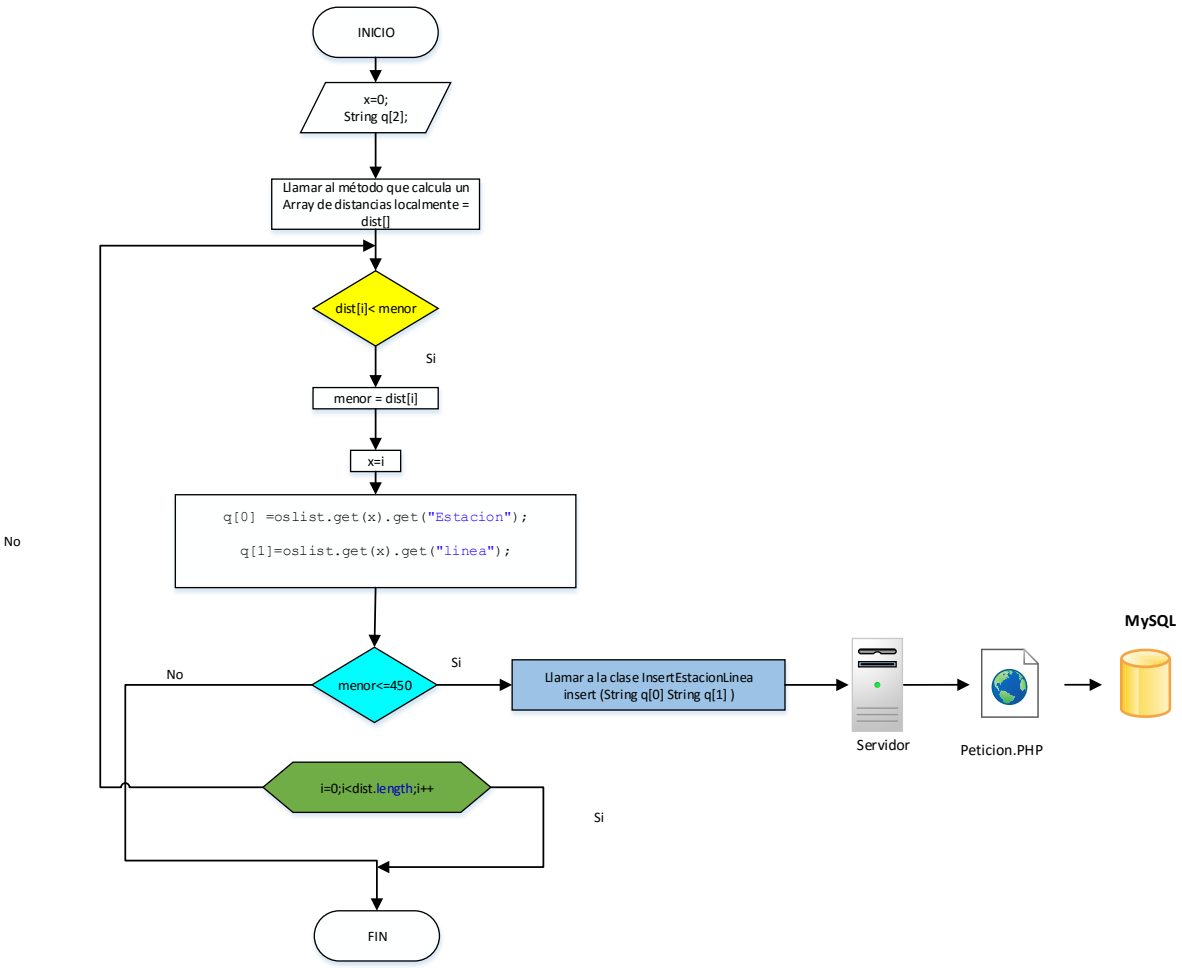


Figura 23: Diagrama de flujo para determinar un valor mínimo dentro de un Arreglo.

Al igual que el archivo ***MinDistancia.php*****,** se asignó arbitrariamente un valor de 450 (m) para que se cumpla la condición de distancia mínima para poder ingresar datos a la base de datos *MySQL*, se utilizó el condicional ***if*** para determinar la cercanía con una estación.

Del objeto *ArrayList* de estaciones y líneas se deben de obtener los datos de consulta que requiere el usuario, es por eso que se asigna a una variable *q[2]* de tipo *String*, es decir, un *Arreglo* de dimensión dos de tal forma que a *q[0]* se le asigna la estación seleccionada y a *q[1]* la línea correspondiente a la estación.

Después sólo se requiere hacer la petición al servidor mediante el archivo ***Peticion.php*** para que realice la consulta a la base de datos.

4.1 Resultados

En este capítulo se muestran las pruebas y resultados obtenidos por la aplicación remotamente y localmente.

Con base a las dos implementaciones que se realizaron, se muestra a continuación un ejercicio hecho con los dos métodos o formas propuestas; se obtuvieron los siguientes resultados:

Para la posición: latitud = 19.4142854 y longitud = -99.1658623.

4.1.1 Resultados locales

Por medio de la instrucción de java "**System.*out*.println()**" (Como se muestra en el Código 7), la cual se imprime el en ***LogCat*** de Android el *arreglo* de distancias obtenido para el ejercicio.

```
System.out.println("Array de distancias:=
"+Arrays.toString(dist) );
```

Código 7.

A continuación se muestra el listado completo del arreglo de distancias obtenidas para el ejercicio:

04-23 10:11:22.271: I/System.out(16698): Array de distancias:= [10327.274, 9071.632, 8553.544, 7739.7915, 6671.7183,5838.5576, 5421.1064, 5001.389, 3833.9949, 3391.422, 3077.372, 2566.586, 2177.343, 1543.8793, 1056.4286,1117.1677, 642.76886, 243.9852, ***137.80008****, 551.5448, 928.99384, 1424.6658, 1902.498, 2438.352, 2855.9639, 3316.0068, 3694.3533, 4023.2385, 4652.067, 5274.159, 5721.8037, 6093.1445, 6471.5444, 6912.1895, 7354.7363, 7824.8296, 8581.406, 10387.445, 11270.773, 12388.316, 12937.625, 13405.633, 13528.784, 14000.59, 14537.642, 14837.167, 14922.619, 2611.376, 2351.65, 2542.0623, 1996.1304, 1547.8833, 1377.9719, 1287.3527, 1455.5232, 1938.9178, 2339.4607, 2388.7546, 2907.6584, 3235.4954, 3476.26, 4034.3538, 4218.624, 4691.9595, 5453.615, 6202.653, 7182.577, 6777.8203, 7599.0713, 8240.322, 8640.034, 9121.523, 9509.553, 9965.647, 10062.901, 10508.822, 11126.76, 11556.715, 11481.835, 11628.46312237.303, 12407.926, 12726.425, 12736.265, 12072.011, 11588.642, 11161.543, 10418.771, 9878.194, 9489.641, 9063.42, 8671.055, 8393.372, 7783.8076, 7435.269, 6868.392, 6613.933, 6336.786, 6339.752, 5901.709, 5520.401, 5108.8945, 4686.535,3962.0505, 3769.801, 3376.2903, 3079.1348, 2684.2646, 2315.8977, 1701.1995, 1385.802, 1186.1364, 1203.4613, 1395.7242, 1784.8972, 2288.7507, 3711.2217, 3443.0105, 3040.477, 3134.6956, 3270.8865, 3512.434, 3865.5405, 4107.4805, 4412.065, 4684.3706, 5019.9204, 5260.202, 5467.8022, 5818.5137, 5588.856, 8979.641, 9368.27, 5471.155, 5479.6387, 5023.6196, 4566.3423, 4411.9424, 4494.66, 4159.5957, 3877.3608, 3515.16, 3294.2747, 3090.0264, 2867.6086, 2677.5005, 2488.3145, 2629.5913, 2568.5227, 2869.2769, 3041.1653, 3399.7065, 3711.2217, 3443.0105, 3040.477, 2787.8096, 2530.4304, 2633.9363, 2534.3494, 2641.3909, 2812.6401, 3100.0645, 3279.675, 3480.2832, 3814.015, 3796.4453, 3965.6646, 4375.831, 4496.71, 4980.302, 5471.155, 5861.9966, 5626.616, 5850.147, 5515.5083, 5291.848, 5050.559, 4907.2373, 4432.387, 4131.725, 3835.6233, 3572.2637, 3299.083, 3071.7043, 3041.1653, 3399.7065, 5713.7827,*

6031.6113, 6307.182, 6686.382, 7050.493, 7475.368, 7910.287, 8412.685, 8915.938, 9366.735, 9868.882, 10358.744, 10760.391, 11121.954, 11692.311, 12159.584, 12692.352, 13324.367]

Posteriormente la aplicación calcula el valor mínimo dentro del *Arreglo* de distancias y se obtuvo la relación de 18 distancias más cercanas al usuario.

04-23 10:11:22.271: I/System.out(16698): Posición del Array 1 distancia 9071.632

04-23 10:11:22.271: I/System.out(16698): Posición del Array 2 distancia 8553.544

04-23 10:11:22.271: I/System.out(16698): Posición del Array 3 distancia 7739.7915

04-23 10:11:22.271: I/System.out(16698): Posición del Array 4 distancia 6671.7183

04-23 10:11:22.271: I/System.out(16698): Posición del Array 5 distancia 5838.5576

04-23 10:11:22.271: I/System.out(16698): Posición del Array 6 distancia 5421.1064

04-23 10:11:22.271: I/System.out(16698): Posición del Array 7 distancia 5001.389

04-2310:11:22.271:I/System.out(16698): Posición del Array 8 distancia 3833.994904-23 10:11:22.271: I/System.out(16698): Posición del Array 9 distancia 3391.422

04-23 10:11:22.271: I/System.out(16698): Posición del Array 10 distancia 3077.372

04-23 10:11:22.271: I/System.out(16698): Posición del Array 11 distancia 2566.586

04-23 10:11:22.271: I/System.out(16698): Posición del Array 12 distancia 2177.343

04-23 10:11:22.271: I/System.out(16698): Posición del Array 13 distancia 1543.8793

04-23 10:11:22.271: I/System.out(16698): Posición del Array 14 distancia 1056.4286

04-23 10:11:22.271: I/System.out(16698): Posición del Array 16 distancia 642.76886

04-23 10:11:22.271: I/System.out(16698): Posición del Array 17 distancia 243.9852

*04-23 10:11:22.271: I/System.out(16698): Posición del Array **18** distancia **137.80008***

*04-23 10:11:22.271: I/System.out(16698): **Estación más cercana obtenida localmente es: = [Sonora***

*04-23 10:11:22.271: I/System.out(16698): **Línea 1**]*

Dando como resultado final el valor ***137.80008*** (m) y que ocupa el lugar número **18** dentro del *Arreglo* de distancias. No hay que olvidar que se debe cumplir la condición de que el usuario no esté a más de 450 (m) de distancia de una estación, que en la práctica la distancia entre el usuario y la estación no debe de ser mayor a 50 (m) para que pueda ser registrado dentro de la base de datos. Esto no significa que el usuario no pueda usar la aplicación si no se encuentra a 50 (m) o más de una estación de metrobús: el usuario podrá verificar el estado de cada estación independientemente del lugar en el que se encuentre.

También se requiere el lugar que tiene la distancia mínima dentro del *Arreglo* porque de esta forma se puede obtener la estación y línea a que corresponde. A continuación se muestran los resultados visualizados en la plataforma *LogCat* de Android.

*04-23 10:11:22.271: I/System.out(16698): Estación más cercana obtenida localmente es: = [**Sonora***

*04-23 10:11:22.271: I/System.out(16698): , **Línea 1**]*

Es necesario mencionar que el *Arreglo* o vector de posiciones será distinto para cada usuario, esto dependerá en gran medida de la ubicación geográfica de cada usuario.

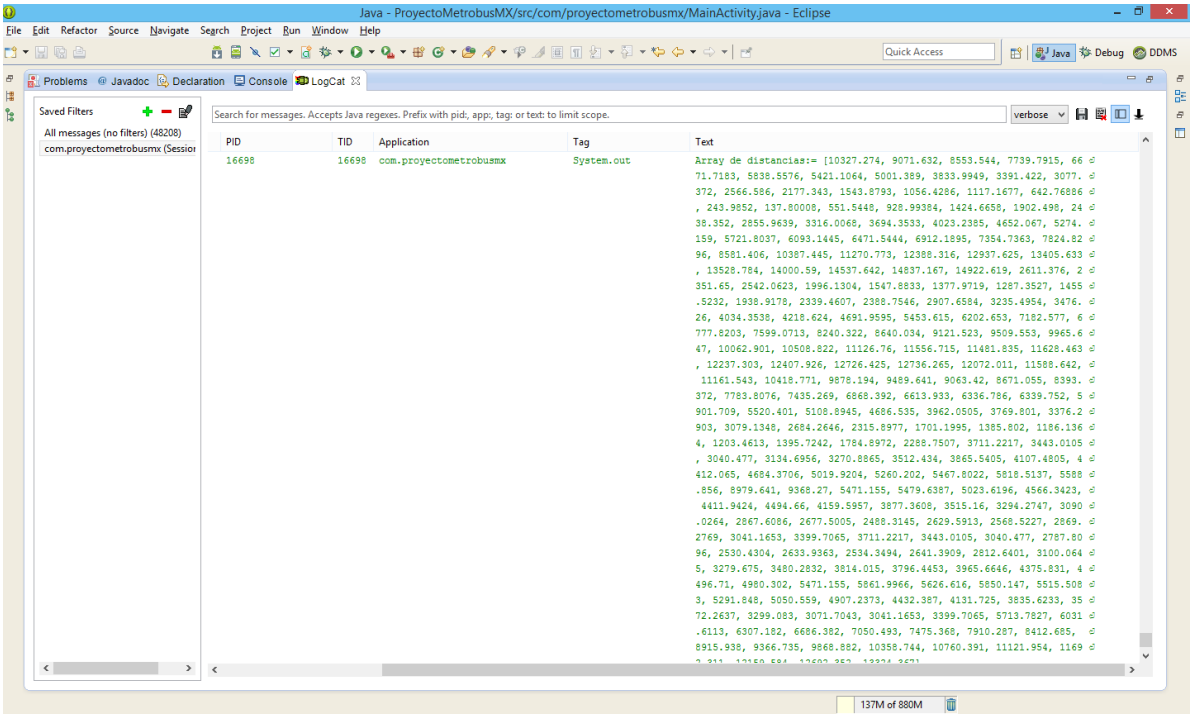


Figura 24: Vista de LogCat de Android del arreglo de distancias de estaciones.

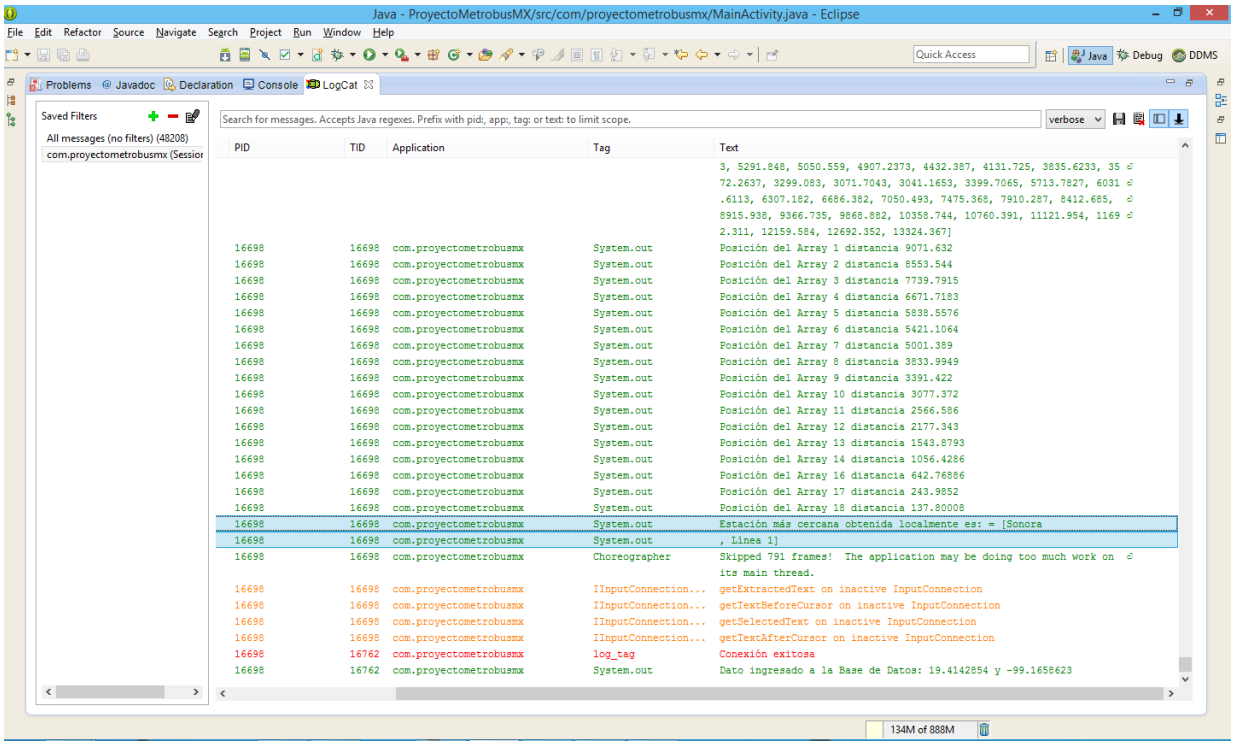


Figura 25: Vista en LogCat para estación cercana.

4.1.2 Resultados obtenidos desde un servidor.

Análogamente también fue necesario obtener un *Arreglo* de distancias remotamente entre el usuario y cada una de las estaciones. El encargado de esta tarea fue el archivo ***MinDistancia.php*.** A continuación se muestran los resultados obtenidos mediante la consulta al archivo *PHP* dentro del servidor.

Para comprobar los resultados se asegurará que los parámetros de entrada longitud y latitud sean los mismos que en la versión local, es decir latitud = 19.4142854 y longitud = -99.1658623. En la Figura 26 se muestran los resultados obtenidos desde un explorador de Internet para el arreglo de distancias.

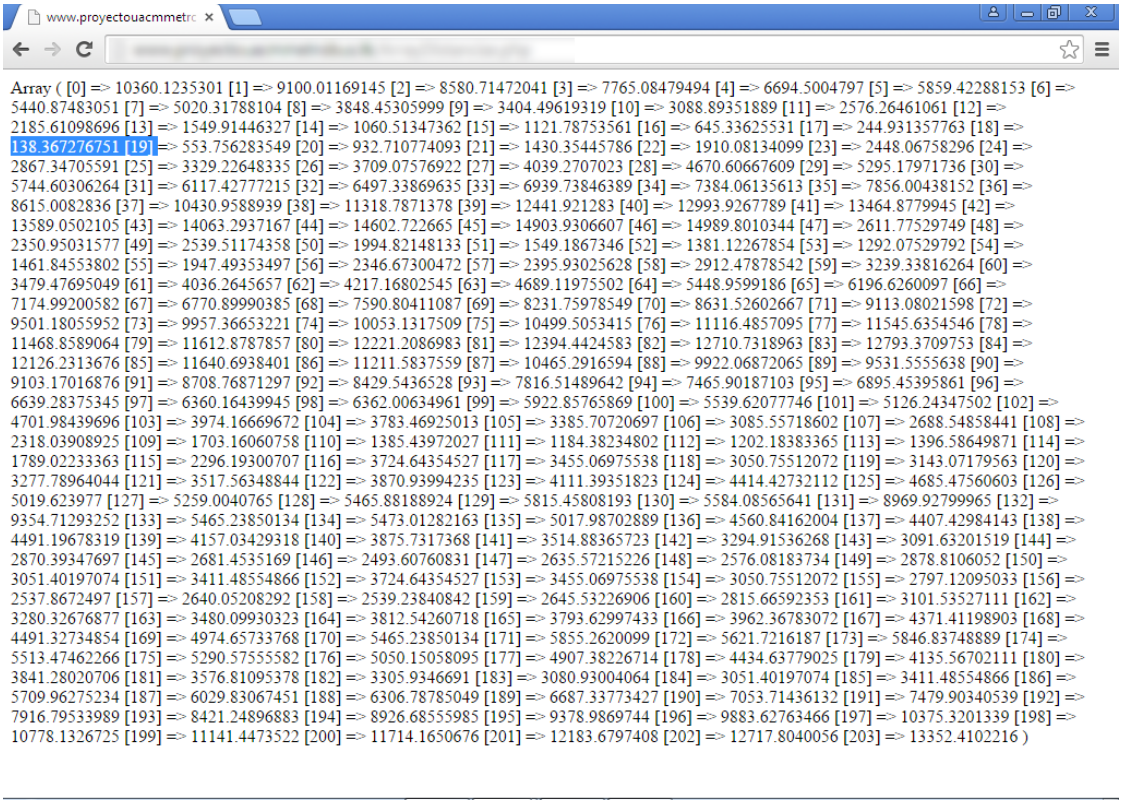


Figura 26: Vista del Arreglo de distancias desde el servidor.

Se debe notar algo muy importante, los resultados obtenidos localmente en el dispositivo Android y mediante el servidor externo son muy semejantes, por ejemplo, el primer valor del *Arreglo* local es de ***137.80008*** **(m)** y el valor obtenido del servidor es de **138.367276751 (m)**. La diferencia es de 0.56 (m) por lo cual el uso de cualquiera de los dos métodos es válido. La diferencia que se obtiene entre un método y el otro es debido al parámetro que se define como ***radio de la tierra,*** que como se mencionó anteriormente es una aproximación al radio de una esfera; hay que recordar que la tierra no tiene una forma totalmente esférica.

Android cuenta con una función propia para cálculos de distancias geográficas, dicha función es: ***distanceBetween(),*** y ya tiene predeterminado el ***radio de la tierra***, es debido a eso, que hay una diferencia entre los resultados obtenidos localmente como los obtenidos desde el servidor.

El siguiente resultado tiene que hacer referencia al valor que ocupa dentro del *Arreglo* la estación que se encuentra más cerca del usuario, para ello se definió la variable $clave y se implementó de la siguiente forma.

```
$clave  = array_search(min($array), $array)+1;
```

Instrucción en php para obtener el valor más pequeño dentro de un Arreglo.

Recordemos que, para encontrar la posición adecuada de un elemento dentro de un *Arreglo* hay que sumarle al índice del elemento un 1, ya que los *Arreglos* comienzan en cero y el id_estacion en uno.

Así se puede obtener el resultado numérico de la estación, que en este caso corresponde al valor **19** dentro del Arreglo, es decir la estación "***Sonora de la línea 1***", siendo para este caso en particular la posición ***19*** dentro del *Arreglo*.

Una vez que el servidor ha calculado las distancias a todas las estaciones y se ha determinado la estación más cercana al dispositivo móvil mediante la instrucción en *PHP* "*array_search(min($array)",* se debe cumplir la condición de distancia mínima a una estación y posteriormente el archivo ***MinDistancia.php*"** , ubicado en el servidor, realizará a su vez una inserción a la tabla *DatosUsuarios* de los valores mencionados a continuación:

- *id_usuario*. Un registro de identificación a cada usuario.
- *Estación.* La estación en la que el dispositivo móvil cumple con la condición de distancia mínima.
- *línea.* La línea a la que pertenece la estación cercana al dispositivo móvil.
- *fecha.* Fecha de inserción de los datos a la tabla *DatosUsuarios.*
- *hora.* La hora de la inserción de los datos en formato de 24 horas.
- *IP.* La dirección *IP* del dispositivo móvil que se conecta a la base de datos.

4.1.3 Interfaz de usuario

En este apartado se abordará la interfaz visual que tiene la aplicación para los usuarios de la aplicación.

La aplicación fue pensada para que los usuarios que no son expertos utilizando Smartphones tengan la posibilidad de realizar la consulta sin mayor problema.

El usuario sólo requiere instalar la aplicación y posteriormente presionar el ícono de la misma.

En la Figura 27 se muestra el ícono diseñado para la aplicación.

Figura 27: Ícono de la aplicación “Asistencia Metrobús”

Una vez que el usuario ha presionado el ícono la aplicación se conectará a Internet para obtener el listado completo de estaciones. En las Figuras 28 y 29 se muestra cómo actúa la aplicación.

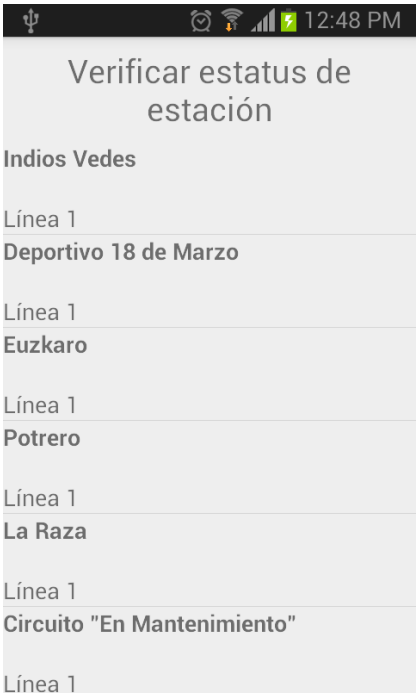


Figura 28: Vista de la aplicación al presionar el ícono.

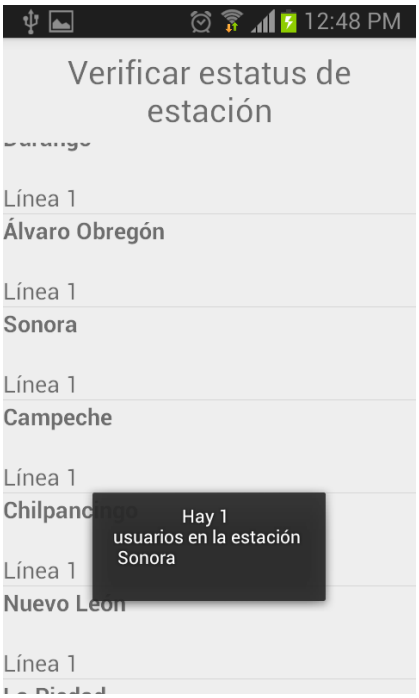


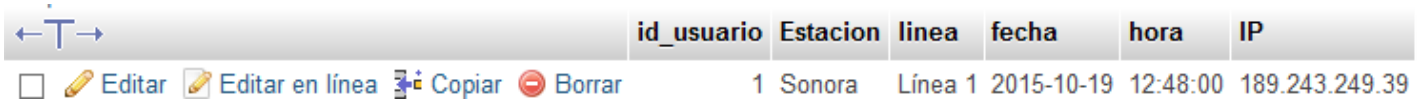
Figura 29: Vista de la aplicación cuando el usuario presiona una estación. En este caso se presionó la estación Sonora de la Línea 1 y hay un registro en la base de datos que cumple con las condiciones de fecha, hora, estación y línea.

La aplicación sólo ingresa a la base de datos a aquellos usuarios que cumplan con tres condiciones:

- Usuarios que estén ejecutando la aplicación.
- Usuarios que no excedan de 200 (min) en tiempo de ejecución de la aplicación, debido a que la aplicación ingresa a la base de datos cada minuto la posición del usuario hasta llegar a un total de 200 registros por usuario cada vez que ejecute la aplicación.
- Usuarios que se encuentren cerca de una estación (a menos de 450(m)).

Si el usuario no cumple dichas condiciones, podrá seguir realizando consultas, pero no podrá ingresar su posición a la base de datos.

En la Figura 30 se muestra una inserción dentro de la base de datos en el servidor externo.



	id_usuario	Estacion	linea	fecha	hora	IP
Editar Editar en línea Copiar Borrar	1	Sonora	Línea 1	2015-10-19	12:48:00	189.243.249.39

Figura 30: Vista de la base de datos ***DatosUsuarios*** *en MySQL.*

Se muestra un registro en la base de datos, con la finalidad de garantizar que hay una correcta inserción de los datos en la tabla *DatosUsuarios,* en las consultas mostradas anteriormente. El registro en la base de datos corresponde a la estación "*Sonora"* de la "*línea 1"* del metrobús.

El registro de esta estación se realizó porque cumple con el requerimiento de distancia mínima entre la estación y el dispositivo móvil (450 m); cabe mencionar que hay una estación más que cumple con la condición de distancia mínima que para este caso es la estación "*Álvaro Obregón"*, pero el algoritmo sólo ingresa a la base de datos la estación que se encuentra más cercana al dispositivo móvil.

El valor de condición mínima de 450 (m) se decidió porque era más sencillo para realizar pruebas, sin la necesidad de estar dentro de una estación del metrobús, pero ya en la aplicación real este valor debe estar aproximadamente entre 50 (m) y 70 (m), para garantizar que sólo ingresarán a la base de datos aquellos usuarios que realmente estén dentro de alguna estación del metrobús.

.

5. Conclusiones

Con el desarrollo de la aplicación se logró la creación de una base de datos capaz de recibir, almacenar y realizar consultas desde un dispositivo móvil, de una forma fácil y rápida. Así mismo se logró que los usuarios del sistema de transporte metrobús puedan saber la cantidad de personas promedio que están en una determinada estación.

La aplicación muestra mayor rendimiento utilizando los servicios de *GPS Asistido*, porque no hay que esperar el *TTF (Time to First Fix)* de los satélites, de igual manera, que los dispositivos móviles conservan mayor tiempo la energía de su batería.

Se utilizaron tecnologías con las que la mayor parte de los usuarios de dispositivos móviles cuentan, de esta forma, no fue necesario que los usuarios adquieran algún dispositivo con características especiales para poder ejecutar la aplicación. La aplicación puede ejecutarse en cualquier dispositivo móvil *Smartphone* con SO Android que cuente con receptor de *GPS* y acceso a Internet. Cabe mencionar que la aplicación se realizó de tal forma que el acceso al contenido a la base de datos fuese lo más rápido posible, logrando así, reducir significativamente el tiempo de espera, para que los usuarios obtengan el resultado que buscan.

La aplicación puede ser implementada de tal forma que sea más dinámica, es decir, que identifique las horas en las que el tiempo de espera es mayor o menor a un minuto para poder llegar de estación a estación, logrando esto, se obtendrán resultados más precisos. La realización de dicha implementación requeriría de más tiempo de estudio e incluso de datos estadísticos dentro del sistema de transporte metrobús que ayuden a identificar cómo se comporta el sistema en determinados días u horas. De esta forma la aplicación sería más precisa y ayudaría más a los usuarios del sistema de transporte metrobús.

Referencias

[1] Eduardo Huerta, Aldo Mangiaterra, Gustavo Noguera, Junio 2015 "*GPS Posicionamiento Satelital*", República de Argentina. 1ERA Edición, UNR Editora. p (2)

[2] Neil Harper "*Server-side GPS and Assisted-GPS in java*" January 2009", 1ST Edition, Artech House. Páginas (1) y (6)

[3] Correia Paul "*Guía práctica del GPS*" 2002, Barcelona España. Marcombo BOIXAREU Editores.

[4] Girones Jesús Tomás "*El gran libro de Android*" ,2013. Barcelona. 3ERA Edición. Marcombo.

[5] Wolfson Mike, "*Android Developer Tools Essentials"*, USA, 2013, 1ST Edition, O'Reilly.

[6] Robledo Fernández David "Desarrollo Para Aplicaciones Android II, *Aula mentor. Programación* ", España, 2014, Ministerio de Educación. Página 281

[7] Oleaga Michael (15 de Febrero de 2014). iOS vs. Android vs. Windows Phone Market share 2013: Google Smartphones OS Hits 78 Percent Globally As Apple Inc. Drops Despite Strong iPhone Sales: Estados Unidos: Latin Post: http://www.latinpost.com/

[8] Egham (13 de febrero de 2014). About Gartner: Reino unido: Gartner: http://www.gartner.com/newsroom/id/2665715

[9] McGrath Mike "*PHP & MySQL in easy steps*" 2012, United Kingdom, 1ST Edition, In Easy Steps,

[10] Cobo Ángel, Gómez Patricia, Pérez Daniel y Rocha Rocío, “ ”, España, 2005, 1ERA Edición, Díaz de Santos páginas 5 y 33.

[11] Welling Luke & Thomson Laura “*PHP and MySQL Web Development*” Australia, 2008, 2nd Edition, Addison-Wesley.

[12] Gargenta Marko & Nakamura Masumi “*Learning Android develop mobile apps using java and eclipse*”, Estados Unidos de Norte América, Enero 2014, 2ERA Edición, O’Reilly.

[13] Mewton Aaron “*MooTools Essentials The essential reference for JavaScript and Ajax Development*”, Estados Unidos, 2008, 1ERA Edición,First Press.

[14] Android developers (14 de Junio de 2014) :Android 4.0 Google Estados Unidos Android: http://developer.android.com/intl/es/reference/android/location/Location.html

[15] Santiago Arnalich Julio Urruela, "GPS, Google Earth y Cooperación, cómo crear. Compartir y colaborar con mapas en la red", Junio 2012, 1ERA Edición, arnalich water and hábitat.

Latitud: Es el arco de Meridiano contado desde el Ecuador hasta el paralelo donde se encuentra el observador.

Se mide a partir del Ecuador, de 0º a 90º, todas las latitudes correspondientes a puntos en el Hemisferio Norte se denominan Norte, y son positivas (+) y todas las del Hemisferio Sur son Sur y negativas (-).

Longitud: Es el arco de Ecuador contado desde el Primer Meridiano (Greenwich) hasta el Meridiano del Lugar.

Se cuenta de 0º a 180º. Los puntos situados a la izquierda del Primer Meridiano, mirando hacia el Polo norte, son Oeste y los que están a la derecha son Este. Los primero son negativos (-) y los segundos son positivos (+). Los puntos situados en un mismo meridiano tienen la misma longitud.

Glosario

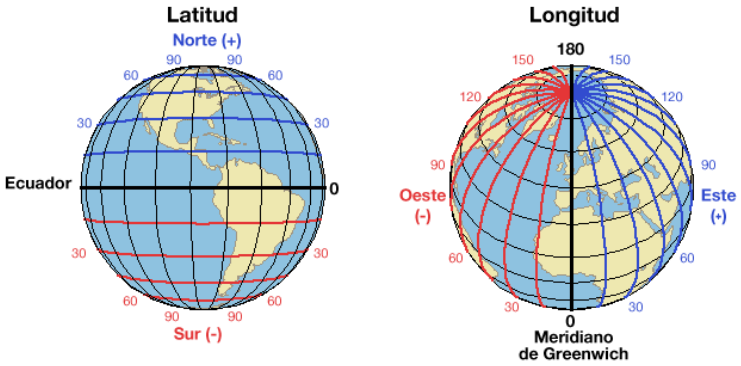


Figura 31: Imagen que muestra la longitud y latitud.

Efemérides: es el conjunto de parámetros que permite calcular la órbita de cada satélite y su posición y sus coordenadas (conocimiento preciso de la órbita de los satélites), por lo general tardan 30 (s) en descargarse [14].

SDK: Kit de desarrollo de Software (Software Development Kit), es un conjunto de herramientas para el desarrollo de software o dicho en otras palabras es la plataforma de programación.

LogCat: El sistema de registro de Android proporciona un mecanismo para la recopilación y visualización de resultados de depuración del sistema. Registros de diversas aplicaciones y partes del sistema se recogen en una serie de topes circulares, que luego pueden ser vistos y se filtra por el comando *Logcat*. Cabe mencionar que es especialmente útil para visualizar información en segundo plano es decir información que no se quiere que vea el usuario final.

DDMS: "*Dalvik Debug Monitor Server"* proporciona servicios de puerto de reenvío, capturas de pantalla de la información del dispositivo, hilo y el montón en el dispositivo, **Logcat**, el proceso y la información de estado de radio, llamadas entrantes y SMS, suplantación de datos de localización, y más [15].

ADB (*Android Debug Bridge*), es una herramienta de línea de comandos versátil que permite comunicarse con una instancia de emulador o dispositivo conectado con Android. Es un programa cliente-servidor que incluye tres componentes:

- Un cliente, que se ejecuta en el equipo de desarrollo. Puede invocar un cliente desde un shell mediante la emisión de un comando adb. Otras herramientas de Android, como el plugin ADT y DDMS también crean clientes adb.
- Un servidor, que se ejecuta como un proceso en segundo plano en el equipo de desarrollo. El servidor gestiona la comunicación entre el cliente y el daemon adb se ejecuta en un emulador o dispositivo.

Daemon en una tarea que se ejecuta como un proceso en segundo plano en cada emulador o dispositivo instancia.

Anexo

Posición geográfica de las estaciones del metrobús proporcionadas por la oficina de información pública del metrobús el día 27 de Agosto de 2014 en la tabla Estaciones dentro de la base de datos en MySQL.

Cabe mencionar que los datos tuvieron que ser convertidos a decimal para poder trabajar con ellos.

INSERT INTO `Estaciones` VALUES(1, 'Indios Vedes\r\n', '19.496689\r\n', '-99.11975\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(2, 'Deportivo 18 de Marzo\r\n', '19.486258\r\n', '-99.12455\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(3, 'Euzkaro\r\n', '19.4825\r\n', '-99.1276\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(4, 'Potrero\r\n', '19.476611\r\n', '-99.132458\r\n', 'Línea 1');INSERT INTO `Estaciones` VALUES(5, 'La Raza\r\n', '19.468833\r\n', '-99.138842\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(6, 'Circuito\r\n', '19.462789\r\n', '-99.144022\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(7, 'San Simón\r\n', '19.459644\r\n', '-99.1464\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(8, 'Manuel González\r\n', '19.456692\r\n', '-99.149431\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(9, 'Buenavista', '19.44675\r\n', '-99.153142\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(10, 'El Chopo\r\n', '19.443231\r\n', '-99.155281\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(11, 'Revolución\r\n', '19.440275\r\n', '-99.155461\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(12, 'Plaza de la República\r\n', '19.436017\r\n', '-99.157344\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(13, 'Reforma\r\n', '19.432767\r\n', '-99.158767\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(14, 'Hamburgo\r\n', '19.427528\r\n', '-99.16125\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(15, 'Insurgentes Oriente\r\n', '19.423311\r\n', '-99.162594\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(16, 'Insurgentes Poniente\r\n', '19.424067\r\n', '-99.163244\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(17, 'Durango\r\n', '19.419842\r\n', '-99.164086\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(18, 'Álvaro Obregón\r\n', '19.416372\r\n', '-99.165114\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(19, 'Sonora\r\n', '19.413081\r\n', '-99.166194\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(20, 'Campeche\r\n', '19.409511\r\n', '-99.167364\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(21, 'Chilpancingo\r\n', '19.40625\r\n', '-99.168414\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(22, 'Nuevo León\r\n', '19.401972\r\n', '-99.169808\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(23, 'La Piedad\r\n', '19.397853\r\n', '-99.171169\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(24, 'Poliforum\r\n', '19.393228\r\n', '-99.172675\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(25, 'Nápoles\r\n', '19.389619\r\n', '-99.173833\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(26, 'Colonia del Valle\r\n', '19.385644\r\n', '-99.175111\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(27, 'Ciudad de los Deportes\r\n', '19.382381\r\n', '-99.176183\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(28, 'Parque Hundido\r\n', '19.379542\r\n', '-99.177106\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(29, 'Félix Cuevas\r\n', '19.374111\r\n', '-99.178861\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(30, 'Río Churubusco\r\n', '19.368739\r\n', '-99.1806\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(31, 'Teatro Insurgentes\r\n', '19.364878\r\n', '-99.181867\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(32, 'José María Velasco\r\n', '19.361669\r\n', '-99.182897\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(33, 'Francia\r\n', '19.3584\r\n', '-99.18395\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(34, 'Olivo\r\n', '19.354597\r\n', '-99.185189\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(35, 'Altavista\r\n', '19.350767\r\n', '-99.186397\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(36, 'La Bombilla\r\n', '19.346728\r\n', '-99.187781\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(37, 'Dr. Gálvez\r\n', '19.340664\r\n', '-99.191447\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(38, 'C.U.\r\n', '19.322944\r\n', '-99.188514\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(39, 'C.C.U.\r\n', '19.314561\r\n', '-99.1875\r\n', 'Línea 1');

```
INSERT INTO `Estaciones` VALUES(40, 'Perisur\r\n', '19.304039\r\n', '-99.186133\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(41, 'Villa Olímpica\r\n', '19.298908\r\n', '-99.185511\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(42, 'Corregidora\r\n', '19.294044\r\n', '-99.181053\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(43, 'Ayuntamiento\r\n', '19.292578\r\n', '-99.177589\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(44, 'Fuentes Brotantes\r\n', '19.288081\r\n', '-99.174614\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(45, 'Santa Úrsula\r\n', '19.283264\r\n', '-99.175328\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(46, 'La Joya \r\n', '19.280308\r\n', '-99.17', 'Línea 1');

INSERT INTO `Estaciones` VALUES(47, 'El Caminero\r\n', '19.279511\r\n', '-99.168981\r\n', 'Línea 1');

INSERT INTO `Estaciones` VALUES(48, 'Tacubaya\r\n', '19.401947\r\n', '-99.187053\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(49, 'Antonio Maceo\r\n', '19.404822\r\n', '-99.185908\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(50, 'Parque Lira\r\n', '19.407758\r\n', '-99.189067\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(51, 'De la Salle\r\n', '19.407569\r\n', '-99.1835\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(52, 'Patriotismo\r\n', '19.405569\r\n', '-99.177386\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(53, 'Escandón\r\n', '19.404406\r\n', '-99.173844\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(54, 'Nuevo León\r\n', '19.403494\r\n', '-99.170431\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(55, 'Viaducto\r\n', '19.401308\r\n', '-99.168092\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(56, 'Amores\r\n', '19.396881\r\n', '-99.163786\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(57, 'Etiopía\r\n', '19.395825\r\n', '-99.155019\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(58, 'Dr. Vértiz\r\n', '19.3956\r\n', '-99.154486\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(59, 'Centro SCOP\r\n', '19.395258\r\n', '-99.146778\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(60, 'Álamos\r\n', '19.394761\r\n', '-99.142939\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(61, 'Xola\r\n', '19.394364\r\n\r\n', '-99.140278\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(62, 'Las Américas\r\n', '19.393444\r\n', '-99.134353\r\n', 'Línea 2');
```

INSERT INTO `Estaciones` VALUES(63, 'Andrés Molina\r\n', '19.397653\r\n', '-99.129725\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(64, 'La Viga\r\n', '19.398028\r\n', '-99.124608\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(65, 'Coyuya\r\n', '19.398267\r\n', '-99.116761\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(66, 'Canela\r\n', '19.39785\r\n', '-99.109406\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(67, 'Tlacotal\r\n', '19.396878\r\n', '-99.099986\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(68, 'Goma\r\n', '19.397206\r\n', '-99.103894\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(69, 'Iztacalco\r\n', '19.396561\r\n', '-99.095967\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(70, 'UPIICSA\r\n', '19.393928\r\n', '-99.0904\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(71, 'El Rodeo\r\n', '19.391686\r\n', '-99.087128\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(72, 'Río Tecolutla\r\n', '19.389064\r\n', '-99.083189\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(73, 'Río Mayo\r\n', '19.386956\r\n', '-99.080031\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(74, 'Rojo Gómez\r\n', '19.384503\r\n', '-99.076328\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(75, 'Río Frío\r\n', '19.387606\r\n', '-99.074278\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(76, 'Del Moral\r\n', '19.384239\r\n', '-99.070958\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(77, 'Leyes de Reforma\r\n', '19.383461\r\n', '-99.065036\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(78, 'CCH Oriente\r\n', '19.383289\r\n', '-99.0608\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(79, 'Constitución de Apatzingán\r\n', '19.388961\r\n', '-99.059858\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(80, 'Canal de San Juan\r\n', '19.397792\r\n', '-99.056524\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(81, 'Nicolás Bravo\r\n', '19.395525\r\n', '-99.051045\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(82, 'Gral. A. de León\r\n', '19.385308\r\n', '-99.05175\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(83, 'Tepalcates\r\n', '19.390597\r\n', '-99.0473\r\n', 'Línea 2');

INSERT INTO `Estaciones` VALUES(84, 'Tenayuca\r\n', '19.529269\r\n', '-99.170117\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(85, 'San José de la Escalera\r\n', '19.523339\r\n', '-99.166094\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(86, 'Progreso Nacional\r\n', '19.518972\r\n', '-99.166261\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(87, 'Tres Anegas\r\n', '19.515042\r\n', '-99.161833\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(88, 'Júpiter\r\n', '19.508189\r\n', '-99.15915\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(89, 'La Patera\r\n', '19.503139\r\n', '-99.157164\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(90, 'Poniente 146\r\n', '19.499464\r\n', '-99.155667\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(91, 'Montevideo\r\n', '19.495422\r\n', '-99.154292\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(92, 'Poniente 134\r\n', '19.491667\r\n', '-99.153044\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(93, 'PONIENTE 128\r\n', '19.488931\r\n', '-99.151833\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(94, 'Magdalena de las Salinas\r\n', '19.482861\r\n', '-99.149472\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(95, 'Coltongo\r\n', '19.479314\r\n', '-99.148136\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(96, 'Cuitláhuac\r\n', '19.473381\r\n', '-99.145931\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(97, 'Héroe de Nacozari\r\n', '19.470625\r\n', '-99.144894\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(98, 'Hospital la Raza\r\n', '19.467528\r\n', '-99.143697\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(99, 'La Raza\r\n', '19.466622\r\n', '-99.141347\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(100, 'Circuito\r\n', '19.463361\r\n', '-99.143903\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(101, 'Tolnahuac\r\n', '19.459725\r\n', '-99.144203\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(102, 'Tlatelolco', '19.455972\r\n', '-99.144986\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(103, 'Ricardo Flores Magón\r\n', '19.452133\r\n', '-99.145864\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(104, 'Guerrero\r\n', '19.445472\r\n', '-99.14735\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(105, 'Buenavista', '19.445761\r\n', '-99.152158\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(106, 'Mina\r\n', '19.440061\r\n', '-99.148675\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(107, 'Hidalgo\r\n', '19.435769\r\n', '-99.147239\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(108, 'Juárez\r\n', '19.431681\r\n', '-99.148056\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(109, 'Balderas\r\n', '19.427558\r\n', '-99.148817\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(110, 'Cuauhtémoc\r\n', '19.424486\r\n', '-99.153747\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(111, 'Jardín Pushkin\r\n', '19.419947\r\n', '-99.154094\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(112, 'Hospital General\r\n', '19.414717\r\n', '-99.154578\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(113, 'Dr. Márquez\r\n', '19.411386\r\n', '-99.154819\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(114, 'Centro Médico\r\n', '19.406814\r\n', '-99.155158\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(115, 'Obrero Mundial\r\n', '19.401447\r\n', '-99.155581\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(116, 'Etiopía\r\n', '19.395894\r\n', '-99.155906\r\n', 'Línea 3');

INSERT INTO `Estaciones` VALUES(117, 'Buenavista\r\n', '19.445244\r\n', '-99.1523\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(118, 'Delegación Cuauhtémoc\r\n', '19.442683\r\n', '-99.152489\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(119, 'Puente de Alvarado\r\n', '19.439053\r\n', '-99.153347\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(120, 'Museo de San Carlos\r\n', '19.437886\r\n', '-99.149367\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(121, 'Hidalgo\r\n', '19.437186\r\n', '-99.146181\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(122, 'Bellas Artes\r\n', '19.436542\r\n', '-99.142025\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(123, 'Teatro Blanquita\r\n', '19.438514\r\n', '-99.139356\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(124, 'Rep. de Chile\r\n', '19.437964\r\n', '-99.13575\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(125, 'Rep. de Argentina\r\n', '19.437442\r\n', '-99.131669\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(126, 'Teatro del Pueblo\r\n', '19.436978\r\n', '-99.128214\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(127, 'Mixcalco\r\n', '19.436425\r\n', '-99.124147\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(128, 'FFCC de Cintura\r\n', '19.436072\r\n', '-99.12135\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(129, 'Morelos\r\n', '19.435794\r\n', '-99.118994\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(130, 'Archivo Gral. de la Nación\r\n', '19.43535\r\n', '-99.115103\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(131, 'San Lázaro\r\n', '19.430711\r\n', '-99.115542\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(132, 'Terminal 1\r\n', '19.435414\r\n', '-99.083311\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(133, 'Terminal 2\r\n', '19.421322\r\n', '-99.076972\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(134, 'Eduardo Molina\r\n', '19.427161\r\n', '-99.115567\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(135, 'Hospital Balbuena\r\n', '19.424967\r\n', '-99.114917\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(136, 'Cecilio Robelo\r\n', '19.425514\r\n', '-99.119517\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(137, 'Mercado de Sonora\r\n', '19.423258\r\n', '-99.123425\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(138, 'La Merced\r\n', '19.425389\r\n', '-99.125517\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(139, 'Circunvalación\r\n', '19.428331\r\n', '-99.125708\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(140, 'Las Cruces\r\n', '19.428617\r\n', '-99.12925\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(141, 'Museo de la Ciudad\r\n', '19.429067\r\n', '-99.132392\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(142, 'Isabel la Católica\r\n', '19.429681\r\n', '-99.136589\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(143, 'El Salvador\r\n', '19.43005\r\n', '-99.139258\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(144, 'Eje Central\r\n', '19.430439\r\n', '-99.141867\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(145, 'Plaza San Juan\r\n', '19.430878\r\n', '-99.144894\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(146, 'Juárez\r\n', '19.431297\r\n', '-99.147739\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(147, 'Vocacional 5\r\n', '19.431747\r\n', '-99.150942\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(148, 'Expo Reforma\r\n', '19.433111\r\n', '-99.150592\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(149, 'Glorieta de Colón\r\n', '19.434244\r\n', '-99.153389\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(150, 'Plaza de la República\r\n', '19.437519\r\n', '-99.15375\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(151, 'Puente de Alvarado\r\n', '19.439022\r\n', '-99.153264\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(152, 'Delegación Cuauhtémoc\r\n', '19.442219\r\n', '-99.152408\r\n', 'Línea 4N');

INSERT INTO `Estaciones` VALUES(153, 'Buenavista\r\n', '19.445244\r\n', '-99.1523\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(154, 'Delegación Cuauhtémoc\r\n', '19.442683\r\n', '-99.152489\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(155, 'Puente de Alvarado\r\n', '19.439053\r\n', '-99.153347\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(156, 'Plaza de la República\r\n', '19.4369\r\n', '-99.154181\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(157, 'Glorieta de Colón\r\n', '19.433939\r\n', '-99.153558\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(158, 'Expo Reforma\r\n', '19.433261\r\n', '-99.150731\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(159, 'Vocacional 5\r\n', '19.431575\r\n', '-99.150044\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(160, 'Juárez\r\n', '19.431325\r\n', '-99.148256\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(161, 'Plaza San Juan\r\n', '19.430903\r\n', '-99.145603\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(162, 'Eje Central\r\n', '19.430308\r\n', '-99.141653\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(163, 'El Salvador\r\n', '19.43\r\n', '-99.139389\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(164, 'Isabel la Católica\r\n', '19.429667\r\n', '-99.136961\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(165, 'Museo de la Ciudad\r\n', '19.429056\r\n', '-99.133053\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(166, 'Pino Suárez \r\n', '19.426386\r\n', '-99.132039\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(167, 'Las Cruces\r\n', '19.426142\r\n', '-99.130231\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(168, 'La Merced\r\n', '19.42545\r\n', '-99.125894\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(169, 'Mercado Sonora\r\n', '19.423239\r\n', '-99.1241\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(170, 'Cecilio Robelo\r\n', '19.425231\r\n', '-99.119867\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(171, 'Eduardo Molina\r\n', '19.427161\r\n', '-99.115567\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(172, 'Moctezuma\r\n', '19.426886\r\n', '-99.11165\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(173, 'San Lázaro Oriente\r\n', '19.430608\r\n', '-99.115125\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(174, 'Archivo Gral. de la Nación\r\n', '19.435014\r\n', '-99.114622\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(175, 'Morelos\r\n', '19.435814\r\n', '-99.1185\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(176, 'FFCC de Cintura\r\n', '19.436092\r\n', '-99.121022\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(177, 'Mixcalco\r\n', '19.436389\r\n', '-99.123792\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(178, 'Teatro del Pueblo\r\n', '19.436589\r\n', '-99.125481\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(179, 'Rep. de Argentina\r\n', '19.437389\r\n', '-99.131392\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(180, 'Rep. de Chile\r\n', '19.437992\r\n', '-99.135475\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(181, 'Teatro Blanquita\r\n', '19.43865\r\n', '-99.139894\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(182, 'Bellas Artes\r\n', '19.436189\r\n', '-99.140883\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(183, 'Hidalgo\r\n', '19.437275\r\n', '-99.145872\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(184, 'Museo de San Carlos\r\n', '19.438317\r\n', '-99.151239\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(185, 'Puente de Alvarado\r\n', '19.439022\r\n', '-99.153264\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(186, 'Delegación Cuauhtémoc\r\n', '19.442219\r\n', '-99.152408\r\n', 'Línea 4S');

INSERT INTO `Estaciones` VALUES(187, 'San Lázaro\r\n', '19.433369\r\n', '-99.115312\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(188, 'Archivo General\r\n', '19.43859\r\n', '-99.11446\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(189, 'Mercado Morelos\r\n', '19.44207\r\n', '-99.11343\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(190, 'Deportivo Eduardo Molina\r\n', '19.445793\r\n', '-99.111542\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(191, 'Canal del Norte\r\n', '19.450568\r\n', '-99.110684\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(192, 'Río Consulado\r\n', '19.45421\r\n', '-99.108452\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(193, 'Santa Coleta\r\n', '19.458661\r\n', '-99.106821\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(194, 'Oriente 101\r\n', '19.463436\r\n', '-99.104761\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(195, 'Victoria\r\n', '19.468292\r\n', '-99.102873\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(196, 'Talismán\r\n', '19.472095\r\n', '-99.100727\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(197, 'Río Guadalupe\r\n', '19.477193\r\n', '-99.099268\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(198, 'San Juan de Aragón \r\n', '19.481482\r\n', '-99.097208\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(199, 'Preparatoria 3\r\n', '19.484637\r\n', '-99.095148\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(200, 'El Coyol\r\n', '19.488198\r\n', '-99.094118\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(201, 'Vasco de Quiroga\r\n', '19.493376\r\n', '-99.092058\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(202, '5 de Mayo\r\n', '19.497907\r\n', '-99.090771\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(203, '314 Memorial News Divine\r\n', '19.501871\r\n', '-99.087853\r\n', 'Línea 5');

INSERT INTO `Estaciones` VALUES(204, 'Río de los Remedios\r\n', '19.507535\r\n', '-99.085621\r\n', 'Línea 5');