
# Benemérita Universidad Autónoma de Puebla

**FACULTAD DE CIENCIAS DE LA COMPUTACIÓN**

"PLATAFORMA DE CÓMPUTO MÓVIL PARA LA ADMINISTRACIÓN DE RUTAS DE TRANSPORTE PÚBLICO CONSIDERANDO EL ANÁLISIS DE TRANSPORTE EN TIEMPO REAL"

TESIS PRESENTADA PARA OBTENER EL TÍTULO DE:

**Licenciatura en Ingeniería en Ciencias de la Computación**

PRESENTA:

**David Efraín Muñoz Morales**

ASESORA:

**M.C. Meliza Contreras González**

PUEBLA, PUE. ABRIL 2018

**“Who am I?**

I am the hands of grandpa…

I am the tears of my mother…

The strength of father…

The jokes of all my brothers…

I am the entire love of my girlfriends…

And the ruthless discipline of my teachers.

I am the inspiration of many to move onwards

and the crowd that applauded my success…

I am the advice of a hundred men...

I am not just me, I am the sum of everything…

The proud result of other’s work,

People who touched my life in many ways…

Now it’s time to give back”

## AGRADECIMIENTOS

*Primeramente agradezco a Dios por estar presente en mi vida,*

*Por colocarme en una familia cuyo apoyo es incondicional,*

*A mi padre y madre por ser mi mejor escuela y permitirme estudiar,*

*A mis hermanos: Isaac, Oscar y Hugo, por su confianza en mí,*

*Al mejor ser vivo que tuve la fortuna de conocer: HODI,*

*A mi auténtico amigo Juan, por ser como un hermano,*

*A mi querida Martha por ser un sendero en mi camino,*

*A mi ejemplar asesora de Tesis Meliza por su inspiración,*

*Y a mis maestros: Yalú, Mariano, Pedro, David… por su motivación*

# PREFACIO

La elaboración de la presente tesis surgió del interés personal de profundizar en la planificación de rutas de transporte público en la Ciudad de Puebla. Durante el transcurso de mi carrera en la licenciatura en Ingeniería en Ciencias de la Computación, siempre mostré un especial interés en el desarrollo de aplicaciones móviles capaces de satisfacer las necesidades de la población en términos de movilidad urbana. De esta manera, mi principal reto en esta investigación consistió en desarrollar una plataforma de cómputo móvil con la que, concesionarios de transporte público así como la Secretaria de Comunicaciones y Transportes (SCT) del Estado de Puebla pudiesen planificar nuevas rutas de transporte público capaces de satisfacer la demanda de usuarios en la Ciudad de Puebla, así como ofrecer una manera más fácil de buscar llegar de un lugar a otro dentro de la Ciudad; con el fin de mejorar la planificación y administración de rutas de transporte público de Puebla.

Para el desarrollo de la plataforma fue importante analizar las rutas de transporte público de la Ciudad de Puebla y, además se observó el nivel de satisfacción de los usuarios, con el fin de determinar nuevas rutas que puedan satisfacer la demanda de usuarios.

Sustentado en una metodología ágil para el desarrollo de software, se produjo rápidamente un software útil, con base a una serie de incrementos donde cada uno de ellos incluyó una nueva funcionalidad del sistema.

Por otro lado, se ha hecho uso de las directrices de diseño de Material Design (Diseño de materiales), el cual es un lenguaje visual que sintetiza los principios clásicos de un buen diseño con la innovación y la tecnología móvil; asegurando de esta manera una interfaz intuitiva para los usuarios.

La plataforma fue desarrollada bajo la IDE (Entorno de Desarrollo Integrado) oficial de Android <<Android Studio>>, compatible con todos los celulares y tabletas Android.

# CONTENIDO

# ÍNDICE DE FIGURAS

# GLOSARIO Y ACRÓNIMOS

*A*

| | |
|---|---|
| Algoritmo | Conjunto ordenado de operaciones sistemáticas que permite hacer un cálculo y hallar la solución de un tipo de problemas. |
| App | Aplicación informática diseñada para ser ejecutada en dispositivos móviles (Smartphones, Tablets, entre otras). |
| Árbol | Un árbol es una estructura (posiblemente no lineal) de datos compuesta de nodos, vértices y aristas que es acíclica. |

*G*

| | |
|---|---|
| GPS | Global Positioning System |

*S*

| | |
|---|---|
| SDK | Software Development Kit |
| Smartphone | Teléfono móvil construido sobre una plataforma informática móvil, el término inteligente, hace referencia a la capacidad de usarse como un computador de bolsillo. |
| SO | Sistema Operativo |
| Spike | Es un método que se usa para determinar cuánto trabajo se requerirá para resolver o solucionar un problema de software. |

*T*

| | |
|---|---|
| Tiempo real | Es un sistema que interactúa con el mundo real, emitiendo respuestas correctas y cumpliendo restricciones temporales. |

*W*

| | |
|---|---|
| Wireframe | Dibujo esquemático que representa el esqueleto de una interfaz de usuario. |

*X*

| | |
|---|---|
| XP | Extreme Programming |

# INTRODUCCIÓN

## Análisis de Redes [1]

Desde una perspectiva de las Ciencias de la Computación, el análisis de redes es aplicado a la teoría de grafos. Puede ser objeto de estudio de la optimización como en el caso del método de la ruta crítica; así como el estudio de las propiedades estructurales y su dinámica. Las redes pueden ser de diversos tipos: social, transporte, eléctrica, biológica, internet, entre otras. Y para esta investigación nos interesan las redes de transporte.

## Análisis de redes de transporte [13]

Las redes de transporte son objeto de estudio particular en ciertos casos donde se pretende analizar el transporte de bienes y personas entre diversas áreas geográficas. Uno de los objetivos de su estudio es a veces la mejora y la eficiencia del tráfico, a través de la planificación del transporte. En este análisis los nodos suelen ser las ciudades, los aeropuertos, las estaciones, etc. Mientras que los enlaces suelen ser las carreteras, las autovías, etc.

*Figura I. 1 Análisis de redes de transporte. Gran Tucumán, Argentina*

## Transporte público [14]

El transporte público es el término aplicado al transporte colectivo de pasajeros, los cuales deben adaptarse a los horarios y a las rutas que ofrezca el operador. El transporte público tiene un impacto positivo en la economía, el medio ambiente y las comunidades.

El transporte público urbano permite el desplazamiento de personas de un punto a otro en el área de una ciudad. Además, el transporte público urbano puede ser proporcionado por una o varias empresas privadas o por consorcios de transporte público. Los servicios se mantienen mediante cobro directo a los pasajeros. Normalmente son servicios regulados y subvencionados por autoridades locales o nacionales.

## Ámbito

En la ciudad de Puebla se cuenta con diferentes servicios de transporte público para la movilidad de los ciudadanos, tales como el servicio de taxis y camiones, pero son estos últimos nuestro objetivo de estudio, ya que existen diversas rutas de transporte público.

Las diferentes rutas de transporte público que hay en la ciudad, operan en diferentes horarios, su capacidad de pasajeros varía así como el costo por viaje, además de las condiciones bajo las que trabajan. Dejando a un lado lo anterior, es preocupante que a pesar de la tecnología de hoy en día el gobierno y los consorcios encargados del transporte no inviertan lo suficiente en investigación y estudios para satisfacer la demanda de transporte de una manera eficiente.

Todo esto se debe a una mala planificación y a que no cuentan con herramientas adecuadas que apoyen al estudio de las rutas de transporte público para la administración y planificación de nuevas rutas capaces de satisfacer la demanda de usuarios [15].

**Descripción del problema**

El transporte público de la ciudad de Puebla es una parte importante en la sociedad, sus objetivos están enfocados a brindar servicios de movilidad a los ciudadanos para llegar de manera fácil a casi cualquier destino de la ciudad, esto gracias a su amplia gama de rutas.

Estas rutas conectan a la ciudad con diferentes puntos, incluyendo aquellos puntos que se encuentran fuera de la ciudad, como Cholula, Amozoc, Tlaxcalancingo, e incluso con el estado vecino de Tlaxcala. Por lo anterior, es claro que las rutas de transporte público de Puebla nos permiten llegar a casi cualquier parte de la ciudad. Pero con tantas rutas de transporte es difícil que los usuarios conozcan de su existencia.

Debido al gran número de rutas y a que no todas pertenecen al mismo sistema, es que muchas de ellas ofrecen un servicio deficiente, haciendo que en ocasiones llegar a un punto cercano sea muy tardado. Esto provoca que las personas tengan que invertir más de su tiempo para llegar a su trabajo, a la escuela y en general a sus destinos. Esto puede orillar a algunos a tomar un taxi, derivando en una inversión extra.

Por otro lado la implementación de propuestas tecnológicas para satisfacer las necesidades de la sociedad así como para mejorar su estilo de vida contemplando el mínimo gasto de recursos es todo un reto. Es por esto y considerando que la gran mayoría de la población cuenta con un Smartphone y acceso a internet, que se hace eminente una solución a través de aplicaciones móviles.

Por lo que surge la inquietud de construir una plataforma de aplicaciones móviles que sirva de apoyo a todo usuario de transporte público para conocer la información en tiempo real acerca de las rutas de transporte en la ciudad de Puebla, permitiendo así conocer la ubicación actual de los camiones, además de visualizar su desplazamiento a cada momento, así como hacer búsquedas para ir de un lugar a otro sin necesidad de preocuparse por los cambios inesperados en las rutas de los camiones.

## Objetivos

### General.-

Proporcionar una plataforma tecnológica con aplicaciones móviles que permita satisfacer la demanda del servicio de transporte público en la ciudad de Puebla, a través de la generación de rutas mediante el análisis de Redes de Transporte considerando la reducción de los tiempos de espera.

### Específicos.-

- Elegir y aplicar la metodología de desarrollo de software más apropiada para el diseño y construcción del sistema.
- Construcción de algoritmos de búsqueda de rutas considerando el tiempo-costo del servicio.
- Diseño intuitivo de la interfaz de usuario.
- Generación de rutas a partir de la demanda de un grupo de usuarios que solicitan el servicio.
- Generación de reportes sobre las rutas más demandadas así como nuevas rutas a establecer.

## Metodología de desarrollo

Para el diseño del sistema que permite cumplir con los objetivos señalados en el apartado anterior se llevó a cabo una serie de actividades, y son las siguientes:

1. Se revisaron las diferentes metodologías de desarrollo de software para elegir la que más resulta adecuada al proyecto.
2. Una vez elegida la metodología de desarrollo ágil de software, se aplicaron los procesos de análisis y diseño señalados, obteniendo como resultado diagramas y modelos para el desarrollo de la plataforma.
3. Para el desarrollo del algoritmo de búsqueda, se estudió y aplicó la teoría de grafos tomando como base el algoritmo de Dijkstra [19].

4. En base a las directrices de diseño de Material Design, se estableció la interfaz para lograr una experiencia de usuario única.
5. En base a las búsquedas de rutas de transporte público realizadas en la aplicación y a los lugares ingresados como origen y destino, fue posible sugerir nuevas rutas de transporte cubran la demanda de usuarios, así como la generación de reportes de estos datos [15].
6. Conforme se desarrollaba cada módulo se realizaron pruebas para garantizar su funcionabilidad. Permitiendo hacer una prueba total de la plataforma para verificar que cuenta con las funcionalidades esperadas.

La terminación de este proyecto facilita la movilidad urbana en Puebla, gracias a su fácil uso por los usuarios, que ahora tienen más alternativas de rutas de transporte público para llegar de un lugar a otro. Además de representar ser una herramienta de apoyo para el gobierno y los consorcios de transporte público, para administración y planificación de rutas de transporte.

# CAPÍTULO I

# ESTADO DEL ARTE

## 1.1 Estado del Arte

Gracias al avance tecnológico en el mundo de los Smartphones, en los últimos años se han desarrollado numerosas aplicaciones para facilitar las actividades cotidianas de las personas. Una de estas actividades es la movilidad, ya que día con día la población en las ciudades aumenta y los servicios de transporte lo hacen a la par, resulta en ocasiones imposible tener conocimiento de todas las posibilidades para llegar de un punto a otro.

Es por lo anterior que resulta eminente tratar los últimos avances tecnológicos en aplicaciones, respecto a la movilidad urbana dentro de la ciudad. Algunas de estas soluciones a tratar son móviles mientras que otras son web.

### 1.1.1 Google Maps Transit [16]

Es un servidor de aplicaciones de mapas en la web, y se encuentra disponible tanto para web como en su versión para Smartphones, y permite a los usuarios la navegación GPS en tiempo real, la visualización del tráfico, el tránsito y los detalles sobre millones de lugares, como reseñas y horarios populares.

Además, desde que Google Maps añadió información del transporte público en el año 2007, ya son más de 18,000 ciudades de 64 países que han incluido sus rutas y horarios de transporte público. Sin embargo, no fue sino hasta el año 2017 que Google Maps permitió conocer las rutas de transporte público en la Ciudad de Puebla.

Ahora, los ciudadanos de Puebla pueden conocer la información sobre que camiones de transporte público tomar para ir de un lugar a otro, así como una aproximación del tiempo que dura el trayecto de viaje. Muestra todas las opciones posibles a tomar de acuerdo a las rutas de transporte público

proporcionadas por el gobierno de Puebla. Sin embargo, estas rutas no están actualizadas y los horarios no son reales, ya que constantemente trabajan bajo cambios indeterminados.

El modo de uso es muy simple, y se muestra en la figura 1.1. Aquí se puede observar que solo se requiere ingresar un lugar de origen y un lugar destino. Google Maps muestra en un listado todas las opciones posibles de rutas a tomar, con sus respectivos tiempos estimados de recorrido.

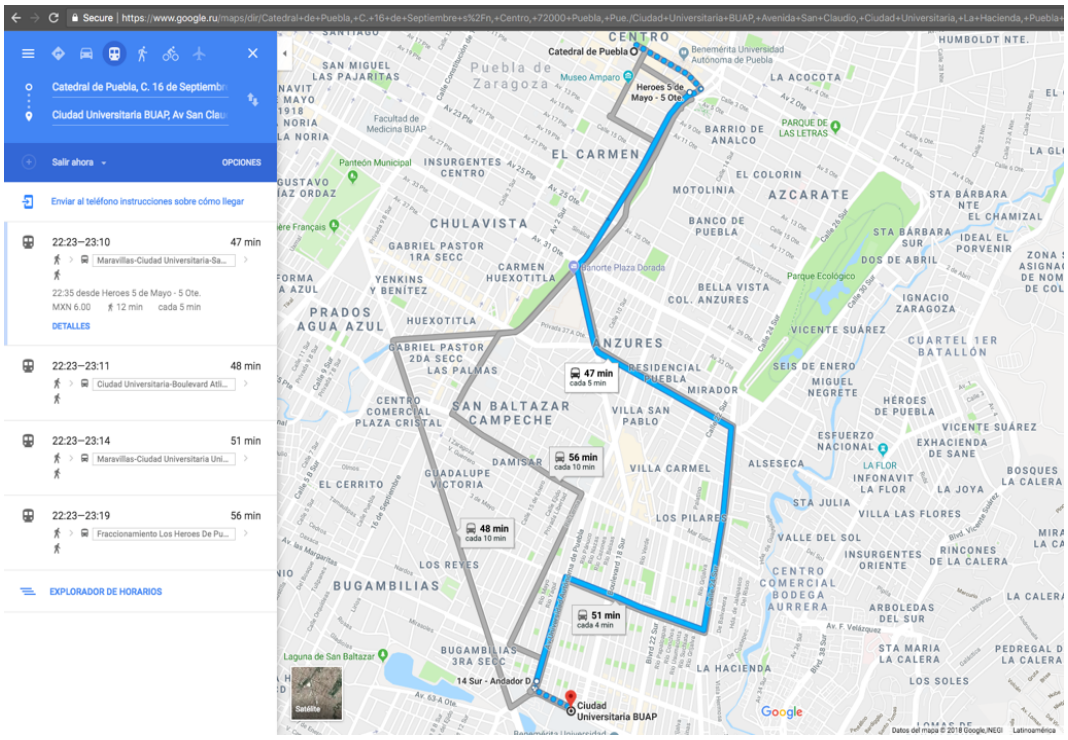



*Figura 1. 1 Uso de Google Maps para encontrar rutas de transporte público de un punto a otro*

A pesar de proveer las posibles opciones de rutas a ser usadas y de estimar un tiempo de trayecto, no se basa en información actualizada. Ya que así como en muchas ciudades, Puebla, está en remodelación constante, las obras públicas requieren que se cierren ciertas calles, obligando a los diferentes camiones a tomar calles alternativas, además, los concesionarios con frecuencia cambian los trayectos, y todo esto imposibilita a los usuarios a conocer los cambios hechos a las rutas en tiempo real. Es por esto que aunque

en muchas partes del mundo esta funcionalidad de Google Maps ayuda mucho, en Puebla se ve limitada.

### 1.1.2 Moovit [17]

Es una aplicación móvil de transito público local operando en más 1700 ciudades alrededor del mundo. Posee información robusta y muy precisa de cualquier transporte público, como autobuses, trenes, metros subterráneos, entre otros.

Moovit funciona bajo el concepto de la importancia de hacer conocer los cambios impredecibles en las rutas de transporte público, haciendo constantes actualizaciones de los cambios hechos a los horarios de servicio de las rutas. En la figura 1.2 se puede apreciar la importancia en que destacan este concepto dentro de su tutorial de inicio, al instalar la aplicación sobre un celular.

Una característica importante es que permite conocer el tiempo exacto de llegada del próximo autobús que estás esperando. Sin embargo, esto solo es real en ciudades Europeas y Estadounidenses, donde los sistemas de transportes son bien diseñados y cuentan con una verdadera inversión y soporte constante. Dado a que en Puebla la situación es muy distinta, resulta imposible conocer el tiempo de llegada del próximo autobús. Pues algunos conductores echan carreras, mientras que otros deciden tomar atajos, o simplemente por algún imprevisto como el tráfico, calles cerradas, o fallas en los sistemas de los camiones.

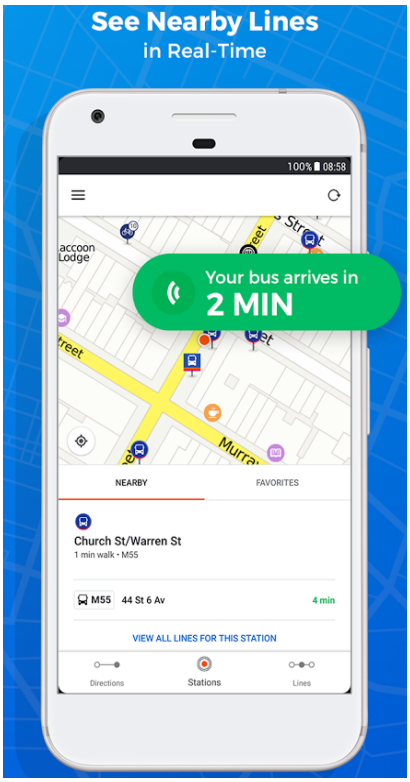



*Figura 1. 2 Tiempo estimado de llegada del próximo camión*

### 1.1.3 Rutadirecta [18]

Es una aplicación tanto móvil como web, que ayuda a las personas a encontrar rutas de transporte público que pueden tomar para llegar de un punto a otro en la ciudad de Puebla. Al igual que Google Maps Transit y Moovit, Rutadirecta permite buscar rutas en otras ciudades, aunque no es tan grande su número de opciones, ya que solo cuenta con 10 ciudades en México, una ciudad en Argentina, y una en USA.

Para realizar una búsqueda solo basta con mantener el dedo sobre un punto del mapa e indicarlo como origen, y luego otro para el destino. También se ofrece la opción de buscar Rutas por punto, esto resulta de ayuda cuando se quiere verificar si una ruta pasa por un lugar determinado.

Su interfaz es más simple, pero menos atractiva. Otra cosa a destacar es que simplemente provee el listado de rutas de la ciudad, así como las soluciones

a cierta búsqueda, sin proveer información específica como el horario. En la figura 1.3 se puede observar la pantalla principal de la aplicación.


*Figura 1. 3 Interfaz móvil de la aplicación Rutadirecta*

## 1.2 Justificación

Dado el análisis de las aplicaciones anteriores en materia de transporte público, es posible percatarse del gran avance que Google Maps Transit representa en temas de movilidad, y aunque permite la navegación GPS a vehículos privados sobre todo, ha logrado incorporar información de transporte público de varias ciudades en el mundo. Por otro lado, Moovit no se queda atrás en cuanto a soluciones de movilidad pública, pues permite hacer búsquedas para llegar de un lugar a otro, además de ofrecer información detallada acerca de los servicios de transporte público, e incluso mostrar aproximaciones del tiempo de llegada del próximo autobús.

A pesar de las grandes implementaciones de estas compañías para ofrecer herramientas tecnológicas capaces de facilitar la movilidad urbana en las ciudades,

es preciso mencionar que en Puebla no son del todo eficientes y, se debe a dos grandes razones:

- En Puebla los servicios de transporte público son muy distintos, pues pertenecen a concesiones privadas y otras más al gobierno, por lo que los costos y cantidad de unidades en buen estado son variantes. Además cada una de ellas trabaja bajo diferentes horarios. Por otro lado, la mayoría de los choferes son muy poco conscientes de su responsabilidad al estar frente al volante, por lo que ponen en riesgo la integridad de los usuarios, al echar carreritas o pasarse los altos.
- Al ser soluciones en masa las aplicaciones anteriormente mencionadas, ninguna de ellas se enfoca en una sola ciudad, como lo es Puebla.

Es por todo lo antes mencionado que la solución propuesta en el presente trabajo es emplear el Análisis de Redes de Transporte para mejorar la movilidad urbana en la Ciudad de Puebla donde se analizan las diferentes rutas de transporte público con el fin de hacer más eficientes los desplazamientos de las personas para ir de un lugar a otro [15].

Por otro lado, la implementación de propuestas tecnológicas para satisfacer las necesidades de la sociedad, así como para mejorar su estilo de vida contemplando el mínimo de gasto de recursos, es todo un reto. Sin embargo, al considerar que la gran mayoría de la población cuenta con un Smartphone y acceso a Internet, hace suponer que es a través de aplicaciones móviles que se debe atender este problema.

Dado esto, surge la inquietud de construir una plataforma de aplicaciones móviles que sirva de apoyo a todo usuario de transporte público para conocer información en tiempo real acerca de las rutas de la ciudad de Puebla, permitiendo así conocer la ubicación actual de los camiones además de visualizar su desplazamiento a cada momento, así como hacer búsquedas para ir de un lugar a otro sin necesidad de preocuparse por los cambios inesperados en las rutas de los camiones.

# CAPÍTULO II

# MARCO TEÓRICO

Dado que el enfoque de esta tesis está centrado en el desarrollo de una plataforma tecnológica para la administración de rutas de transporte público considerando el análisis de redes de transporte, es importante plantear algunos conceptos que sirvan de ejes conceptuales para apoyar la lectura interpretativa de la presente.

Mencionado lo anterior, a continuación se explican los conceptos teóricos que establecen las bases para la correcta aplicación de las mismas en los procesos de análisis, diseño, desarrollo y pruebas de la plataforma objetivo.

## 2.1 Metodologías para el desarrollo de software

Las metodologías de desarrollo de software se refieren a un marco de trabajo que es utilizado para estructurar, planear y controlar el proceso de desarrollo en sistemas de información. Por lo que pueden ser vistas como un proceso de software el cual es una serie de actividades relacionadas que conduce a la elaboración de un producto de software [6].

Procesos de software existen muchos, pero todos deben incluir cuatro actividades que resultan cruciales para la Ingeniería de software, y son las siguientes [6]:

1. *Especificación del software*

   Tienen que definirse tanto la funcionalidad del software como las restricciones de su operación.
2. *Diseño e implementación del software*

   Debe desarrollarse el software para cumplir con las especificaciones.
3. *Validación del software*

   Hay que validar el software para asegurarse de que cumple lo que el cliente quiere.
4. *Evolución del software*

El software tiene que evolucionar para satisfacer las necesidades cambiantes del cliente.

### 2.1.1 Desarrollo ágil de software

En la actualidad, el desarrollo rápido de sistemas de software, es por lo general un requerimiento fundamental. Esto se debe principalmente a que las empresas hoy en día operan en un entorno global que cambia rápidamente. En este sentido, es preciso responder frente a nuevas oportunidades y mercados, al cambio en las condiciones económicas, así como al surgimiento de productos y servicios competitivos [8].

El software es parte de casi todas las operaciones industriales, de modo que el nuevo software se desarrolla rápidamente para aprovechar las actuales oportunidades, con la finalidad de poder competir en el mercado actual [8].

Debido al mercado de las aplicaciones móviles, ahora el software funciona bajo un entorno cambiante, tanto así que a menudo es prácticamente imposible derivar un conjunto completo de requerimientos de software estable. De esta manera, los requerimientos cambian de modo inevitable, porque los clientes encuentran imposible predecir cómo un sistema responderá a las necesidades actuales, en relación a las tecnologías nuevas. Incluso, es posible que debido a factores externos, los requerimientos cambien rápida e impredeciblemente. En tal caso, el software podría ser obsoleto al momento de entregarse [8].

Por lo que, los procesos de desarrollo de software que buscan especificar por completo los requerimientos y, luego, diseñar y probar el sistema, no están orientados al desarrollo rápido de software [8].

Los procesos de desarrollo de software rápido se diseñan para producir rápidamente un software útil. El software no se desarrolla como una sola unidad, sino como una serie de incrementos, y cada uno de ellos incluye una nueva funcionalidad del sistema. Por otro lado, es importante destacar que los

procesos de especificación, diseño e implementación están entrelazados. No existe una especificación detallada del sistema, y la documentación del diseño se minimiza o es generada automáticamente en el entorno de programación que se usa para implementar el sistema [8].

La filosofía detrás de los métodos agiles se refleja en el manifiesto ágil, que acordaron muchos de los desarrolladores líderes de estos métodos. Este manifiesto afirma [6]:

> Estamos descubriendo mejores formas para desarrollar software, al hacerlo y al ayudar a otros a hacerlo. Gracias a este trabajo llegamos a valorar [6]:
>
> *A los individuos y las interacciones sobre los procesos y las herramientas*
>
> *Al software operativo sobre la documentación exhaustiva*
>
> *La colaboración con el cliente sobre la negociación del contrato*
>
> *La respuesta al cambio sobre el seguimiento de un plan*
>
> Esto es, aunque exista valor en los objetos a la derecha, valoraremos más los de la izquierda.

#### 2.1.1.1 Programación extrema

La programación extrema es quizás el método ágil mejor conocido y más ampliamente usado. El nombre lo acuño Beck (2000) debido a que el enfoque se desarrolló llevando a niveles “extremos” las prácticas reconocidas, como el desarrollo iterativo. Por ejemplo, en la XP muchas versiones actuales de un sistema pueden desarrollarse mediante

diferentes programadores, integrarse y ponerse a prueba en un solo día [7].

En la programación extrema, los requerimientos se expresan como escenarios (llamados historias de usuario), que se implementan directamente como una serie de tareas. Los programadores trabajan y antes de escribir el código desarrollan pruebas para cada tarea. Todas las pruebas deben ejecutarse con éxito una vez que el nuevo código se integre en el sistema. Entre las liberaciones del sistema existe un breve lapso. La figura 2.1 (*Imagen tomada de [6]*) ilustra el proceso XP para producir un incremento del sistema por desarrollar [6].

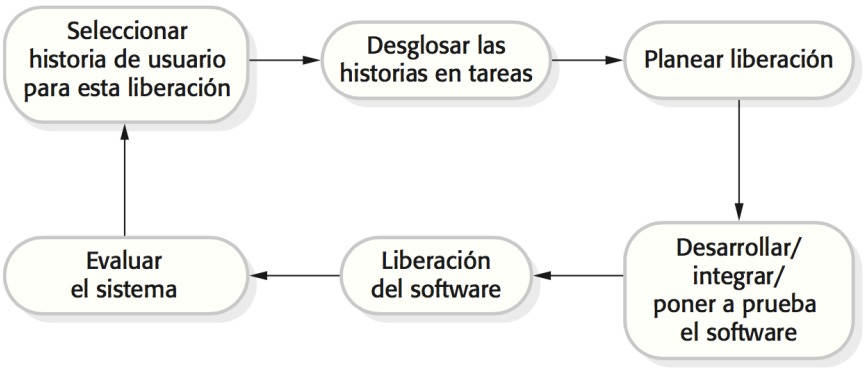



*Figura 2. 1 El ciclo de liberación de la programación extrema*

La programación extrema incluye algunas prácticas que reflejan los principios de los métodos agiles [7]:

1. El desarrollo incremental se apoya en pequeñas y frecuentes liberaciones del sistema. Los requerimientos se fundamentan en simples historias del cliente, o bien, en escenarios usados como base para decidir qué funcionalidad debe incluirse en un incremento del sistema.
2. La inclusión del cliente se apoya a través de un enlace continuo con el cliente en el equipo de desarrollo. El representante del

cliente participa en el desarrollo y es responsable de definir las pruebas de aceptación para el sistema.

3. Las personas, no los procesos, se basan en la programación en pares, en la propiedad colectiva del código del sistema y en un proceso de desarrollo sustentable que no incluya jornadas de trabajo excesivamente largas.
4. El cambio se acepta mediante liberaciones regulares del sistema a los clientes, desarrollo de primera prueba, refactorización para evitar degeneración del código e integración continua de nueva funcionalidad.
5. Mantener la simplicidad se logra mediante la refactorización constante, que mejora la calidad del código, y con el uso de diseños simples que no anticipan innecesariamente futuros cambios al sistema.

## 2.2 Modelos de base de datos

Los sistemas de bases de datos se diseñan para gestionar grandes cantidades de información. La colección de datos, normalmente denominada base de datos, contiene información relevante principalmente para una empresa. La gestión de los datos implica tanto la definición de estructuras para almacenar la información como la provisión de mecanismos de manipulación de la información [10].

Bajo la estructura de la base de datos se encuentra el modelo de datos, el cual es una colección de herramientas conceptuales para describir los datos, las relaciones, la semántica y las restricciones de consistencia. Un modelo de base de datos muestra la estructura lógica de la base, incluidas las relaciones y limitaciones que determinan cómo se almacenan los datos y cómo se accede a ellos [10].

Los modelos de bases de datos individuales se diseñan en base a las reglas y los conceptos del modelo de datos que se elija. Aunque diferentes modelos

aplican a diferentes etapas del proceso de diseño de bases de datos. Los modelos de datos conceptuales de alto nivel son mejores para crear mapas de relaciones entre los datos en las formas en que la gente percibe esos datos. Por otro lado, los modelos lógicos basados en registros reflejan más estrechamente las formas en que los datos se almacenan en el servidor.

### 2.2.1 Modelo entidad-relación

El modelo de datos entidad-relación está basado en una percepción del mundo real que consta de una colección de objetos básicos, llamados entidades, y de relaciones entre estos objetos. Una entidad es una <<cosa>> u <<objeto>> en el mundo real que es distinguible de otros objetos. Por ejemplo, casa persona es un entidad, y las cuentas bancarias pueden ser consideradas entidades [11].

Las entidades se describen en una base de datos mediante un conjunto de atributos. Por ejemplo, los atributos numero-cuenta y saldo describen cada cuenta particular de un banco y pueden ser atributos del conjunto de entidades cuenta. Análogamente, los atributos nombre-cliente, calle-cliente y ciudad-cliente pueden describir una entidad cliente [11].

Un atributo extra, id-cliente, se usa para identificar unívocamente a los clientes (dado que puede ser posible que haya dos clientes con el mismo nombre, dirección y ciudad). Se debe asignar un identificador único de cliente a cada cliente.

Una relación es una asociación entre varias entidades. Por ejemplo, una relación impositor asocia un cliente con cada cuenta que tiene. El conjunto de todas las entidades del mismo tipo, y el conjunto de todas las relaciones del mismo tipo, se denominan respectivamente conjunto de entidades y conjunto de relaciones [11].

### 2.2.2 Normalización

La normalización de bases de datos es un proceso que consiste en designar y aplicar una serie de reglas a las relaciones obtenidas tras el paso del modelo entidad-relación.

Las bases de datos se normalizan para evitar la redundancia de los datos, disminuir problemas de actualización o de los datos en las tablas, así como proteger la integridad de los datos. Una base de datos está normalizada si cumple con los requisitos naturales para funcionar óptimamente y no perjudicar el desempeño por mala arquitectura. Y normalmente solo basta con aplicar las 3 primeras formas normales [12].

#### 2.2.2.1 Dependencia Funcional

Este concepto de dependencia funcional se tomó de las matemáticas discretas elementales. Se dice que Y es función de X, Y = f(X), si el valor de Y está siempre determinado por el valor de X.

Si se aplica la misma terminología a una relación, la dependencia funcional entre los atributos A y B en una relación se define de la siguiente manera:

> *El atributo A es funcionalmente dependiente del atributo B si el valor de A está determinado por el valor de B.*

En otras palabras, una dependencia funcional es una conexión entre uno o más atributos. A partir de la dependencia funcional se determina la forma normal en que se encuentra la relación, para que se pueda aplicar el proceso de normalización [12].

#### 2.2.2.2 Formas Normales

El proceso de normalización comienza con la combinación de todos los datos de la base en una relación, la que a su vez se descompone en dos o más relaciones más pequeñas [12].

Primera Forma Normal (1FN)

Esta forma normal elimina los valores repetidos dentro de una Base de Datos.

Una tabla está en Primera Forma Normal si:

- Todos los atributos son atómicos. Un atributo es atómico si los elementos del dominio son simples e indivisibles.
- La tabla contiene una clave primaria única.
- La clave primaria no contiene atributos nulos.
- No debe existir variación en el número de columnas.
- Los campos no clave deben identificarse por la clave (Dependencia Funcional)
- Debe existir una independencia del orden tanto de las filas como de las columnas.

Segunda Forma Normal (2FN)

Dependencia completa. Está en 2FN si esta en 1FN y si sus atributos no principales dependen de forma completa de la clave principal.

Tercera Forma Normal (3FN)

La tabla se encuentra en 3FN si esta en 2FN y si no existe ninguna dependencia funcional transitiva en los atributos que no son clave.

Dicho de otra forma, un esquema de relación R está en 3FN, si para toda dependencia funcional $X \rightarrow A$, se cumple al menos una de las siguientes condiciones:

1. X es superllave o clave.
2. A es atributo primo de R, esto es, si es miembro de alguna clave en R.

Aparte de cumplir dicho esquema obligatoriamente, con las condiciones de 2FN.

## 2.3 Modelos de redes

Hay una multitud de situaciones, en investigación de operaciones, que se pueden modelar y resolver con el uso de grafos (red de nodos conectados por aristas). Hasta hace una década algunos informes decían que alrededor del 70% de los problemas de programación matemática en el mundo real se pueden presentar como modelos relacionados con grafos. Algunas de las aplicaciones posibles de los grafos se listan a continuación [2].

1. Diseño de una red de gasoductos marinos para conectar bocas de pozos en el Golfo de México con un punto de entrega en tierra. El objetivo del modelo es minimizar el costo de construcción del gasoducto.
2. Determinación de la ruta más corta entre dos ciudades, en una red de carreteras.
3. Determinación de la capacidad máxima (en toneladas anuales) de una red de tubería para lodo de carbón que une las minas en Wyoming con las centrales eléctricas en Houston. (Las tuberías de lodo de carbón transportan el carbón suspendido en agua a través de tubos de diseño especial.)
4. Determinación del programa de flujo con costo mínimo desde los campos petroleros hasta las refinerías a través de una red de oleoductos.
5. Determinación del cronograma (fechas de inicio y terminación) de las actividades en la construcción de un proyecto.

La solución de estas situaciones y otras parecidas se logra con una variedad de algoritmos de optimización de redes. Los más importantes a considerar son los siguientes [2]:

1. Árbol de expansión mínima
2. Algoritmo de la ruta más corta
3. Algoritmo del flujo máximo
4. Algoritmo de red capacitada con costo mínimo
5. Algoritmo de la ruta crítica

Las situaciones en las que se pueden aplicar estos algoritmos también se pueden formular y resolver en forma de programas lineales explícitos. Sin embargo, los algoritmos propuestos, basados en redes, son más eficientes que el método simplex [2].

### 2.3.1 Definiciones para redes (grafos)

Un grafo consiste en una serie de nodos enlazados con aristas (o arcos). La notación para describir un grafo es *(N, A)*, donde *N* es el conjunto de nodos y *A* es el conjunto de aristas [1].

Por ejemplo, la red de la figura 2.2 se define como sigue:

N = {1,2,3,4,5}

A = {(1,2), (1,3), (2,3), (2,5), (3,4), (3,5), (4,2), (4,5)}

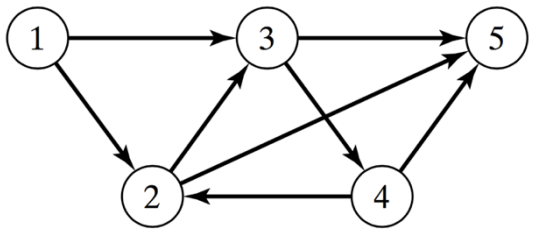



*Figura 2. 2 Ejemplo de una red (N,A)*

Con cada red se asocia algún tipo de flujo (por ejemplo, flujo de productos petroleros en un oleoducto y flujos de tráfico de automóviles en carreteras). En general, el flujo un una red está limitado por la capacidad de sus arcos, que pueden ser finitos o infinitos [1].

Se dice que un arco es dirigido u orientado si permite un flujo positivo en una dirección, y flujo cero en la dirección opuesta. Una red dirigida tiene todos sus arcos dirigidos [1].

Una ruta es una sucesión de arcos distintos que unen dos nodos pasando por otros nodos, independientemente de la dirección de flujo de cada arco. Una ruta forma un ciclo si conecta un nodo consigo mismo, pasando por otros nodos [2]. Por ejemplo, en la figura 2.2, los arcos (2,3), (3,5) y (5,2) forman un bucle o circuito cerrado. Un ciclo es dirigido si consiste en una ruta dirigida, por ejemplo (2,3), (3,4) y (4,2) en la figura 2.2.

Una red conectada es aquella en que cada dos nodos distintos están enlazados al menos por una ruta. La red de la figura 2.2 es un ejemplo de este tipo. Un árbol es una red conectada que puede consistir solo en un subconjunto de todos los nodos en ella, donde no se permiten ciclos, y un árbol de expansión es un árbol que enlaza todos los nodos de la red, también sin permitir ciclos [1]. En la figura 2.3 se ven ejemplos de un árbol y de un árbol de expansión para la red de la figura 2.2.

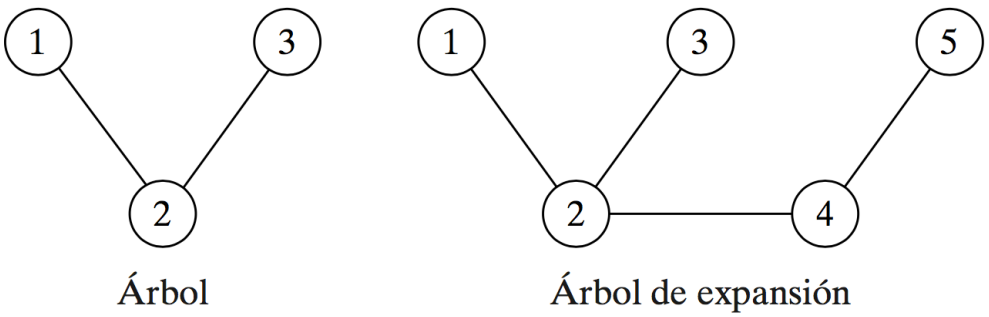



*Figura 2. 3 Ejemplos de un árbol y de un árbol de expansión, para la red de la figura 2.2*

## 2.4 Problema de la ruta más corta

En el problema de la ruta más corta se determina ésta, entre una fuente y un destino, en una red de transporte.

### 2.4.1 El algoritmo de Dijkstra

El algoritmo de Dijkstra tiene por objeto determinar las rutas más cortas entre el nodo fuente y todos los demás nodos de la red [2].

**Algoritmo.** Sea $u_i$ la distancia más corta del nodo origen 1 al nodo i, y defina $d_{ij}(>=0)$ como la longitud del arco (i,j). El algoritmo define la etiqueta para un nodo j que sigue inmediatamente como

$$[u_j,i] = [u_i + d_{ij},i], d_{ij} >= 0$$

La etiqueta para el nodo de inicio es [0, 2], que indica que el nodo no tiene predecesor.

Las etiquetas de nodo en el algoritmo de Dijkstra son de dos tipos: temporales y permanentes. Una etiqueta temporal en un nodo se modifica si puede hallarse una ruta más corta al nodo. De lo contrario, el estado temporal cambia a permanente.

**Paso 0.** Etiquete el nodo del origen (nodo 1) con la etiqueta permanente [0, -]. Establezca i = 1.

**Paso general I.**

(a) Calcule las etiquetas temporales $[u_i + d_{ij}, i]$ para cada nodo j con $d_{ij} > 0$, siempre que j no esté etiquetado permanentemente. Si el nodo j ya tiene una etiqueta temporal existente $[u_j, k]$ hasta otro nodo k y si ui + $d_{ij} < u_j$, reemplace $[u_j, k]$ con $[u_i + d_{ij}, i]$.

(b) Si todos los nodos tienen etiquetas permanentes deténgase. De lo contrario, seleccione la etiqueta [ur, s] que tenga la distancia más corta ( = ur) entre todas las etiquetas temporales (rompa los empates arbitrariamente). Establezca i = r y repita el paso i.

### 2.4.2 El algoritmo de Floyd

El algoritmo de Floyd es general, porque permite determinar la ruta más corta entre dos nodos cualesquiera de la red. El algoritmo representa una red de n nodos como una matriz cuadrada con n filas y n columnas. La entrada (i, j) de la matriz da la distancia $d_{ij}$ del nodo i al nodo j, la cual es finita si i está vinculado directamente a j, en infinita en caso contrario [2].

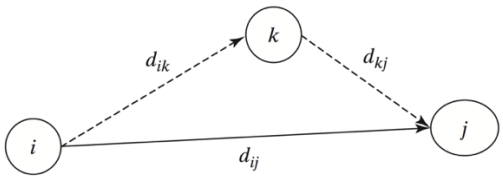



*Figura 2. 4 Operación triple de Floyd*

La idea del algoritmo de Floyd es simple. Dados tres nodos, i, j, k en la figura 2.4 con las distancias de conexión que se muestran en los tres arcos, es más corto llegar de j a i pasando por k si

$$d_{ik} + d_{kj} < d_{ij}$$

En este caso es óptimo reemplazar la ruta directa de i → j con la ruta indirecta i → k → j. Este intercambio de operación triple se aplica a la matriz de distancias por medio de los siguientes pasos:

## Algoritmo.

**Paso 0.** Defina la matriz de la distancia de inicio $D_0$ y la matriz de secuencia de nodos $S_0$ (todos los elementos en las diagonales están bloqueados). Establezca k = 1. (Como se muestra en la figura 2.5 (*Imagen tomada de [2]*)).

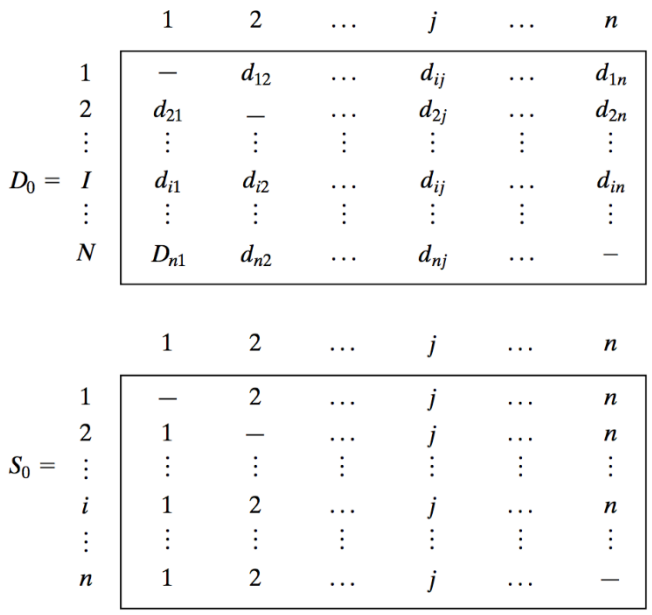


*Figura 2. 5 Definición de la matriz de distancia y la matriz S*

**Paso general k.** Defina la fila k y la columna k como fila pivote y columna pivote. Aplique la operación triple a cada elemento dij en Dk-1, para todas las i y j. Si la condición

$d_{ik} + d_{kj} < d_{ij}$, ($i \neq k$, $j \neq k$, y $i \neq j$)

Se satisface, realice los siguientes cambios:

- **a.** Cree $D_k$ reemplazando $d_{ij}$ en Dk-1 con $d_{ik} + d_{kj}$.
- **b.** Cree $S_k$ reemplazando $s_{ij}$ en $S_{k-1}$ con k. Establezca k = k + 1. Si k = n + 1, deténgase: de lo contrario repita el paso k.

El paso k del algoritmo puede explicarse representando $D_{k-1}$ como se muestra en la figura 2.6 (*Imagen tomada de [2]*). Aquí, la fila k y la columna k definen la fila y la columna pivote actuales. La fila i representa cualquiera de las filas 1, 2, … ,y k – 1, y la fila p representa cualquiera de las filas k + 1, k + 2, … , y n. Asimismo, la columna j representa cualquiera de las columnas 1, 2, … , y k – 1, y la columna q representa cualquiera de las columnas k + 1, k + 2, … , y n. La operación triple puede aplicarse como sigue: Si la suma de los elementos en la fila pivote y la columna (mostrados por cuadrados) es menor que el elemento de intersección asociado (mostrado por un círculo), entonces es

óptimo reemplazar la distancia de intersección por la suma de las distancias pivote.

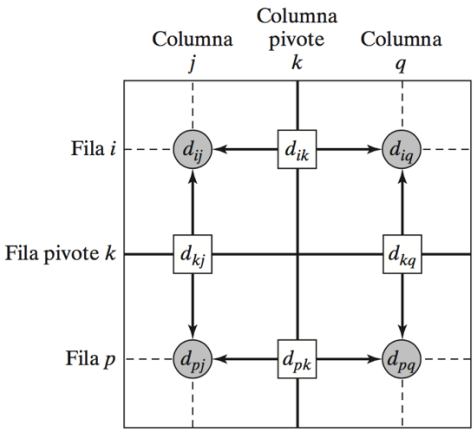



*Figura 2. 6 Implementación de la operación triple en forma de matriz*

Después de n pasos, podemos determinar la ruta más corta entre los nodos i y j a partir de las matrices $D_n$ y $S_n$ aplicando las siguientes reglas:

1. $d_{ij}$, a partir de $D_n$, da la ruta más corta entre los nodos i y j.
2. A partir de $S_n$, determine el nodo intermedio k = $s_{ij}$ que da en resultado la ruta i → k → j. Si $s_{ik}$ = k y $s_{kj}$ = j, deténgase; todos los nodos intermedios de la ruta han sido encontrados. De lo contrario, repita el procedimiento entre los nodos i y k y entre los nodos k y j.

## 2.5 Aplicación móvil

Las aplicaciones móviles –también llamadas apps– son en esencia un software informático diseñado y desarrollado para correr en dispositivos móviles, pero por lo regular en Smartphones y Tablets. Estas aplicaciones permiten a los usuarios efectuar una tarea en concreto para facilitarles sus actividades, y las hay de cualquier ámbito: profesional, educativas, de acceso a servicios, ocio, entre otras [3].

Al tratarse de aplicaciones que residen en los dispositivos móviles, éstas son escritas en algún lenguaje de programación compilado, y para hacerlo hay

varias maneras. Sin embargo, siempre es mejor hacer un desarrollo de forma nativa, pues para ello han sido desarrollados los Kits de desarrollo que ofrece cada sistema operativo (SO) a los programadores, estos Kits son llamados genéricamente *Software Development Kit* o SDK. De esta manera, Android, iOS y Windows Phone poseen un SDK diferente, y las aplicaciones nativas se diseñan y programan específicamente para cada plataforma, en el lenguaje utilizado por el SDK [4].

Las aplicaciones nativas no requieren Internet para poder ser usadas, por lo que ofrecen una experiencia de uso más fluida y están realmente integradas al teléfono, lo cual les permite utilizar todas las características de hardware, como la cámara y los sensores (GPS, acelerómetro, entre otros) [5].

A nivel de diseño, esta clase de aplicaciones tiene una interfaz basada en las directrices de diseño de cada sistema operativo, logrando mayor coherencia y consistencia con el resto de las aplicaciones y con el propio SO. Esto favorece la usabilidad de las apps, pues se logran unas interfaces intuitivas [3].

### 2.5.1 Lenguaje de diseño

Un lenguaje de diseño es un esquema general o estilo que guía el diseño de un complemento de productos o configuraciones arquitectónicas. Los diseñadores que desean dar a su conjunto de productos un aspecto único pero uniforme definen un lenguaje de diseño para él, que puede describir las opciones de aspectos de diseño, como materiales, combinaciones de colores, formas, patrones, texturas o diseños. Luego siguen el esquema en el diseño de cada objeto en el conjunto [9].

Por lo general, los lenguajes de diseño no están rigurosamente definidos; el diseñador básicamente hace una cosa de manera similar a otra. En otros casos, se siguen estrictamente, para que los productos adquieran una fuerte calidad temática. Por ejemplo, aunque hay una gran variedad de diseños de

juegos de ajedrez inusuales, las piezas dentro de un conjunto suelen ser temáticamente consistentes [9].

A veces, los diseñadores alientan a otros a seguir sus lenguajes de diseño cuando decoran o complementan. En el contexto de las interfaces gráficas de usuario, por ejemplo, las pautas de interfaz humana pueden considerarse como lenguajes de diseño para aplicaciones.

#### 2.5.1.1 Material Design

Es un lenguaje de diseño desarrollado en 2014 por Google. Y la verdad es que el mismo Google se desafío para crear un lenguaje visual para sus usuarios que sintetizara los principios clásicos del buen diseño con la innovación y la posibilidad de la tecnología y la ciencia.

Con Material Design, google permite desarrollar un único sistema subyacente que permita una experiencia unificada en todas las plataformas y tamaños de dispositivos (tal como se muestra en la figura 2.7 (*Imagen tomada de [20]*), el diseño adaptativo dentro de cada dispositivo). Los preceptos móviles son fundamentales, pero el tacto, la voz, el mouse y el teclado son todos métodos de entrada de primera clase.

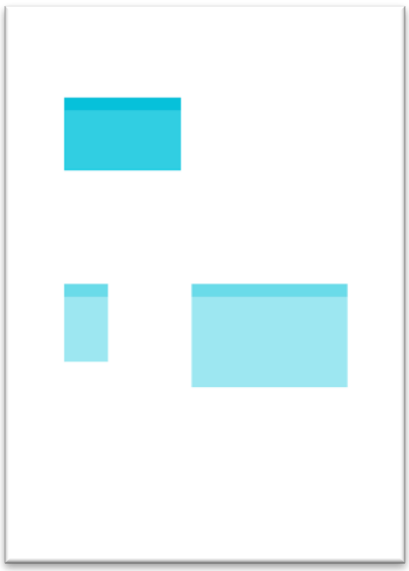


*Figura 2. 7 Diseño adaptativo*

#### 2.5.1.1.1 Principios

**Material es la metáfora**

Una metáfora material es la teoría unificadora de un espacio racionalizado y un sistema de movimiento. El material está basado en la realidad táctil, inspirado en el estudio del papel y la tinta, pero tecnológicamente avanzado y abierto a la imaginación y la magia [20].

Las superficies y los bordes del material proporcionan pistas visuales que se basan en la realidad. El uso de atributos táctiles familiares ayuda a los usuarios a comprender rápidamente las posibilidades. Sin embargo, la flexibilidad del material crea nuevas posibilidades que reemplazan a aquellas en el mundo físico, sin romper las reglas de la física.

Los fundamentos de la luz, la superficie y el movimiento son clave para transmitir cómo los objetos se mueven, interactúan y existen en el espacio y en relación entre sí (Como se muestra en

la figura 2.8). La iluminación realista muestra costuras, divide el espacio e indica partes móviles. Ver figura 2.9 (*Imagen tomada de [20]*).

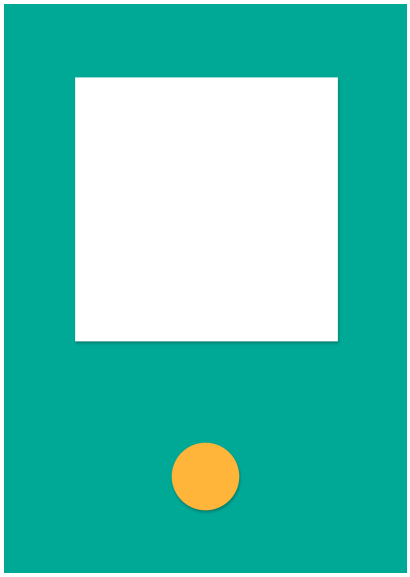


*Figura 2. 8 Superficie intuitiva y natural*

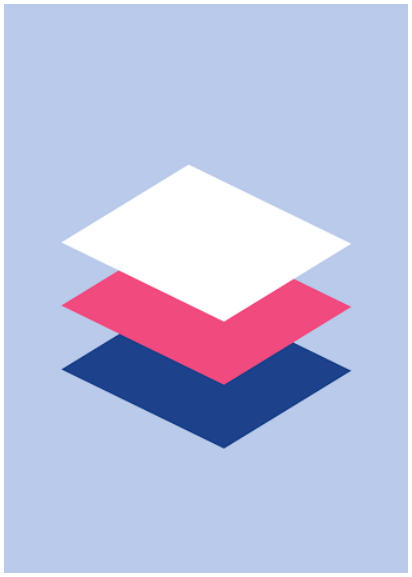


*Figura 2.9 La dimensionalidad permite la interacción*

**Intrépido, gráfico e intencional**

Los elementos fundamentales de diseño basado en la impresión (tipografía, color y uso de imágenes) guían los tratamientos visuales. Estos elementos hacen mucho más que agradar a los ojos. Crean jerarquía, significado y enfoque. Las elecciones de color deliberadas, las imágenes de borde a borde, la tipografía a gran escala y el espacio en blanco intencional crean una interfaz gráfica y audaz que sumerge al usuario en la experiencia [20]. Tal como se aprecia en la figura 2.10 (*Imagen tomada de [20]*).

El énfasis en las acciones del usuario hace que la funcionalidad central sea inmediatamente evidente y proporciona puntos de referencia para el usuario. Ver figura 2.11 (*Imagen tomada de [20]*).

*Figura 2. 10 Inmersión y claridad*

*Figura 2.11 El color y la superficie destacan las acciones*

**El movimiento proporciona significado**

El movimiento respeta y refuerza al usuario como principal motor (ver figura 2.12 (*Imagen tomada de [20]*)). Las acciones del usuario primario son puntos de inflexión que inician el movimiento, transformando todo el diseño [20].

Toda acción tiene lugar en un solo entorno. Los objetos se presentan al usuario sin romper la continuidad de la experiencia, incluso cuando se transforman y se reorganizan.

El movimiento es significativo y apropiado, lo que sirve para enfocar la atención y mantener la continuidad. La retroalimentación es sutil pero clara. Las transiciones son eficientes pero coherentes. Tal como es posible observar en la figura 2.13 (*Imagen tomada de [20]*).

De esta manera Material Design hace un uso más liberal de diseños basados en cuadriculas, animaciones y transiciones

receptivas, relleno y efectos de profundidad como iluminación y sombras.

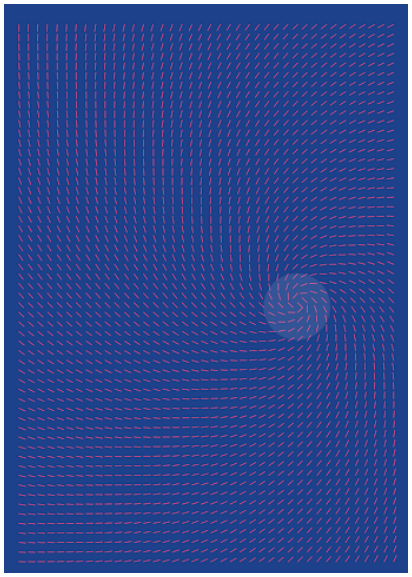


*Figura 2. 12 Los usuarios inician el cambio*

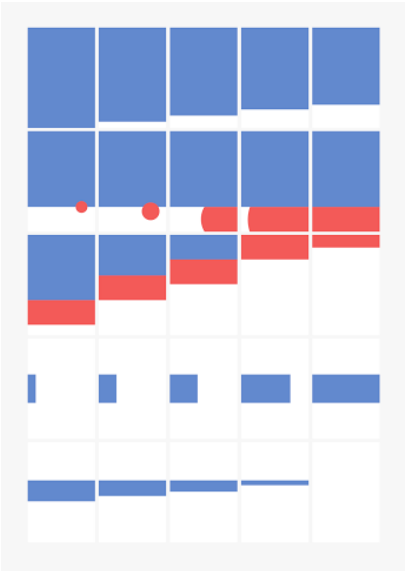


*Figura 2.13: La animación es como una coreografía*

Así es como Google ha logrado a través de su lenguaje de diseño Material Design un diseño plano que se muestra en toques de color brillante para reflejar un aspecto acogedor ligero. Las animaciones se han ajustado para hacer la experiencia más lúcida, al tiempo que garantiza la máxima cantidad de contenido siempre visible.

### 2.5.2 Proceso de diseño y desarrollo de una app

El proceso de diseño y desarrollo de una aplicación abarca desde la concepción de la idea hasta el análisis posterior a su publicación en las tiendas oficiales. Durante estas fases, tanto diseñadores como desarrolladores trabajan de manera simultánea [3]. El diagrama de estas fases puede observarse en la figura 2.14 (*Imagen tomada de [3]*).

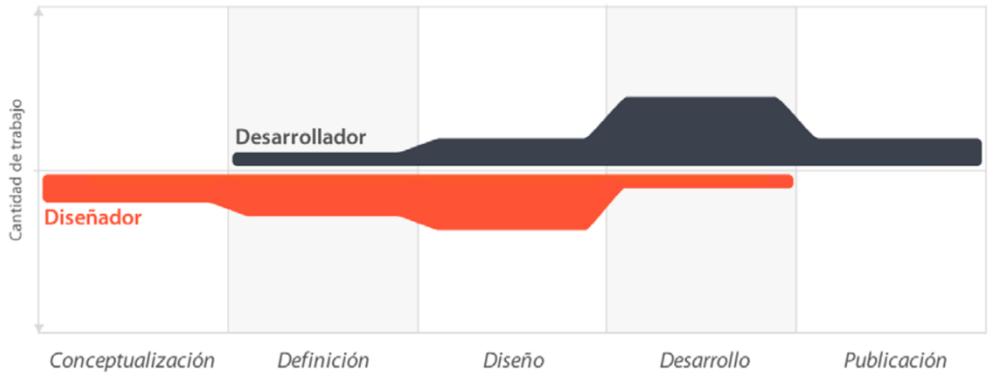



*Figura 2. 14 Fases del proceso de diseño y desarrollo de una app*

1. **Conceptualización**

El resultado de esta etapa es una idea de aplicación, que tiene en cuenta las necesidades y problemas de los usuarios a ser atendidas. La idea responde a una investigación preliminar y a la posterior comprobación de la viabilidad del concepto.

2. **Definición**

En este paso del proceso se describe con detalle a los usuarios para quienes se diseñará la aplicación, usando metodologías como <<Personas>> y <<Viaje de usuario>>. También aquí se sientan las bases de la funcionalidad, lo cual determinará el alcance del proyecto y la complejidad de diseño y programación de la app.

3. **Diseño**

En la etapa de diseño se llevan a un plano tangible los conceptos y definiciones anteriores, primero en forma de wireframes, que permiten crear los primeros prototipos para ser probados con usuarios, y posteriormente, en un diseño visual acabado que será provisto al desarrollador, en forma de archivos separados y pantallas modelo, para la programación del código.

4. **Desarrollo**

El programador se encarga de dar vida a los diseños y crear la estructura sobre la cual se apoyará el funcionamiento de la aplicación. Una vez que existe la versión inicial, dedica gran parte del tiempo a corregir errores funcionales para asegurar el correcto desempeño de la app y la prepara para su aprobación en las tiendas.

**5. Publicación**

La aplicación es finalmente puesta a disposición de los usuarios en las tiendas. Luego de este paso trascendental se realiza un seguimiento a través de analíticas, estadísticas y comentarios de usuarios, para evaluar el comportamiento y desempeño de la app, corregir errores, realizar mejoras y actualizar en futuras versiones.

# CAPÍTULO III

# ANÁLISIS Y DISEÑO

Este capítulo contempla el análisis de la plataforma de cómputo móvil considerando la metodología ágil de desarrollo de software: Programación Extrema, así que en base a los objetivos planteados en la Introducción es como podemos diseñar la plataforma. Ya que las tarjetas de historia de usuario son las entradas principales al proceso de planeación de XP, lo primero a hacer son las historias de usuario para que una vez diseñadas, puedan ser descompuestas en tareas y estimar el esfuerzo y los recursos requeridos para implementar cada tarea. Para su implementación es, se van a priorizar las historias y elegir aquellas que pueden usarse inmediatamente.

## 3.1 Historias de usuario

Como sabemos, en esta metodología de desarrollo los requerimientos se expresen como escenarios llamados historias de usuario, las cuales poseen las aquellas especificaciones y prioridades, de lo que necesita hacer el software.

Aunque existen muchos formatos para realizar estas historias de usuario, lo importante es que contengan:

- Número de historia
- Nombre
- Prioridad
- Iteración
- Descripción

En este caso, al tratarse de una plataforma también necesitamos especificar a qué parte de la plataforma (usuario) corresponde tal historia, y solo por no perder detalle agregaremos un espacio para observaciones.

La prioridad de cada historia es posible que cambie en el transcurso de la planificación de entrega, sin embargo es muy útil al momento de redactar los escenarios.

A continuación se presentan las historias de usuario de la plataforma a desarrollar.

| Historia de Usuario | |
|---|---|
| **Número:** 1 | **Usuario:** Usuario de transporte público |
| **Nombre:** Buscar un lugar de la ciudad en el mapa | |
| **Prioridad:** Alta | **Iteración Asignada:** 0 |
| **Descripción:** Se podrá ingresar un lugar a buscar dentro de la ciudad, entonces se colocará un marcador sobre el mapa (correspondiente a la geo posición del lugar). | |
| **Observaciones:** Al encontrarse el lugar, deberá almacenarse el punto (latitud, longitud) para ser usado en análisis de búsquedas. | |

| Historia de Usuario | |
|---|---|
| **Número:** 2 | **Usuario:** Usuario de transporte público |
| **Nombre:** Listar y visualizar las rutas de transporte público de la ciudad | |
| **Prioridad:** Alta | **Iteración Asignada:** 0 |
| **Descripción:** En forma de lista deberán mostrarse las rutas de transporte público, y al dar click sobre una de éstas, deberá dibujarse la ruta sobre el mapa. | |
| **Observaciones:** | |

| Historia de Usuario | |
|---|---|
| **Número:** 3 | **Usuario:** Usuario de transporte público |
| **Nombre:** Realizar una búsqueda para llegar de un lugar a otro | |
| **Prioridad:** Alta | **Iteración Asignada:** 0 |
| **Descripción:** Podrán realizar búsquedas, donde se deberá ingresar un origen y un destino, entonces se mostrarán las posibilidades para llegar del origen al destino haciendo uso de las rutas de transporte público. Además deberán alistarse en cuanto al camino más corto. | |
| **Observaciones:** | |

| Historia de Usuario | |
|---|---|
| **Número:** 4 | **Usuario:** Conductor |

| | |
|---|---|
| **Nombre:** Realizar un trayecto | |
| **Prioridad:** Media | **Iteración Asignada:** 0 |
| **Descripción:** El conductor de autobús podrá transmitir su recorrido mientras conduce. | |
| **Observaciones:** Esta app toma las coordenadas en tiempo real vía GPS y las almacena en el servidor de manera que puedan ser consultadas. | |

| **Historia de Usuario** | |
|---|---|
| **Número:** 5 | **Usuario:** Usuario de transporte público |
| **Nombre:** Visualizar en tiempo real los trayectos de los autobuses | |
| **Prioridad:** Baja | **Iteración Asignada:** 0 |
| **Descripción:** Se podrá seleccionar una ruta de transporte público en el listado de rutas, entonces, se ofrecerá la opción de poder visualizar su trayecto real sobre el mapa, con la intención de conocer que tan lejos o cerca está del usuario. | |
| **Observaciones:** Consulta las coordenadas almacenadas en el servidor. | |

| **Historia de Usuario** | |
|---|---|
| **Número:** 6 | **Usuario:** Concesionario o administrador |
| **Nombre:** Visualizar las rutas más buscadas así como los lugares de origen y destino | |
| **Prioridad:** Media | **Iteración Asignada:** 0 |
| **Descripción:** Se podrán visualizar sobre el mapa los orígenes y destinos más demandados, así como las rutas más usadas, con el fin de determinar nuevas mejoras sobre las rutas actuales, o incluso para proponer nuevas rutas de transporte público. | |
| **Observaciones:** En base a la información generada por la aplicación que usan los usuarios de transporte público. | |

Como es posible observar la plataforma consta de 3 tipos de usuario:

- *Usuario de transporte público*

  Enfocado a satisfacer las necesidades de búsqueda para llegar de un punto a otro dentro de la ciudad.
- *Conductor*

Con el objetivo principal de generar la información de su recorrido, con el fin de poder usarlo para visualización en tiempo real, así como para ser posibles mejoras en la ruta.

- *Administrador*

  En base a las búsquedas hechas por los usuarios, y las rutas generadas por los conductores, es posible determinar puntos en la ciudad que puedan ser atendidos por alguna ruta actual (realizando una modificación sobre esta).

Hasta este momento ya poseemos la entrada principal al proceso de planeación XP. Lo siguiente es descomponer en tareas y estimar el esfuerzo y recursos necesarios para cada implementación. Sin embargo, para comprender aún mejor el comportamiento de la plataforma y la relación con cada uno de los diferentes usuarios que interactúan en ella, haremos un diagrama general de casos. El cual se ha diseñado con el editor de diagramas DIA (Diagram Editor), esta herramienta facilita el modelado de estos casos de uso como se muestra en la figura 3.1.

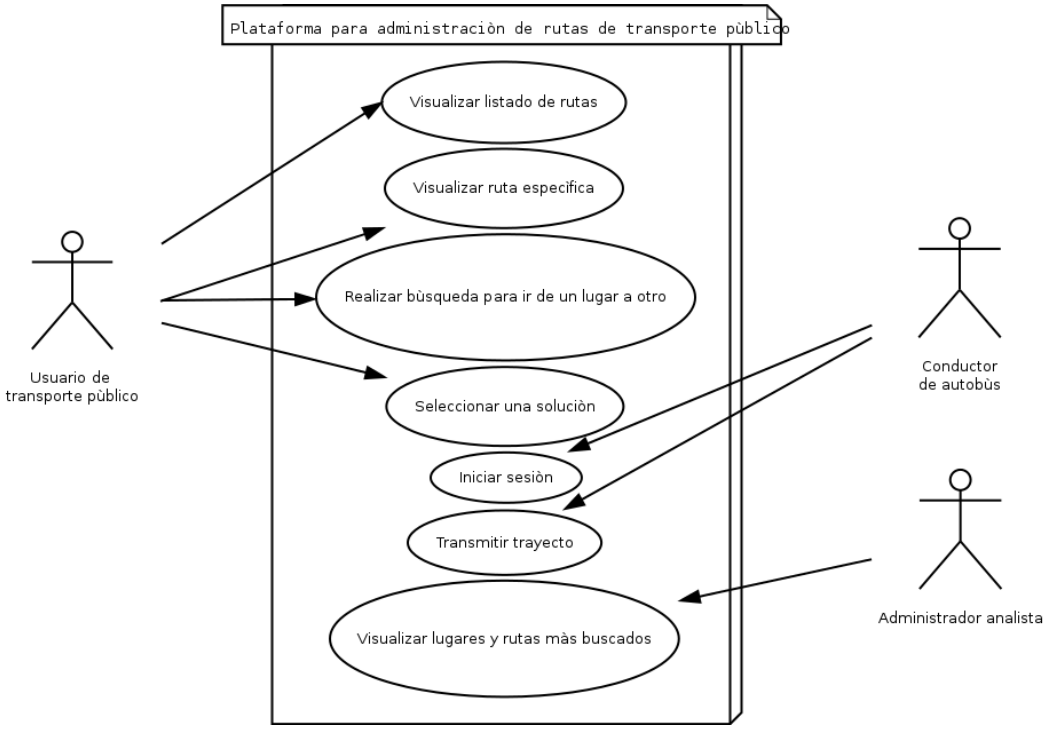



*Figura 3. 1 Diagrama general de casos*

## 3.2 Planeación

Es momento de dividir en tareas cada historia de usuario, para esto, hay ocasiones en que algunas historias no se pueden estimar bien hasta que el equipo de desarrollo hace un trabajo real para resolver una pregunta técnica o un problema de diseño. Es entonces cuando la solución a esta situación es crear un "spike", el cual es un trabajo cuyo propósito es proveer una respuesta a la solución. Podemos decir que un "spike" es un bosquejo.

### 3.2.1 Tareas

Comenzando por la primera historia de usuario, nos damos cuenta que tan solo se requiere de ingresar un lugar para que se coloque se obtenga su geolocalización sobre el mapa.

Para esto es necesario hacer uso de alguna API que provea las coordenadas así como listar los lugares reales. La alternativa más viable de todas las existentes es usando la API Google Places.

| **Tarea 1. Ingresar un lugar** | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 1 - Buscar un lugar de la ciudad en el mapa |
| **Fecha de Inicio:** 09 de octubre de 2017 | **Fecha Fin:** 16 de octubre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Proveer un cuadro de texto para escribir un lugar, este debe desplegar las opciones de autocompletado mientras se escribe. | |

| **Tarea 2. Obtener la coordenada del lugar y almacenarla** | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 1 - Buscar un lugar de la ciudad en el mapa |
| **Fecha de Inicio:** 16 de octubre de 2017 | **Fecha Fin:** 23 de octubre de 2017 |
| **Programador responsable:** David Muñoz | |

| |
|---|
| **Descripción:** En base al lugar seleccionado, se obtiene la coordenada (Latitud, Longitud) y se coloca un marcador sobre el mapa. |

Hasta aquí se ha tomado lo más básico para el desarrollo, y llega con gran peso el hecho de ofrecer al usuario la posibilidad de visualizar toda ruta de transporte público que existe en su ciudad. Aunque la lista de rutas este ordenada alfabéticamente, siempre resulta bueno permitir buscar una ruta en específico porque mejora el tiempo de búsqueda.

Es muy importante destacar que para mostrar las rutas de transporte, estas serán tomadas de la página de Datos Abiertos (la cual facilita en un archivo .json todas las rutas con su conjunto de puntos (Lat, Lng) por el que se compone cada una). Estos datos se pueden descargar en: https://datosabiertos.pueblacapital.gob.mx/login

| **Tarea 3. Listar rutas de transporte público** | |
|---|---|
| **Tipo:** Diseño y Desarrollo | **Historia:** 2 - Listar y visualizar las rutas de transporte público de la ciudad |
| **Fecha de Inicio:** 23 de octubre de 2017 | **Fecha Fin:** 30 de octubre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Mostrar en una lista ordenada alfabéticamente todas las rutas de transporte público de la ciudad. | |

| **Tarea 4. Permitir buscar una ruta específica** | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 2 - Listar y visualizar las rutas de transporte público de la ciudad |
| **Fecha de Inicio:** 30 de octubre de 2017 | **Fecha Fin:** 06 de noviembre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Justo arriba del listado de rutas debe haber un cuadro de búsqueda, para facilitar la misma al usuario. | |

| **Tarea 5. Visualizar ruta** | |
|---|---|
| **Tipo:** Diseño y Desarrollo | **Historia:** 2 - Listar y visualizar las rutas de transporte público de la ciudad |

| | |
|---|---|
| **Fecha de Inicio:** 06 de noviembre de 2017 | **Fecha Fin:** 13 de noviembre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Al seleccionar una ruta del listado, esta se dibuja sobre el mapa. | |

Llega un momento crucial dentro de la plataforma, pues es preciso facilitar a los usuarios de transporte público encontrar el mejor camino para llegar de un punto a otro dentro de la ciudad. Y aunque esta historia tiene una alta prioridad dentro del sistema, depende totalmente de las anteriores 2 para ser ejecutada satisfactoriamente.

| **Tarea 6. Diseño del algoritmo** | |
|---|---|
| **Tipo:** Diseño y Desarrollo | **Historia:** 3 - Realizar una búsqueda para llegar de un lugar a otro |
| **Fecha de Inicio:** 13 de noviembre de 2017 | **Fecha Fin:** 27 de noviembre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Usando lo logrado en las tareas 1 y 2, deberá obtenerse un lugar de partida y un lugar destino, estos parámetros (Lat, Lng) deben ser enviados al algoritmo. El algoritmo debe comparar con cada ruta estos parámetros y determinar cuál es la mejor en base a la distancia. | |

| **Tarea 7. Mostrar soluciones** | |
|---|---|
| **Tipo:** Diseño y Desarrollo | **Historia:** 3 - Realizar una búsqueda para llegar de un lugar a otro |
| **Fecha de Inicio:** 27 de noviembre de 2017 | **Fecha Fin:** 11 de diciembre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** En base a las soluciones que arroje el algoritmo (de existir soluciones), es como deberán listarse, permitiendo al usuario seleccionar alguna de ellas y mostrar sobre el mapa la ruta o rutas así como el punto origen y punto destino. | |

Con lo que se ha abarcado hasta ahora, los usuarios de transporte público poseerán una herramienta bastante sofisticada para la movilidad urbana. No obstante, puede ser mejorada aún más para permitir la visualización en tiempo real de las unidades de transporte público, cambiando para siempre

el esquema del servicio que provee el transporte público actualmente en la ciudad de Puebla.

| Tarea 8. Transmitir geo-posicionamiento actual | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 4 – Realizar un trayecto |
| **Fecha de Inicio:** 11 de diciembre de 2017 | **Fecha Fin:** 25 de diciembre de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Esta parte de la plataforma debe permitir a un conductor de autobús transmitir su recorrido en tiempo real, tomando las coordenadas del dispositivo móvil (Smartphone) y enviándolas a un servidor, para que estas puedan ser consultadas por los usuarios de transporte público. | |

| Tarea 9. Visualización de autobuses | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 5 – Visualizar en tiempo real los trayectos de los autobuses |
| **Fecha de Inicio:** 25 de diciembre de 2017 | **Fecha Fin:** 08 de enero de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Dentro de la aplicación móvil que corresponde a los usuarios de transporte público, se debe agregar la visualización en tiempo real de los autobuses dada una ruta específica. | |

Ahora que se cuenta con una app que genera la información en tiempo real de los trayectos de los autobuses, y con otra app que almacena información de los lugares más buscados así como las rutas más utilizadas, es momento de usar esta información para proponer mejoras en las rutas actuales o incluso plantear nuevas rutas por las autoridades encargadas de este servicio hacia los ciudadanos.

Al plasmarse esta valiosa información que generan ambas partes y mediante un análisis de redes de transporte, se pueden alcanzar los objetivos.

| Tarea 10. Mostrar la información generada por los usuarios | |
|---|---|
| **Tipo:** Desarrollo | **Historia:** 6 – Visualizar las rutas más buscadas así como los lugares de origen y destino |

| | |
|---|---|
| **Fecha de Inicio:** 08 de enero de 2017 | **Fecha Fin:** 22 de enero de 2017 |
| **Programador responsable:** David Muñoz | |
| **Descripción:** Mostrar sobre un mapa los lugares de origen y destino más demandados por los usuarios, así como las rutas más buscadas. Con el fin de simplificar la información a la vista y mediante análisis de los responsables del servicio, proponer mejoras sobre las rutas. | |

## 3.3 Diseño de la Base de datos

El modelo de datos son abstracciones que permiten la implementación de un sistema eficiente de base de datos. Así en el diseño de esta base de esta base de datos se necesita definir la estructura de los datos, así como las relaciones que existen entre estos.

Para lograr esto lo primero a analizar es el diseño conceptual y se hace a continuación.

### 3.3.1 Diseño conceptual

Este modelo es utilizado para representar la realidad a un alto nivel de abstracción. Así que, lo que se va a hacer será construir una descripción de la realidad fácil de entender.

Para expresar los resultados del diseño conceptual se utilizará el modelo entidad-relación y en particular para este proyecto de tesis se usa la notación estándar, misma que está disponible en el editor de diagramas DIA.

#### 3.3.1.1 Modelo entidad-relación

El modelo de datos entidad-relación está basado en una percepción del mundo real que consta de un conjunto de objetos básicos llamados entidades y de relaciones entre estos objetos.

#### 3.3.1.1.1 Entidades

Una entidad es una “cosa” u “objeto” en el mundo real que podemos distinguir de otros objetos y del que nos interesan sus propiedades.

Tomando esto en consideración, y analizando el funcionamiento los requerimientos de la plataforma, es posible observar que la principal información son las rutas, que a su vez son recorridas por camiones, mismas que manejan los conductores. Por otro lado, los lugares que buscan los usuarios (origen y destino) son algo que tienen propiedades.

Dado lo anterior, las entidades del presente modelo son las siguientes:

- CAMION
- RUTA
- CONDUCTOR
- LUGAR

#### 3.3.1.1.2 Atributos

Un atributo es una característica o propiedad de una entidad, conocido como elemento de datos (valor específico) para cada uno de sus atributos que se encuentran en los campos de un registro que describen a una entidad y así será posible su identificación única.

Aplicando la teoría y los conceptos del modelo para encontrar los atributos de las entidades anteriores, los atributos quedarían de la siguiente manera:

- CAMION (unidad, posicion)
- RUTA (nombre, busquedas)
- CONDUCTOR (usuario, contrasena)

- LUGAR (nombre, posicion, busquedas)

La representación de dichas entidades con sus respectivos atributos en el editor de diagramas DIA, se muestra a continuación en la figura 3.2.

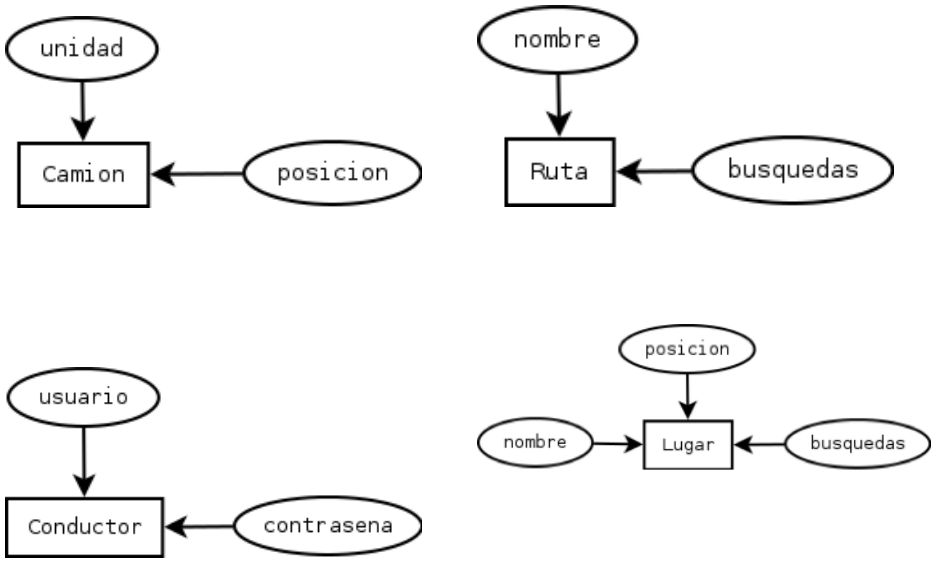



*Figura 3. 2 Atributos de las entidades*

### 3.3.1.1.3 Identificadores Únicos

Una entidad debe ser distinguible del resto de los objetos del mundo real. Esto hace que para toda entidad sea posible encontrar un conjunto de atributos que permitan identificarla.

Dicho en otras palabras, un identificador de una entidad es un atributo o conjunto de atributos que determina de modo único cada ocurrencia de esa entidad.

De esta manera se procede a analizar los atributos de las entidades para distinguir un identificador único o clave primaria, dado que no se han especificado en los requerimientos

identificadores únicos, es como se asignará un identificador único para cada entidad, tal como se muestra a continuación:

- RUTA (#id_Ruta)
- CAMION (#id_Camion)
- CONDUCTOR (#id_Conductor)
- LUGAR (#id_Lugar)

Los identificadores clave se representan con una línea bajo el nombre del identificador, y se muestran agregados a los diagramas de anteriores, tal como se muestra en la figura 3.3.

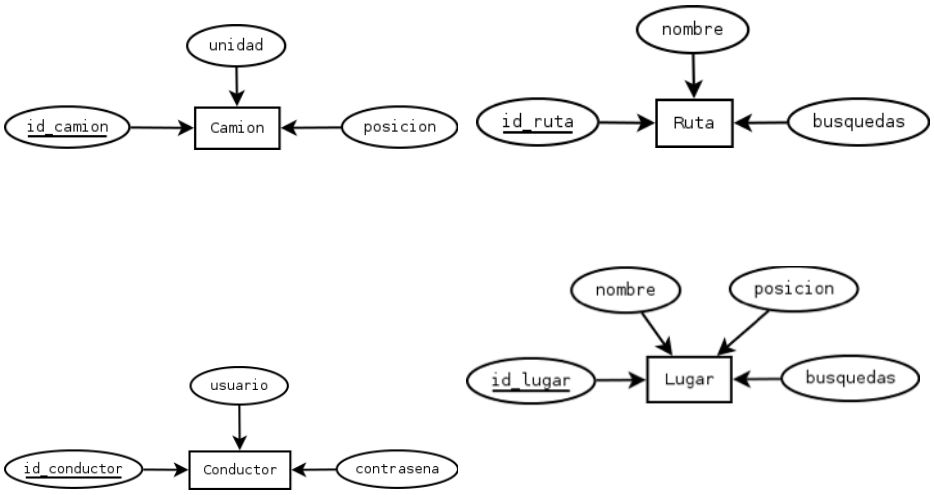



*Figura 3. 3 Identificadores de las entidades*

### 3.3.1.1.4 Relaciones

Una relación es una asociación entre varias entidades. Y en este proyecto representan las reglas y la información que el negocio necesita.

Así, en base a las entidades que ya se han establecido es importante establecer las relaciones que puedan existir entre las mismas y el tipo de relación. Las relaciones entre dichas entidades con el uso de la herramienta DIA se ilustran en la figura 3.4

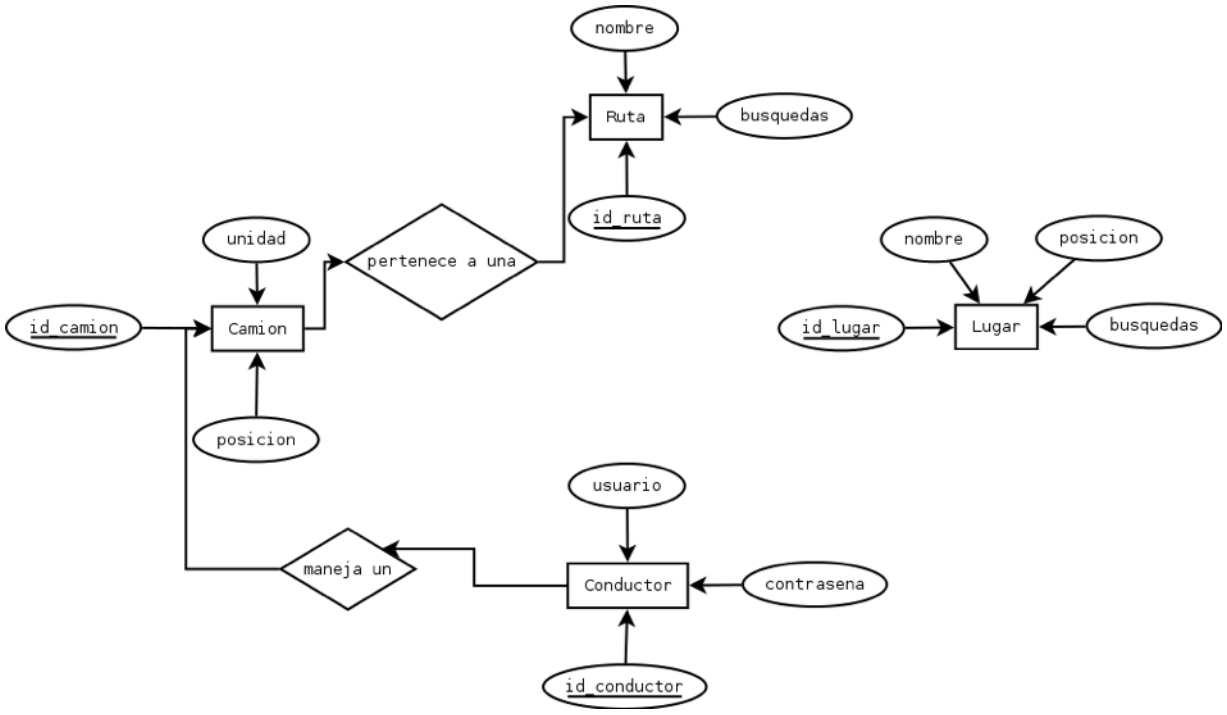



*Figura 3. 4 Relaciones entre las entidades*

#### 3.3.1.1.5 Normalización

Ya que se han obtenido las entidades con sus respectivos atributos e identificadores únicos, es momento de normalizar dicha base de datos, para después poder llevarla a su implementación. Cabe destacar que para la normalización de esta base de datos solo se hará uso de las 3 primeras formas normales, ya que son consideradas como las que garantizan que la base de datos cumple con los requisitos naturales para funcionar óptimamente.

Normalización de la tabla “Camion”

1FN: Eliminar atributos no atómicos y atributos que presentan grupos repetidos en valores de tuplas de una misma relación.

Camion (#id_camion, unidad, posicion)

El atributo #id_camion, sí cumple.

El atributo unidad, no cumple con la segunda condición, ya que se repiten grupos de valores, sin embargo, en la práctica es mejor dejarlo, ya que esto permite identificar a las unidades de las rutas y de cualquier manera de crear una tabla aparte, esta contendría exactamente la misma información (número de unidad) que el identificador (es decir, id_unidad = unidad) y esto sería gastar espacio.

El atributo posicion, sí cumple.

2FN: Eliminar atributos que no dependan funcionalmente de la llave primaria de una misma relación: R.x -> R.y con “x” llave, e “y” otro atributo.

| | |
|---|---|
| Camion.id_camion -> Camion.unidad | Sí cumple |
| Camion.id_camion -> Camion.posicion | Sí cumple |

Por lo cual, la tabla queda como:

Camion (#id_camion, unidad, posicion)

3FN: Eliminar atributos que dependan funcionalmente de otros atributos que no sean llaves, de una misma relación, para que esto se cumpla la notación será:

R.x ¬ R.y con “x” no llave, e “y” no llave.

| | |
|---|---|
| Camion.unidad ¬ Camion.posicion | Sí cumple |
| Camion.posicion ¬ Camion.unidad | Sí cumple |

Por lo cual, la tabla queda como:

Camion (#id_camion, unidad, posicion)

<u>Normalización de la tabla “Ruta”</u>

1FN: Eliminar atributos no atómicos y atributos que presentan grupos repetidos en valores de tuplas de una misma relación.

Ruta (#id_ruta, nombre, busquedas)

El atributo #id_ruta, sí cumple.

El atributo nombre, sí cumple.

El atributo busquedas, puede no cumplir con la segunda condición ya que es un entero que va a depender del número de veces que una ruta sea buscada, sin embargo, por la implementación es preciso dejarlo así ya que no afecta con el performance.

2FN: Eliminar atributos que no dependan funcionalmente de la llave primaria de una misma relación: R.x -> R.y con “x” llave, e “y” otro atributo.

| | |
|---|---|
| Ruta.id_ruta -> Ruta.nombre | Sí cumple |
| Ruta.id_ruta -> Ruta.busquedas | Sí cumple |

Por lo cual, la tabla queda como:

Ruta (#id_ruta, nombre, busquedas)

3FN: Eliminar atributos que dependan funcionalmente de otros atributos que no sean llaves, de una misma relación, para que esto se cumpla la notación será:

R.x ¬ R.y con “x” no llave, e “y” no llave.

| | |
|---|---|
| Ruta.nombre ¬ Camion.busquedas | Sí cumple |
| Ruta.busquedas ¬ Camion.nombre | Sí cumple |

Por lo cual, la tabla queda como:

Ruta (#id_ruta, nombre, busquedas)

<u>Normalización de la tabla “Conductor”</u>

1FN: Eliminar atributos no atómicos y atributos que presentan grupos repetidos en valores de tuplas de una misma relación.

Conductor (#id_conductor, usuario, contrasena)

El atributo #id_conductor, sí cumple.

El atributo usuario, sí cumple, considerando que el usuario es el correo de único del conductor.

El atributo contrasena, sí cumple.

2FN: Eliminar atributos que no dependan funcionalmente de la llave primaria de una misma relación: R.x -> R.y con “x” llave, e “y” otro atributo.

Conductor.id_conductor -> Conductor.usuario Sí cumple

Conductor.id_conductor -> Conductor.contrasena Sí cumple

Por lo cual, la tabla queda como:

Conductor (#id_conductor, usuario, contrasena)

3FN: Eliminar atributos que dependan funcionalmente de otros atributos que no sean llaves, de una misma relación, para que esto se cumpla la notación será:

R.x ¬ R.y con “x” no llave, e “y” no llave.

Conductor.usuario ¬ Conductor.contrasena Sí cumple

Conductor.contrasena ¬ Camion.usuario Sí cumple

Por lo cual, la tabla queda como:

Conductor (#id_conductor, usuario, contrasena)

Normalización de la tabla “Lugar”

1FN: Eliminar atributos no atómicos y atributos que presentan grupos repetidos en valores de tuplas de una misma relación.

Lugar (#id_lugar, nombre, posición, busquedas)

El atributo #id_lugar, sí cumple.

El atributo nombre, sí cumple.

El atributo posicion, sí cumple.

El atributo busquedas, sí cumple.

2FN: Eliminar atributos que no dependan funcionalmente de la llave primaria de una misma relación: R.x -> R.y con “x” llave, e “y” otro atributo.

| | |
|---|---|
| Lugar.id_lugar -> Lugar.nombre | Sí cumple |
| Lugar.id_lugar -> Lugar.posicion | Sí cumple |
| Lugar.id_lugar -> Lugar.busquedas | Sí cumple |

Por lo cual, la tabla queda como:

Lugar (#id_lugar, nombre, posicion, busquedas)

3FN: Eliminar atributos que dependan funcionalmente de otros atributos que no sean llaves, de una misma relación, para que esto se cumpla la notación será:

R.x ¬ R.y con “x” no llave, e “y” no llave.

| | |
|---|---|
| Lugar.nombre ¬ Lugar.posicion | Sí cumple |
| Lugar.nombre ¬ Lugar.busquedas | Sí cumple |
| Lugar.posicion ¬ Lugar.nombre | Sí cumple |
| Lugar.posicion ¬ Lugar.busquedas | Sí cumple |
| Lugar.busquedas ¬ Lugar.nombre | Sí cumple |
| Lugar.busquedas ¬ Lugar.posicion | Sí cumple |

Por lo cual, la tabla queda como:

Lugar (#id_lugar, nombre, posicion, busquedas)

### 3.4 Diseño de algoritmo de búsqueda

Ahora, el objetivo del algoritmo a diseñar es encontrar la ruta más corta entre una fuente y un destino, en una red de transporte. Pero además, se deben encontrar las rutas que aunque no sean las más cortas, también den solución a cómo llegar de un lugar a otro ya que estas soluciones son igual de importantes para el usuario, y el mejor camino solo representa ser una sugerencia.

Para esto ya se mencionó en el capítulo II el problema del camino más corto y una de las soluciones usando el algoritmo de Dijkstra, el cual consiste en ir explorando todos los caminos más cortos que parten del vértice origen y que llevan a todos los demás vértices; cuando se obtiene el camino más corto desde el vértice origen, al resto de vértices que componen el grado, el algoritmo se detiene.

Una vez dicho esto, es importante definir cada parte del algoritmo para esta aplicación en específico, y siendo que las búsquedas serán realizadas dentro de grafos ponderados (distancia) se definirá a un vértice (punto) como un par de coordenadas (Latitud, Longitud), siendo el punto de origen y el punto destino de este tipo.

Por otro lado, las rutas de transporte se conforman por un conjunto de puntos (Latitud, Longitud), lo que es básicamente un arreglo de pares de coordenadas.

Esta definición se explica en la figura 3.5, donde se dice que los parámetros de entrada al algoritmo son dos puntos (lat, lng) y el algoritmo busca dentro de las rutas, las cuales se conforman por conjuntos de puntos.

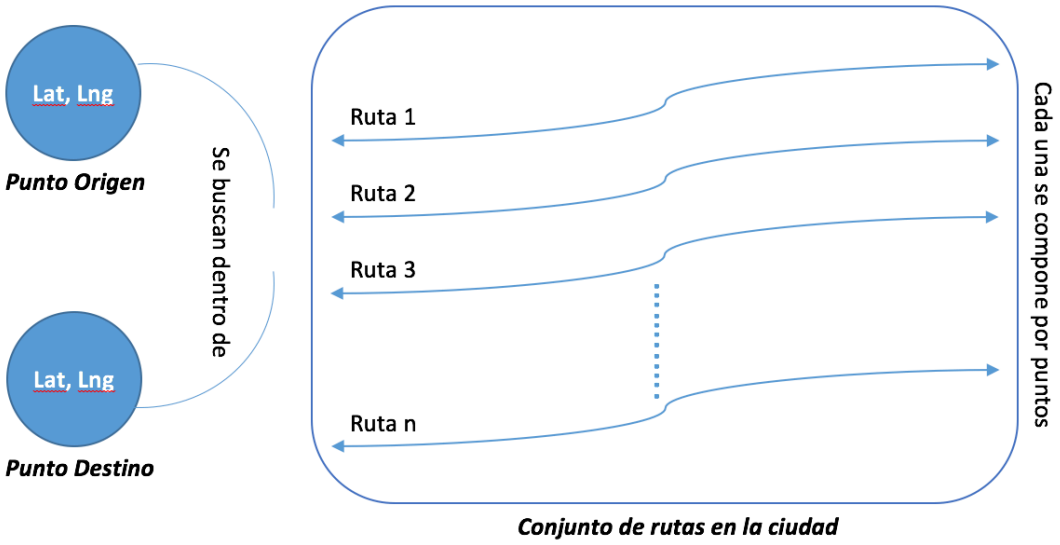



*Figura 3. 5 Definición de los parámetros de entrada y de las rutas*

Observando lo anterior es posible percatarse que una sola ruta puede conducir de A hacia B, pero en caso de que no exista una sola ruta que pueda llevar de A hacia B tendrían que ser más. Dado que la ciudad de Puebla no es tan grande y existen más de 200 rutas, siempre es posible llegar a un destino con tan solo usar 2 rutas.

Por otro lado, muchas veces el usuario requerirá caminar algunos metros para llegar a la parada más próxima de la ruta que está esperando, es por eso que se ha definido buscar cualquier ruta que pase dentro de los 300 metros del origen y dentro de los 300 metros de distancia del destino, puesto que 300 metros es una distancia muy cómoda para cualquier persona, haciendo que no tengan que desperdiciar tanto tiempo caminando (alrededor de 4 minutos caminando).

De igual manera, en caso de que no exista una sola ruta que pueda llevar de A hacia B entonces el algoritmo debe determinar si es posible llegar mediante 2 rutas, y es probable que estas puedan intersecarse en un radio de 300 metros, ya que las personas suelen bajar de un camión y caminar un par de calles para tomar otro.

Tomando en consideración todo lo anterior, el diseño del algoritmo de búsqueda sería el siguiente (**Solucionar( )**):

```
función Cercanos (Ruta rutas[ ], nodo a, nodo b)
    Ruta A_cercanas[ ]
    Ruta B_cercanas[ ]
    entero puntos_A[ ], puntos_B[ ]
    para cada ruta ∈ V[rutas] hacer
            para i = 0 hasta i < ruta.tamaño() hacer
                    si ruta ∉ A_cercanas[ ] entonces
                            si distancia (a, ruta[i]) <= 300 entonces
                                    A_cercanas[].agregar(ruta)
                                    puntos_A[].agregar(i)
                            fin si
                    fin si
                    si ruta ∉ B_cercanas[ ] entonces
                            si distancia (b, ruta[i]) <= 300 entonces
                                    B_cercanas[].agregar(ruta)
                                    puntos_B[].agregar(i)
                            fin si
                    fin si
            fin para
    fin para
fin función


función medirSimple (Ruta ruta)
    entero trayecto = 0
    entero indiceA = puntos_A[ A_cercanas.obtenerIndice(ruta) ]
    entero indiceB = puntos_B[ B_cercanas.obtenerIndice(ruta) ]
    para i = indiceA hasta i < indiceB hacer
            trayecto = trayecto + distancia(ruta[i], ruta[i+1])
    fin para
    return trayecto
```

**fin función**

**función medirDoble** (Ruta ruta_A, Ruta ruta_B, entero int_A, entero int_B)

```
entero trayecto = 0
entero indiceA = puntos_A[ A_cercanas.obtenerIndice(ruta_A) ]
entero indiceB = puntos_B[ B_cercanas.obtenerIndice(ruta_B) ]
para i = indiceA hasta i < int_A hacer
        trayecto = trayecto + distancia(ruta_A[i], ruta_A[i+1])
fin para
para i = int_B hasta i < indice_B hacer
        trayecto = trayecto + distancia(ruta_B[i], ruta_B[i+1])
fin para
return trayecto
```

**fin función**

**función Solucionar** (nodo a, nodo b)

```
Ruta rutas[ ] = red[ ]
Cercanos(rutas[ ], a, b)

Ruta soluciones[ ]
entero distancias[ ]
para cada ruta_A ∈ A_cercanas[ ] hacer
        si B_cercanas[].contiene(ruta_A) = cierto entonces
                soluciones[].agregar(ruta_A)
                distancias[].agregar(medirSimple(ruta_A))
        fin si
fin para
si soluciones[].tamaño() > 0 entonces
        soluciones[].ordenar(distancias[])
        imprimir(soluciones[])
si no entonces
        Rutapar solucion_par[ ]
```

```
        entero punto_AB[ ]
        para cada ruta_A ∈ A_cercanas[ ] hacer
            para cada ruta_B ∈ B_cercanas[ ] hacer
            entero ca = 0
            repetir
                entero cb = 0
                repetir
                    si distancia (ruta_A[ca], ruta_B[cb]) <= 300 entonces
                        solucion_par[].agregar(ruta_A, ruta_B)
                        distancias[].agregar(medirDoble(ruta_A, ruta_B,
ruta_A[ca], ruta_B[cb]))
                    fin si
                    cb++
                hasta que (solucion_par[].contiene(ruta_A, ruta_B) o cb ==
ruta_B.tamaño())
                ca++
            hasta que (solucion_par[].contiene(ruta_A, ruta_B) o ca == ruta_A.tamaño())
            fin para
        fin para
    fin si
    si solucion_par[].tamaño() > 0 entonces
        solucion_par[].ordenar(distancias[])
        imprimir(solucion_par[])
    si no entonces
        imprimir("No hay solución")
    fin si
fin función
```

## 3.5 Diseño de wireframes

Dentro de esta etapa es cuando se lleva a un plano tangible los conceptos y definiciones anteriores, primero en forma de wireframes (plano de pantalla que representa el esqueleto o estructura visual de una aplicación), que permiten crear los primeros prototipos para ser probados con usuarios, y posteriormente, en un

diseño visual acabado con pantallas modelos, que servirá al momento de la programación del código.

Así que lo primero a hacer son los wireframes, y para esto existen algunas herramientas en internet que nos ayudan a lograrlo. Para ello se ha elegido la herramienta Balsamiq la cual facilitará la creación de wireframes en menos tiempo. Esta herramienta permite la facilidad de crear proyectos por un mes gratis, y también desde la web. Así que para usarlo es necesario registrarse en su página: https://balsamiq.com/.

### 3.5.1 Wireframes de usuario de transporte público

De acuerdo a la prioridad establecida en las historias de usuario, es como lo primero a diseñar son los wireframes del usuario de transporte público.

Y de esta manera es posible saber que las funciones de esta app son:

- Buscar una ruta
- Realizar una búsqueda para ir de un lugar a otro
- Visualizar rutas

En la figura 3.6 puede apreciarse el diseño de la pantalla con la funcionalidad de buscar una ruta específica.

De igual manera en la figura 3.7 es posible apreciar la pantalla para realizar una búsqueda para ir de un lugar a otro.

Y finalmente, la figura 3.8 muestra la visualización de las rutas encontradas como solución de la búsqueda.


*Figura 3. 6 Listado de rutas de la ciudad*


*Figura 3.7 Búsqueda para llegar de un lugar a otro*


*Figura 3. 8 Soluciones de la búsqueda*

### 3.5.2 Wireframes del conductor

Ahora es momento de plasmar las funcionalidades de la app del conductor de autobús. Al analizar las historias de usuario es posible percatarse que se requiere de algún inicio de sesión con el fin de conocer qué ruta es la que se está transmitiendo, pues en la ciudad de Puebla hay alrededor de 150 rutas y cada una cuenta con al menos 15 unidades.

Y así es posible percatarse que las funciones de esta app son:

- Iniciar sesión para identificar la ruta (ver figura 3.9)
- Transmitir (ver figura 3.10)


*Figura 3. 9 Inicio de sesión*


*Figura 3.10 Transmisión de recorrido*

### 3.5.3 Wireframes del administrador

Con los datos que son generados por las aplicaciones del usuario y del conductor es posible determinar la demanda de unidades de transporte, y en qué puntos en específico de la ciudad.

Así, es posible darse cuenta que las funciones de esta app son:

- Conocer y visualizar los lugares más buscados por los usuarios (figura 3.11)
- Conocer las rutas de transporte público más buscadas (figura 3.12)


*Figura 3. 11 Lugares más buscados*

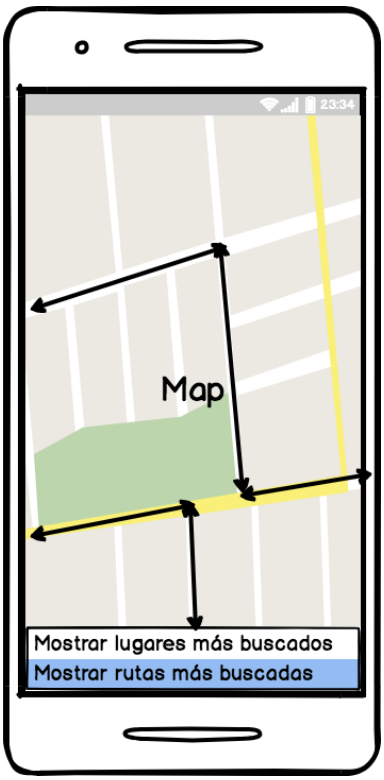



*Figura 3.12 Rutas más buscadas*

Estos wireframes que se han creado tomando en cuentas las funcionalidades de cada parte de la plataforma, nos permiten crear un prototipo visual de lo que será cada app final.

## 3.6 Diseño de prototipo

Ahora es momento de realizar un diseño que unifique las funcionalidades de cada parte de la plataforma tomando en consideración que en caso de requerir cambios futuros, estos no intervengan con el diseño actual de manera significativa. Es preferible diseñar una interfaz simple e intuitiva que pueda ser modificada fácilmente, pues esto asegura un desarrollo ágil.

Un aspecto importante a ser tomado en cuenta son las directrices de diseño, ya que hasta hace unos años cada aplicación que se desarrollaba era sin guías de diseño establecidas (todavía se desarrollan muchas), ahora existen principios de diseño que mejoran la experiencia de usuario y pertenecen al llamado Material Design.

Como primer análisis es posible observar que:

- Las funciones principales de cada aplicación deben ser fácilmente accesibles a los usuarios cuidando la navegación sobre los elementos.
- Es necesario cuidar el no sobrecargar de elementos la pantalla, pues podría confundir a los usuarios, haciendo que no sea una aplicación intuitiva.

Dado lo anterior, lo más conveniente es agrupar estas funcionalidades dentro de un contenedor y acceder a ellas a manera de menú. De esta manera solo habría un espacio que sería de uso exclusivo para ingresar información y para mostrarla, cambiando de acuerdo a las solicitudes del usuario.

### 3.6.1 Prototipo de interfaz de usuario de transporte público

Ya que se conocen las funciones que corresponden a la aplicación del usuario de transporte público, pueden ser priorizadas en cuanto a la usabilidad que tendrían. Es decir, si el usuario entra a la app, ¿para qué sería?

Inmediatamente es posible percatarse que lo más seguro es que quiera saber cómo llegar de un lugar a otro. Una vez ejecutada una búsqueda deberán

mostrarse los resultados automáticamente. Además, la interfaz necesita proveer al usuario la capacidad de navegar a través de las diferentes opciones en cualquier momento, sin necesidad de rehacer una búsqueda.

Para desarrollar este prototipo de interfaz al igual que a la hora de diseñar los wireframes, existen herramientas para hacerlo. Una de ellas es proto.io y las razones para esta elección se basan en las siguientes características:

- Permite crear prototipos completamente interactivos que se ven y trabajan exactamente como debería hacerlo la app.
- Rapidez en el proceso de diseño ya que permite utilizar los componentes de interfaz de usuario nativos de iOS y Android además de permitir importar diseños previamente hechos en Sketch o Photoshop.
- Uso de eventos touch, con el fin de crear el flujo de la app con transiciones de pantalla automatizadas así como la posibilidad de animar cualquier capa.
- Vista previa del prototipo sobre cualquier dispositivo móvil.

Al igual que la herramienta anterior (Balsamiq) proto.io permite su uso gratis por 15 días. Y solo es necesario registrarse en su página para poder tener acceso a la versión de prueba, la cual no limita a los usuarios en cuanto a las funcionalidades.

Una vez analizado y considerado lo anterior, se presenta lo que simplifica este análisis dentro de una interfaz. Comenzando por una pantalla de inicio como se ve en la figura 3.13, después una pantalla principal como se muestra en la figura 3.14, permitiendo realizar búsquedas (figura 3.15) y ver los resultados (figura 3.16), además de permitir explorar el listado de rutas de la ciudad de Puebla como se observa en la figura 3.17.


*Figura 3. 13 Splash de la app*


*Figura 3.14: Pantalla principal*

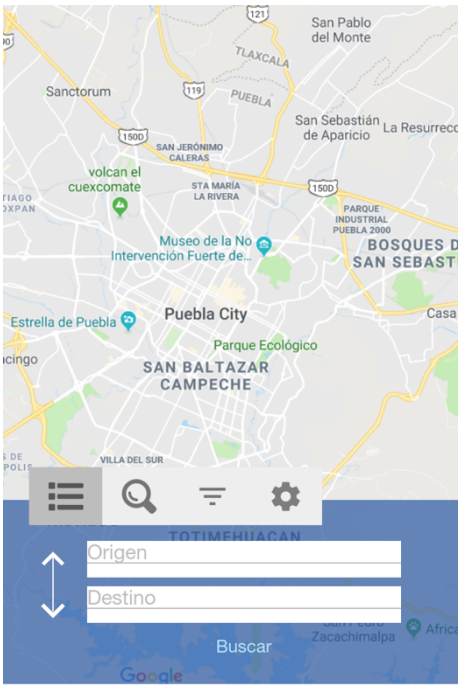



*Figura 3. 15 Búsqueda para llegar de un lugar a otro*

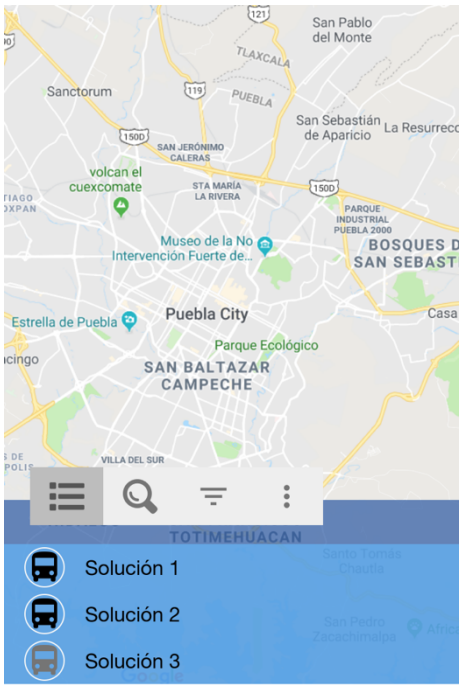



*Figura 3.16: Resultados de la búsqueda*

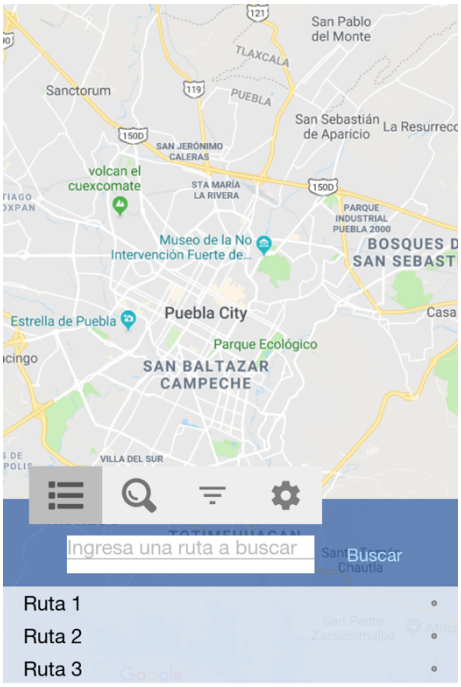



*Figura 3. 17 Listado general de las rutas de la ciudad de Puebla*

### 3.6.2 Prototipo de interfaz del conductor

La interfaz del conductor es muy simple ya que solo requiere de transmitir su trayecto, sin embargo, es necesario conocer qué ruta es la que está transmitiendo. Para lograr identificar la ruta, bastará con hacer una autenticación de usuario. A un usuario se le puede asociar una unidad de una ruta, y con esto se puede conocer la ruta que transmite.

Para esto se requiere una pantalla de acceso o log in, y una vez hecha la autenticación el conductor puede comenzar a transmitir el trayecto de la unidad de transporte público que maneja.

Los datos de la pantalla de autenticación se pueden observar en la figura 3.18, y la pantalla de inicio de recorrido puede observarse en la figura 3.19.

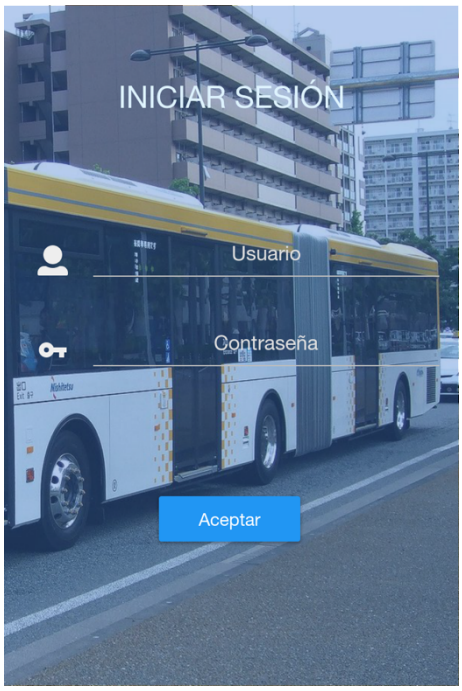



*Figura 3. 18 Autenticación del conductor*

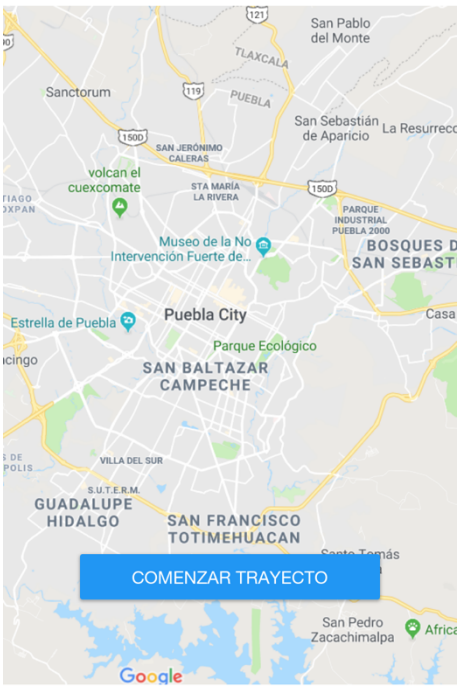



*Figura 3.19: Inicio de transmisión del recorrido*

### 3.6.3 Prototipo de interfaz del administrador

Esta interfaz concentra la información de las rutas más buscadas así como los lugares más buscados por los usuarios de transporte público. Los lugares más buscados deben aparecen como marcadores sobre el mapa, tal como se muestra en la figura 3.20 y las rutas más buscadas se muestran como poli líneas sobre el mapa tal como se muestra en la figura 3.21.

Una vez que se tienen los lugares más buscados es posible analizar si alguna de las rutas de la ciudad pasa cerca de esos puntos, con el objetivo de proponer alguna modificación sobre esa ruta y así mejorar el servicio para los pasajeros. En caso de haber muchos marcadores y de no existir una ruta que pase cerca de los puntos se puede pensar en proponer alguna nueva ruta de transporte público.

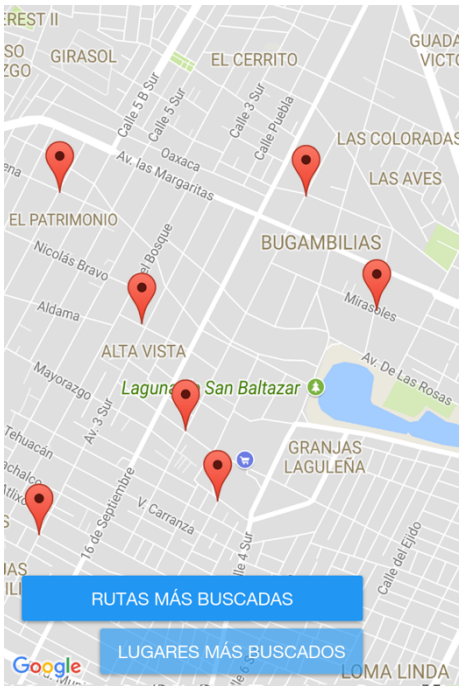



*Figura 3. 20 Visualización de los lugares más buscados*

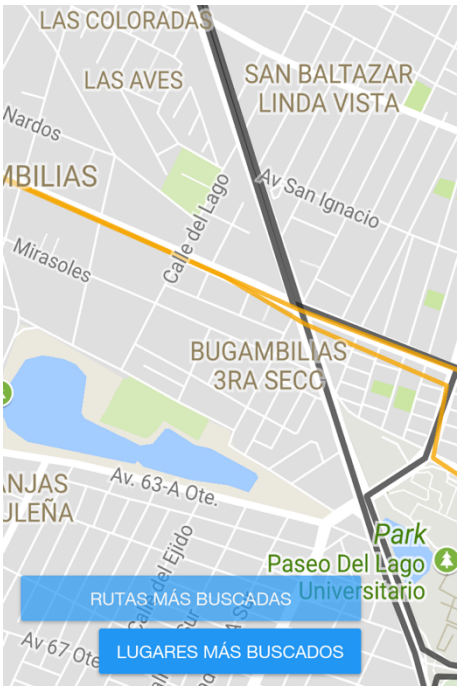



*Figura 3.21: Visualización de las rutas más buscadas*

# CAPÍTULO IV

# IMPLEMENTACIÓN

En XP, las tarjetas de historia de usuario que representan los requerimientos se han descompuesto en tareas, y éstas son la principal unidad de implementación. Así, cada tarea genera una o más pruebas de unidad, que verifican la implementación descrita en dicha tarea. Sin embargo, antes de pasar a la implementación de las aplicaciones móviles, es necesario implementar la base de datos que ya se ha modelado en el capítulo anterior.

## 4.1 Implementación de la Base de Datos

La transformación del diseño lógico de la BD a nuestro modelo físico, es mediante la elaboración de un script de instrucciones MySQL, que nos proporcionan la base de datos con todas las características diseñadas en el capítulos anterior.

Para ello utilizaremos MySQL Workbench, en esta herramienta podremos llevar el diseño de la base de datos a la implementación. Así que lo primero es diseñar las tablas, tal como se muestra en la figura 4.1. Esta figura muestra la conceptualización del problema en su primera etapa.

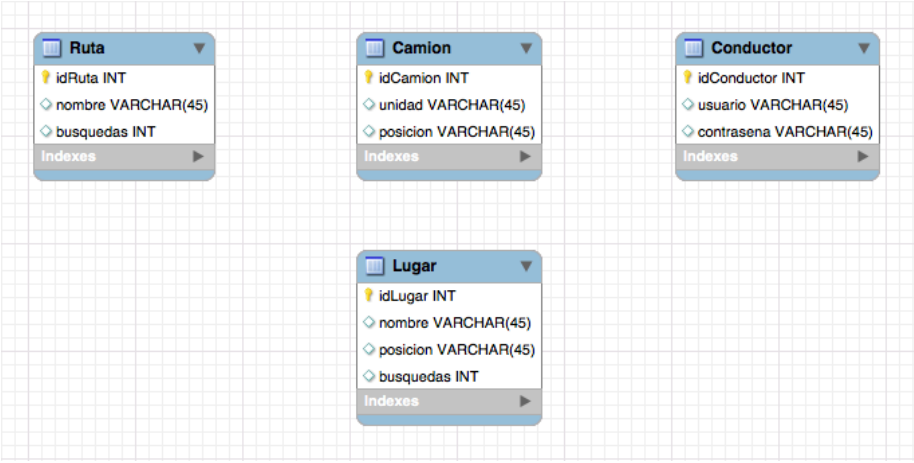



*Figura 4. 1 Creación de las tablas de la base de datos*

A continuación se generan las relaciones pertenecientes al modelo entidad relación, mismas que ya se establecieron en el capítulo anterior. La implementación de esta etapa se muestra en la figura 4.2.

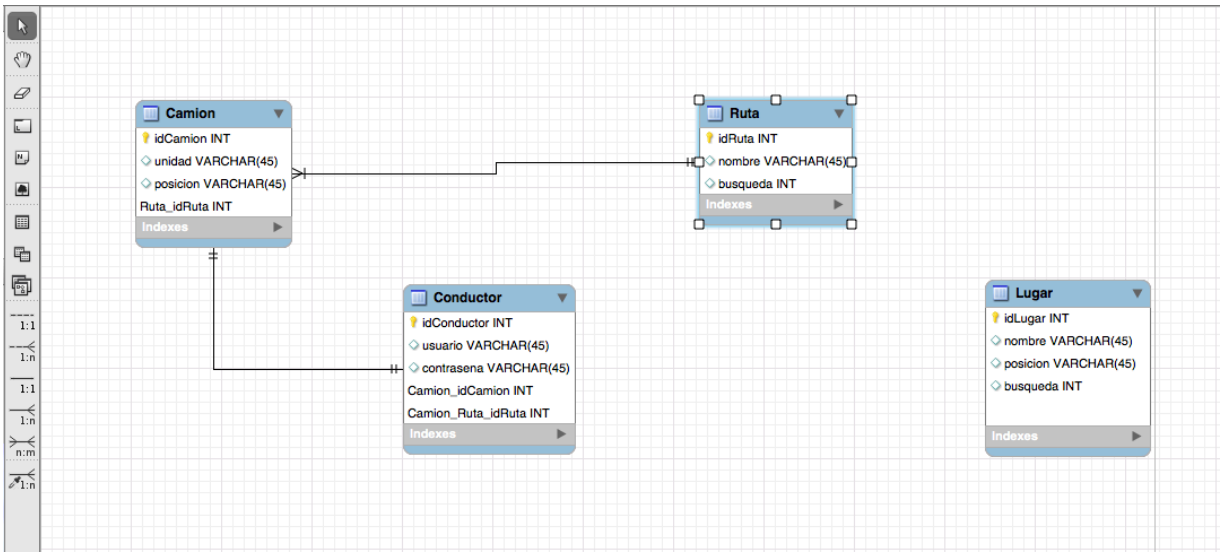



*Figura 4. 2 Relaciones de las tablas*

Cuando tenemos el modelo entidad-relación de la base de datos, es momento de exportarlo como un script de instrucciones SQL, este script se muestra a continuación:

```
CREATE SCHEMA IF NOT EXISTS `transport` DEFAULT CHARACTER SET utf8 ;
USE `transport` ;

-- -----------------------------------------------------
-- Table `transport`.`Ruta`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Ruta` (
  `idRuta` INT NOT NULL,
  `nombre` VARCHAR(45) NULL,
  `busquedas` INT NULL,
  PRIMARY KEY (`idRuta`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `transport`.`Camion`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Camion` (
  `idCamion` INT NOT NULL,
  `unidad` VARCHAR(45) NULL,
  `posicion` VARCHAR(45) NULL,
  PRIMARY KEY (`idCamion`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Conductor`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Conductor` (
  `idConductor` INT NOT NULL,
  `usuario` VARCHAR(45) NULL,
  `contrasena` VARCHAR(45) NULL,
  PRIMARY KEY (`idConductor`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Lugar`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Lugar` (
  `idLugar` INT NOT NULL,
  `nombre` VARCHAR(45) NULL,
  `posicion` VARCHAR(45) NULL,
  `busquedas` INT NULL,
  PRIMARY KEY (`idLugar`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Ruta`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Ruta` (
  `idRuta` INT NOT NULL,
  `nombre` VARCHAR(45) NULL,
  `busquedas` INT NULL,
  PRIMARY KEY (`idRuta`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Camion`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Camion` (
  `idCamion` INT NOT NULL,
  `unidad` VARCHAR(45) NULL,
  `posicion` VARCHAR(45) NULL,
  PRIMARY KEY (`idCamion`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Conductor`
-- -----------------------------------------------------
```

```
CREATE TABLE IF NOT EXISTS `transport`.`Conductor` (
  `idConductor` INT NOT NULL,
  `usuario` VARCHAR(45) NULL,
  `contrasena` VARCHAR(45) NULL,
  PRIMARY KEY (`idConductor`))
ENGINE = InnoDB;

-- -----------------------------------------------------
-- Table `transport`.`Lugar`
-- -----------------------------------------------------

CREATE TABLE IF NOT EXISTS `transport`.`Lugar` (
  `idLugar` INT NOT NULL,
  `nombre` VARCHAR(45) NULL,
  `posicion` VARCHAR(45) NULL,
  `busquedas` INT NULL,
  PRIMARY KEY (`idLugar`))
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

## 4.2 Implementación de la API de Google Places

Usando la API de Google Places dentro de las aplicaciones, permite construir apps que responden contextualmente a los negocios locales y otros lugares cerca del Smartphone. Esto significa que se pueden construir apps enriquecidas basadas en lugares que significan algo a los usuarios.

Google dice que un lugar es un espacio físico que tiene un nombre. Otra forma de pensar sobre un lugar es que es algo que puedes encontrar en el mapa. Los ejemplos incluyen negocios locales, puntos de interés y ubicaciones geográficas. En la API, un lugar está representado por la interfaz Place; y esta incluye información como el nombre del lugar y su dirección, ubicación geográfica, ID del lugar, número de teléfono, tipo de lugar, URL del sitio web, y más.

La característica que nos interesa de la API es que facilita el ingreso de nombres de sitios y direcciones: autocompleta las consultas de los usuarios a medida que escriben. Esto se logra gracias al uso del widget de UI de autocompletado o llamando a la API para predicciones de sitios. Así que, lo que nos interesa del lugar es su nombre y la ubicación geográfica (lat, lng).

Para poder hacer uso de esta API es necesario configurar el proyecto en Android, cubriendo las siguientes características:

- El proyecto de desarrollo de la aplicación debe incluir Google Play Services.
- Agregar la clave de la API, misma que se obtiene registrando el proyecto que se va a desarrollar dentro de la Google API Console mediante el certificado digital del cual se tiene la clave privada.
- Agregar la clave al manifiesto de la aplicación.

Así que para obtener la huella digital del certificado de depuración se debe ingresar a la terminal del proyecto la siguiente instrucción, tal como se muestra en la figura 4.3:

```
keytool -list -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android -keypass android
```

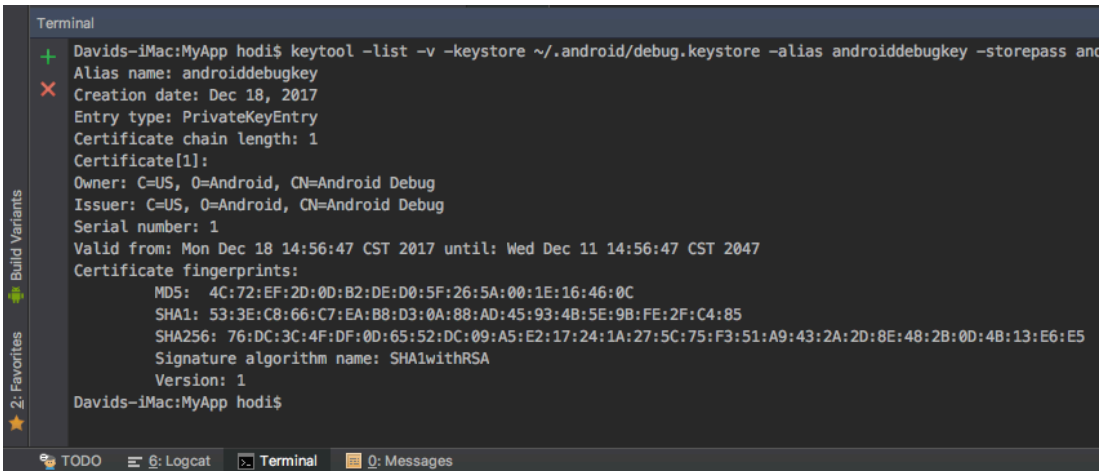



*Figura 4. 3 Obtención de huella digital del certificado de la aplicación*

En la línea que comienza con SHA1, se incluye la huella digital SHA-1 del certificado. La huella digital es una secuencia de 20 números hexadecimales de dos dígitos separados por dos puntos.

Lo siguiente a hacer es la activación de la API de Google Places dentro de nuestra cuenta de desarrollador de Google, y obtener la clave de API para nuestro proyecto. Para esto se ingresa la huella digital de certificado y el nombre de paquete de nuestro proyecto de Android (este último se encuentra al inicio de cada clase .java de nuestro proyecto). Este paso se muestra en la figura 4.4

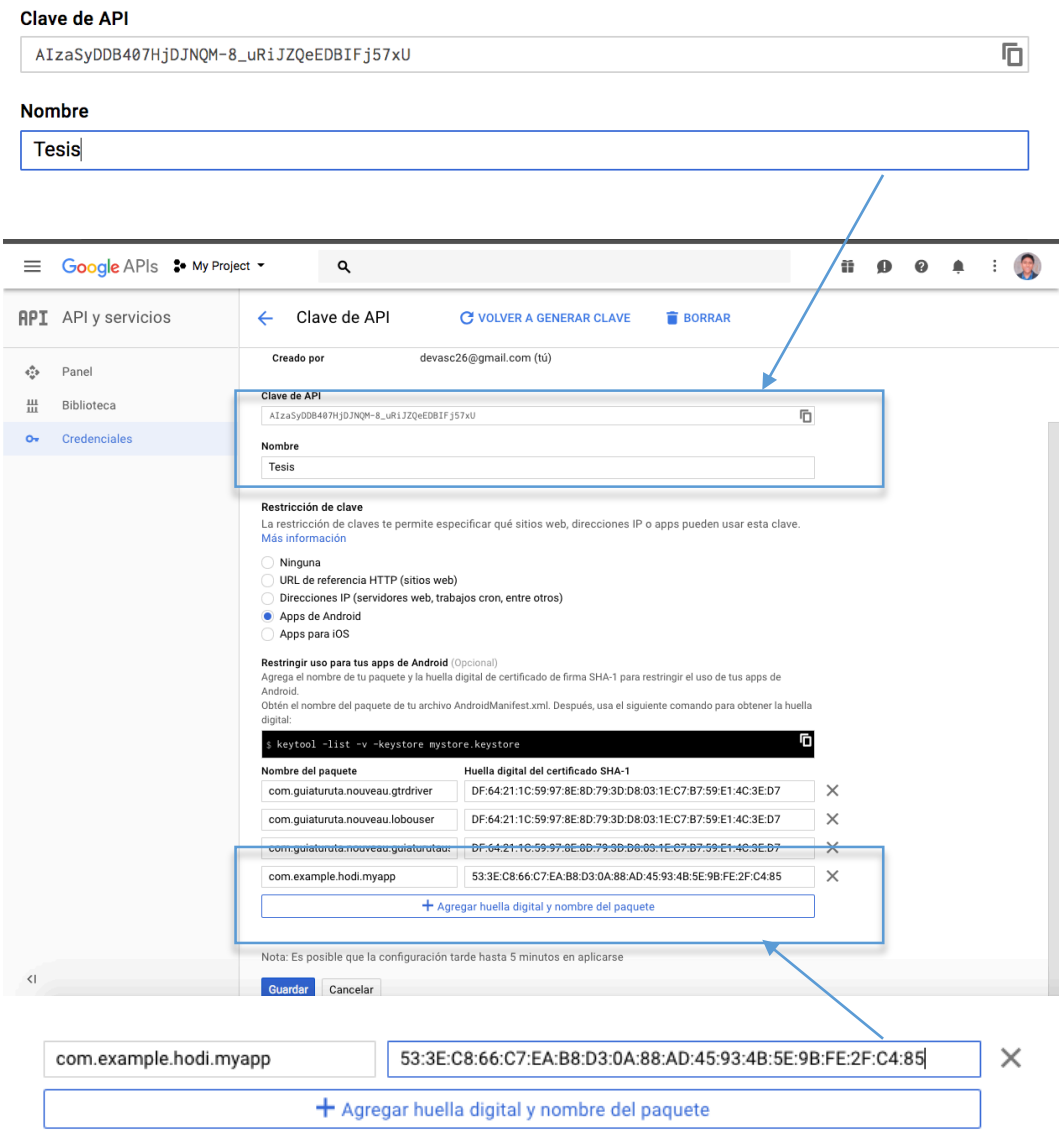



*Figura 4. 4 Obtención de la clave de la API*

Ahora que ya se hizo la obtención de la clave de la API, es momento de agregarla al manifiesto del proyecto de la aplicación, con las siguientes líneas, tal como se muestra en la figura 4.5:

```
<meta-data
        android:name="com.google.android.geo.API_KEY"
        android:value="@string/google_maps_key" />
```

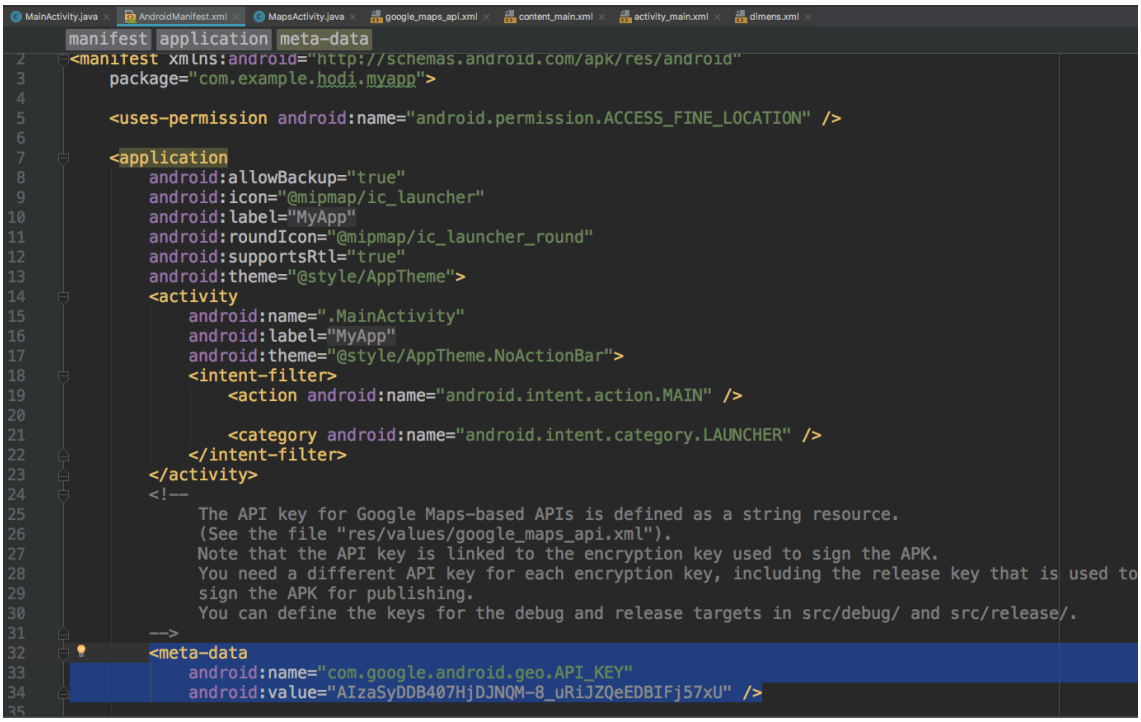



*Figura 4. 5 Agregación de la clave de la API al manifiesto*

Una cosa a tomar en consideración es que se requiere limitar las búsquedas a una región específica (Ciudad de Puebla) ya que su base de datos posee información de lugares de todo el mundo, y para evitar que el usuario pueda confundir el lugar que busca con uno de nombre parecido, es que se debe delimitar la búsqueda.

Esto se logra llamando al método **setBoundsBias** en la instancia del objeto IntentBuilder que es el que especifica al Auto Completado de la API.

El método **setBoundsBias** debe recibir un objeto de tipo LatLngBounds el cual posee las coordenadas de los limites laterales, en este caso son los límites de la ciudad de Puebla. Y las coordenadas en el método quedarían de la siguiente manera:

**setBoundsBias**(new LatLngBounds(new LatLng(19.041292, -98.330306), new LatLng(19.056869, -98.039168))

Una vez hecha la configuración y habiendo implementado el servicio de la API dentro de la app, estos son los resultados, mismos que se muestran en la figura 4.6.

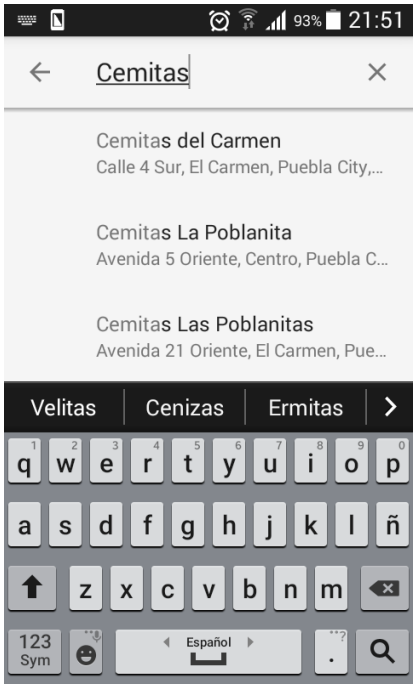



*Figura 4. 6 Implementación y validación de la API de Google Places*

Como es posible observar, al ir ingresando caracteres dentro del objeto de Auto Complementado que provee la API, los resultados que va encontrando están limitados a la región de Puebla, mitigando así el riesgo de cometer un error al seleccionar un lugar.

## 4.3 Obtención de la coordenada del sitio y marcador sobre el mapa

Como se mencionó anteriormente, un objeto del tipo Place tiene un nombre y una ubicación geográfica, entre otros atributos. Lo que sigue es obtener esa geo coordenada y colocar un marcador sobre el mapa.

Así que dentro del método que se ha hecho para cuando se comienza a escribir en el campo de Auto Completado, en cuanto el usuario seleccione un lugar de la lista de opciones, servirá el nombre de ese lugar para obtener su geo coordenada. Una vez que se obtiene esta, se crea un objeto de tipo **Marker** y se muestra sobre el mapa.

La figura 4.7 muestra la implementación y validación de esta etapa del proyecto:


*Figura 4. 7 Obtención del lugar y colocación sobre el mapa*

Y el código para este marcador es muy simple, tan solo necesita las coordenadas y el nombre del marcador, que en este caso es el nombre del sitio.

## 4.4 Listado de rutas de transporte público

Siempre resulta de mucha ayuda tener un listado de todas las rutas de transporte público de la ciudad, pues es una opción fácil para conocer más acerca

de una ruta en específico. Ya que representa una funcionalidad básica y no tan compleja, es que se requiere su implementación en este punto.

En base a las rutas (definidas como un conjunto de puntos lat, lng) en archivo xml, que proveen los datos abiertos del gobierno de México a través de su página https://datosabiertos.pueblacapital.gob.mx/login es que se hará este listado de rutas. Pero antes, es preciso manejar la información que se encuentra dentro de este archivo, para que pueda ser leída de una manera más rápida y efectiva.

Almacenar estas rutas dentro de una base de datos en la aplicación haría lento el proceso de consulta y análisis de los datos. Hay otra forma en que podría ser más rápido el acceso a los datos, siendo que estos residan en un archivo de texto, aunque para lograrlo habría que diseñar métodos de lectura para ese archivo.

Afortunadamente dentro de Android existen los recursos, estos son archivos que permiten almacenar listas de datos (numéricos, de texto, colores, entre otros), y lo hace a través de archivos xml, usando la siguiente sintaxis:

```
<resources>
   <string-array name="string_array_name">
      <item>text_string</item>
   </string-array>
</resources>
```

La etiqueta resources denota el nodo raíz del archivo

La etiqueta string-array denota un nombre para el array, y es usado como el identificador de recurso para referenciar al array.

La etiqueta ítem denota a un string que pertenece a este array.

Dada la sintaxis para estos recursos, es que conviene usarla para almacenar las rutas de transporte público. Pues además de poseer la estructura en que serán almacenadas, facilitan el acceso a los datos desde la aplicación, ya que son recursos de la misma.

Para ello las rutas serán almacenadas de la siguiente forma:

```
<resources>
   <string-array name="Route_0">
      <item>-98.1472860534399</item>
      <item>19.049387763742</item>
      <item>-98.1473250923376</item>
      ...
   </string-array>
   <string-array name="Route_1">
      <item>-98.1790307663373</item>
      <item>19.0816744390375</item>
      <item>-98.179485992289</item>
      ...
   </string-array>
   ...
   ...
   ...
 </resources>
```

Y para acceder a ellas basta con hacerlo a través del identificador de recurso. Una vez que el usuario selecciona una ruta del listado basta con tomar el array de los recursos y crear un objeto de tipo Polyline, el cual se compone por un conjunto de coordenadas. Entonces se agrega al mapa, representando así la ruta de transporte público indicada, tal como puede verse en la figura 4.11.

De igual manera es conveniente colocar un buscador de rutas, para facilitar al usuario esta tarea de consultar una ruta en específico, además de que el campo de texto debe autocompletar mientras se va escribiendo, tal como se observa en la figura 4.9. Cuando se ha seleccionado del campo de autocompletado, y se busca se muestra en inmediatamente en el listado, como se observa en la figura 4.10. Toda esta implementación corresponde al listado de rutas (mismo que se muestra en la figura 4.8)

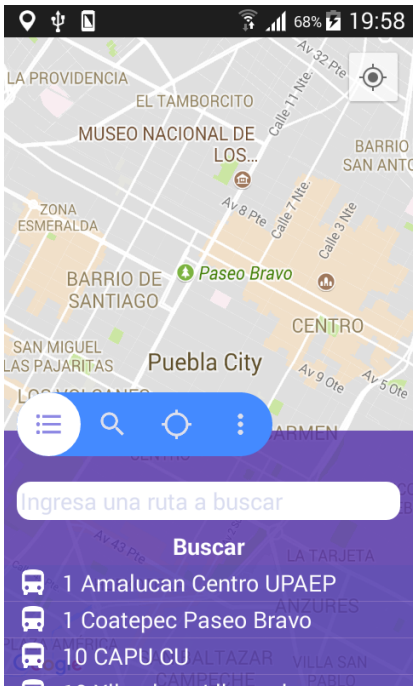



*Figura 4. 8 Listado de rutas de transporte público de Puebla*

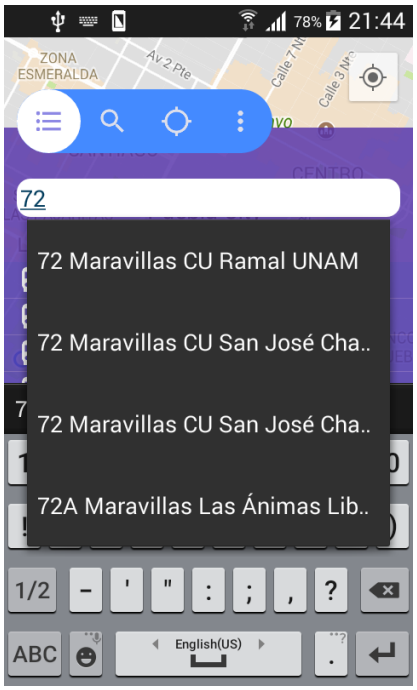



*Figura 4.9: Campo con sugerencia de texto*

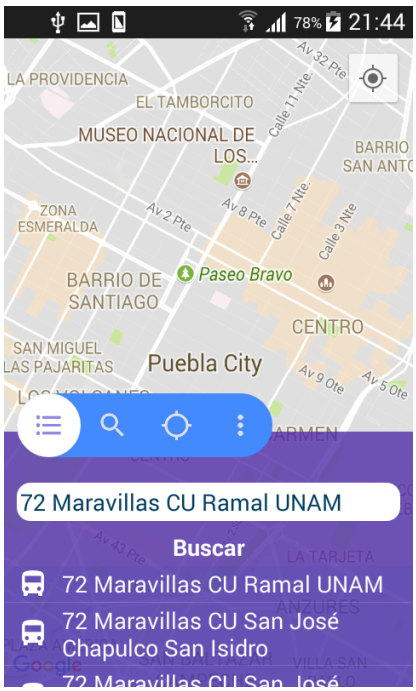



*Figura 4. 10 Búsqueda específica de una ruta*

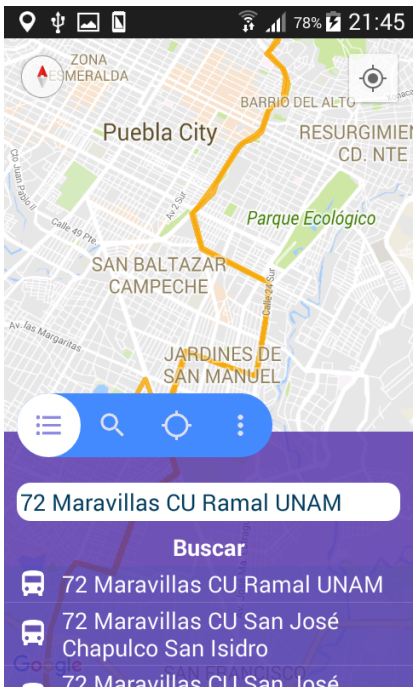



*Figura 4.11: Visualización de una ruta seleccionada*

## 4.5 Búsqueda para llegar de un lugar a otro

Esta representa la parte más importante para el usuario de transporte público, ya que le permite tener un abanico de opciones para poder ir de un punto a otro dentro de la ciudad. Y es aquí donde el algoritmo de búsqueda que se ha diseñado en el capítulo anterior toma gran importancia dentro de la aplicación, pues no sólo nos va a permitir encontrar el mejor camino, sino todos los posibles caminos para la movilización urbana.

Aunque antes es importante obtener los parámetros de entrada , y estos son tan solo el punto origen y el punto destino (figura 4.13). Para esto basta con repetir lo que se ha implementado en la sección 4.3 y el algoritmo podrá hacer su tarea (ver figura 4.12). Cuando se seleccionen los lugares, deben ser almacenados en la base de datos y en caso de ya existir en la misma se incrementará el contador "*busquedas*".

A continuación se muestran los datos de entrada a través del uso de la API de Google places.

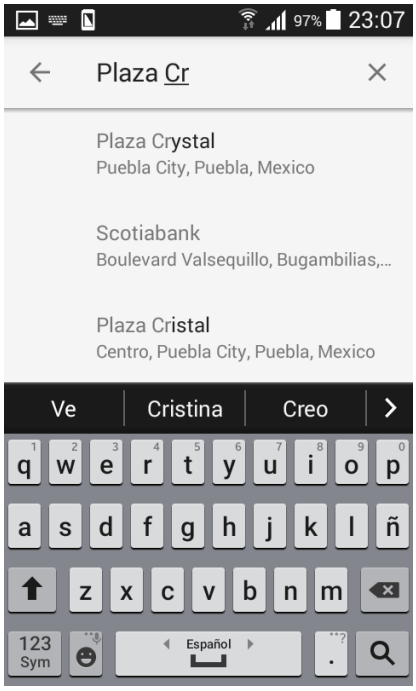



*Figura 4. 12 Búsqueda de un lugar*

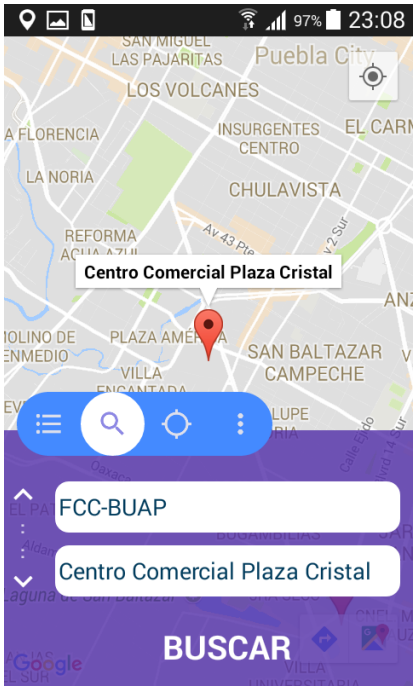



*Figura 4.13: Lugar de partida y destino*

En cuanto al algoritmo, lo que hace es verificar si estos puntos se encuentran en un radio de 300 metros dentro de alguna ruta, de ser así las va almacenando en una lista y al final las ordena en cuanto al camino más corto. En una lista (como la del listado de rutas) es como muestra las soluciones al usuario, haciéndolas accesibles en cada momento a través del menú de opciones, así el usuario puede ir visualizando las diferentes opciones y elegir la que más crea conveniente, tal como se muestra en la figura 4.14.

Una vez que el usuario selecciona alguna de las diferentes soluciones, aparte de dibujar la(s) ruta(s), dentro de la interfaz de encapsulan tanto el punto origen como el punto destino, para que sean más fáciles de identificar, logrando así una interfaz de usuario entendible, ver la figura 4.15.

Todo lo mencionado en el párrafo anterior se muestra en las figuras 4.14 y 4.15:

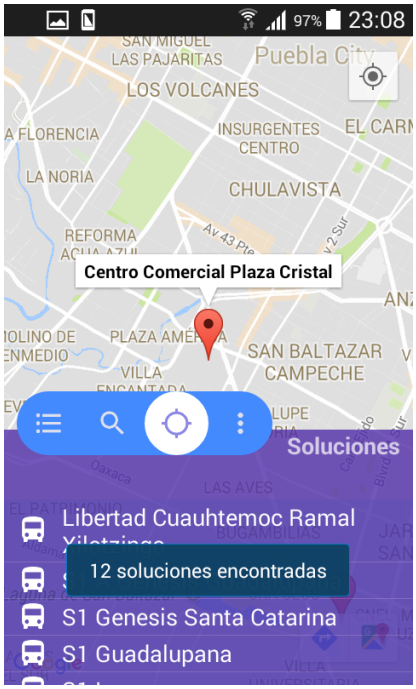



*Figura 4. 14 Rutas encontradas para llegar al destino*

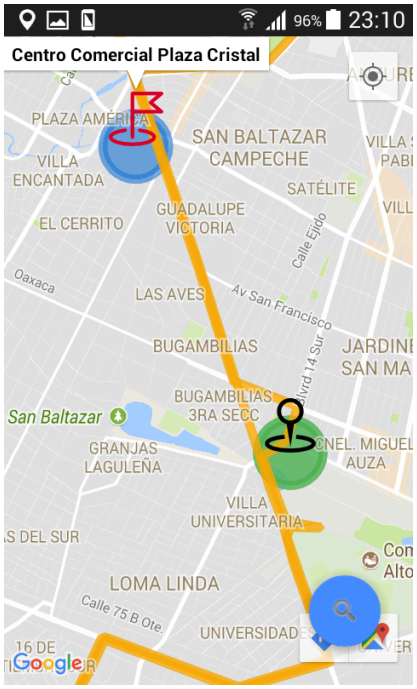



*Figura 4.15: Selección de una ruta encontrada*

## 4.6 Autenticación de conductor

La autenticación del conductor es con el objetivo de identificar la ruta que se va a transmitir. Una ruta es operada por varias unidades (camiones), y para esto se requiere conocer la ruta, así que bastará con una pantalla de acceso que contemple un id de usuario y su contraseña para hacerlo, tal como se muestra en la figura 4.16.

Si los datos introducidos son correctos entonces se le permite pasar a la pantalla donde se comienza el trayecto, tal como se muestra en la figura 4.17 y posteriormente se comienza la transmisión, ver la figura 4.18.

Para la autenticación es necesario conectarse a una base de datos externa donde se almacenan los conductores así como las diferentes rutas en la ciudad. Y esto corresponde a la implementación de la base de datos donde los conductores se guardan en la tabla Conductor y cada conductor conduce un Camión, mismo que pertenece a una Ruta.


*Figura 4. 16 Inicio de sesión de conductor*


*Figura 4.17: Pantalla para comenzar trayecto*


*Figura 4. 18 Transmisión de trayecto*

## 4.7 Visualización en tiempo real

La capacidad de visualizar los camiones en tiempo real es un plus enorme para los usuarios y sobre todo representa una fuerte innovación ya que no hay ciudad alguna donde los usuarios puedan visualizar el trayecto de los camiones en tiempo real.

Esta característica de la app representa una gran ventaja, ya que los usuarios no están con la incertidumbre de a cuánto tiempo y dónde es que está la ruta que esperan.

La manera en que está característica ha sido implementada es en base a un listener que realiza una consulta a la base de datos cada 2 segundos, extrayendo así la posición de cada camión perteneciente a una ruta específica.

A continuación se muestran las capturas de pantalla de esta implementación desde la figura 4.19 a la 4.22, donde se ha tomado de ejemplo a la ruta 54A:

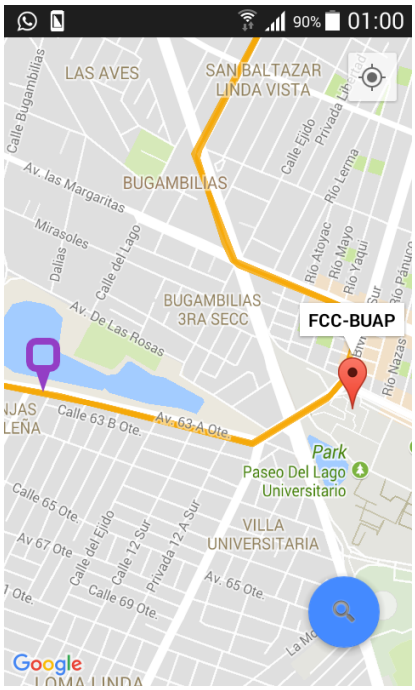



*Figura 4. 19 Captura 1 de visualización*

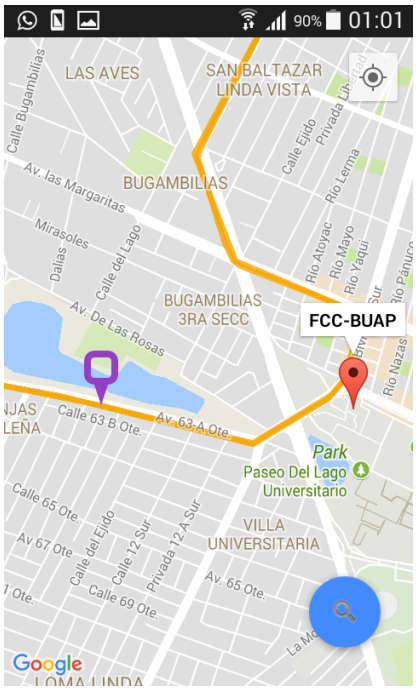



*Figura 4.20: Captura 2 de visualización*

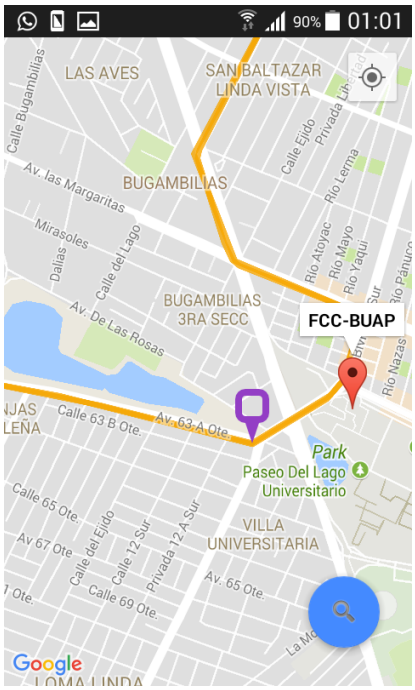



*Figura 4. 21 Captura 3 de visualización*

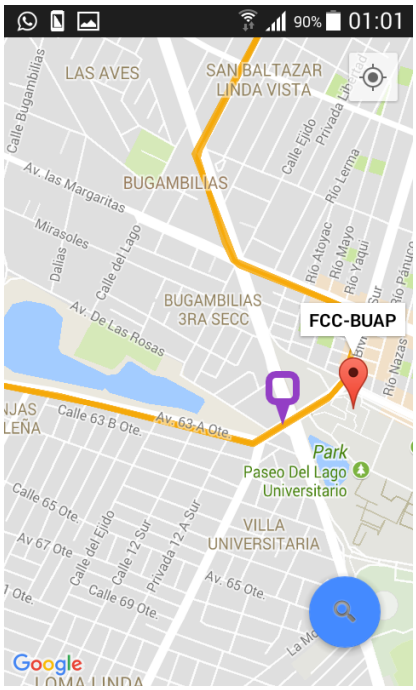



*Figura 4.22: Captura 4 de visualización*

## 4.8 Propuesta de mejoras en las rutas en base a los lugares y rutas más buscados

Finalmente es posible sacar provecho de los datos generados por el uso de las aplicaciones. Cuando los usuarios buscan una ruta en específico es porque les interesa conocer por dónde viene el camión más cercano de esa ruta, significa que es porque la están esperando y ya que si fuera una ruta constante no habría necesidad de conocer a qué distancia se encuentra la más próxima, es importante que las rutas buscadas a menudo puedan cubrir mejor la demanda de los usuarios.

De igual manera, cuando los usuarios buscan lugares específicos significa que pueden ser lugares muy concurridos, como centros comerciales, plazas o universidades. Y aunque en lugares muy concurridos suele haber suficiente transporte muchas veces los caminos van muy saturados, haciendo que los usuarios deban esperar hasta que pase alguno con cupo disponible.

Con estas consideraciones es posible determinar qué rutas no cubren la demanda de usuarios, qué lugares son muy concurridos y de existir rutas que pasen por estos lugares, sería más fácil modificar solo los tramos que se acercan a los lugares concurridos.

Para lograr esto es necesario acceder a la información que se ha generado por el uso de las aplicaciones y que se encuentra almacenada en la base de datos, específicamente la de los campos *"búsqueda"* de las tablas: **Ruta** y **Lugar**. A continuación se muestran ambas tablas (ver figura 4.23) con el fin de recordar sus campos.

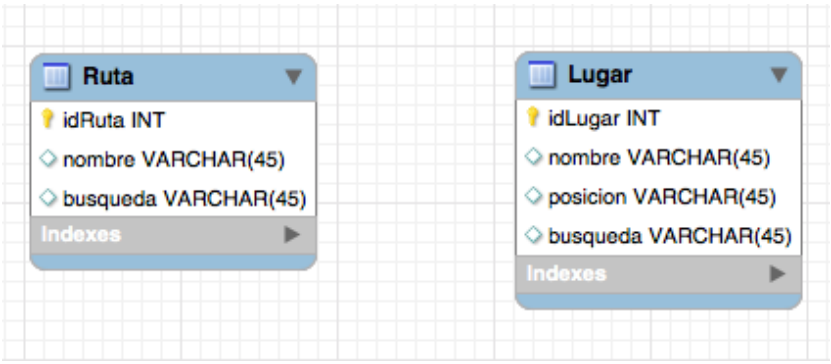



*Figura 4. 23 Vista de las tablas Ruta y Lugar*

La manera de acceder y utilizar esta información es tomando las rutas más buscadas, considerando como las más buscadas aquellas que sobrepasan las 100 búsquedas, al mostrarse dichas rutas al usuario se hace de manera ordenada descendientemente. De igual manera se hace con los lugares, y ambos listados se muestran de forma paralela para que sea más fácil observar y analizar los datos generados. En la figura 4.24 se muestra dicha implementación.

Dentro de un panel, el cual se encuentra en la parte inferior de la pantalla, se listan en la mitad izquierda las rutas más buscadas por los usuarios de transporte público y en la mitad lateral derecha del panel se listan los lugares más buscados por los mismos usuarios. Cuando se selecciona una de las rutas del listado izquierdo se dibuja sobre el mapa dicha ruta y además se colocan todos los marcadores del listado derecho, los marcadores siempre aparecen y lo único que va cambiando es la ruta que se selecciona. Observar la figura 4.25 donde se muestra dicha implementación. De esta manera se puede ir apreciando si alguna ruta pasa cerca de diferentes puntos y de no ser así deja paso a la posibilidad de que pueda existir una nueva ruta que pueda satisfacer la demanda en dichos puntos.

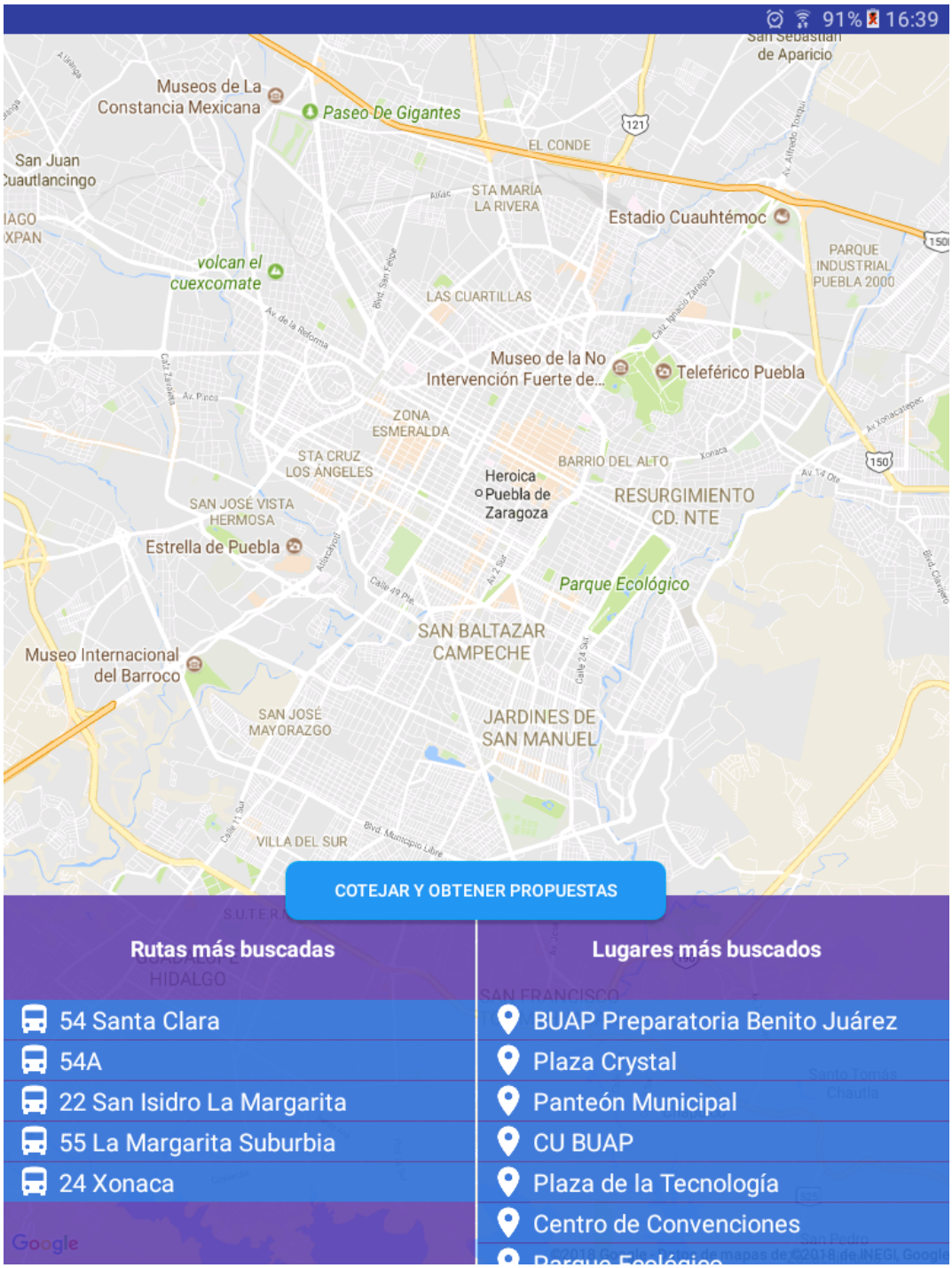



*Figura 4. 24 Vista de las rutas y lugares más buscados*

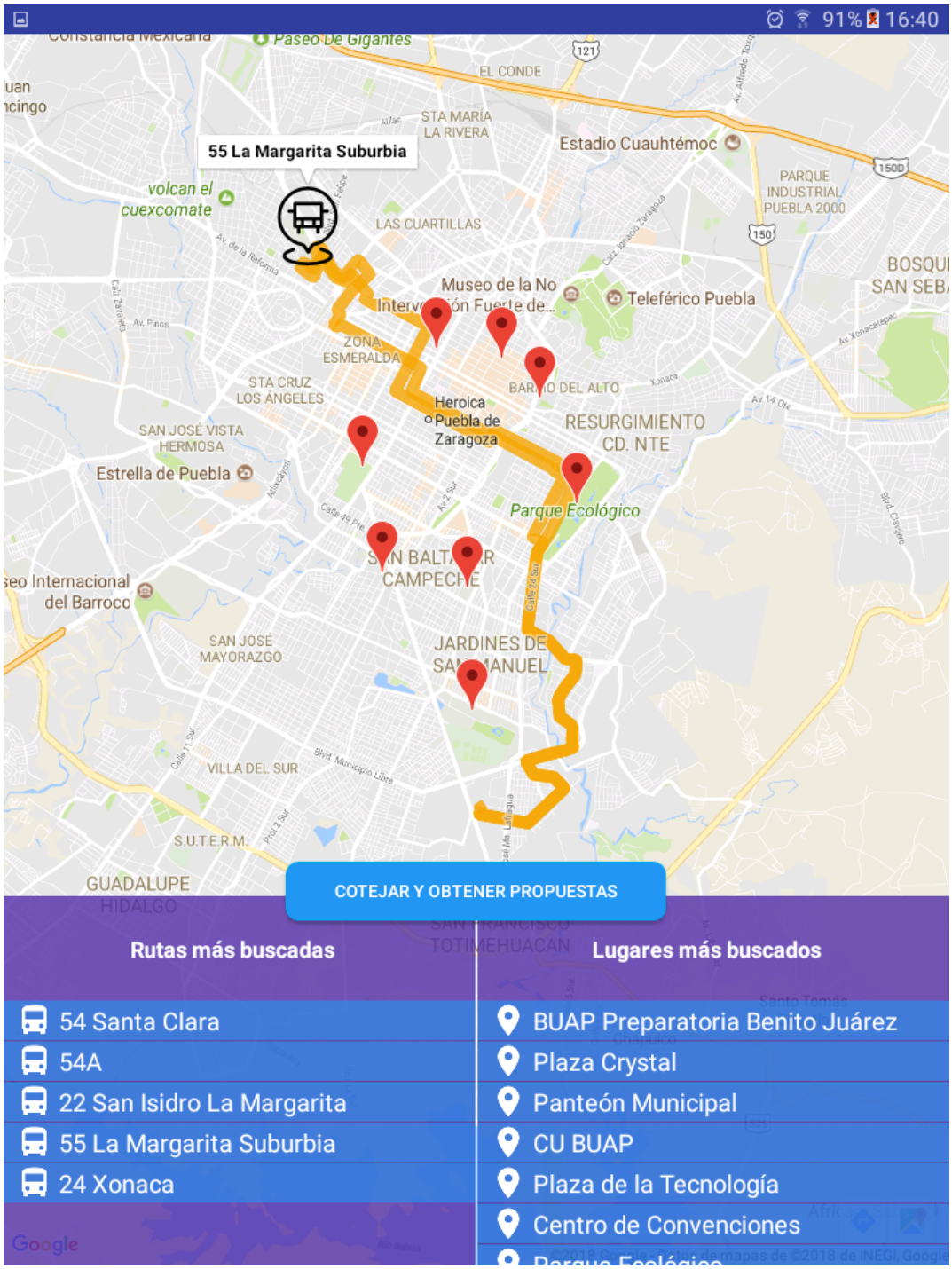



*Figura 4. 25 Vista de la ruta "55" y los lugares más buscados*

# CAPÍTULO V

# EVALUACIÓN

Finalmente para la evaluación del funcionamiento de la plataforma se han aplicado una serie de pruebas que han permitido comprobar tanto la funcionalidad de los módulos como las interfaces de usuario, y gracias a estas se lograron detectar y corregir las fallas que había en el sistema.

De igual manera, dichas pruebas de evaluación demuestran que la plataforma realiza las funciones que se han especificado en los requerimientos iniciales. Esto se ha logrado debido a que se han aplicado dos tipos de pruebas: funcionales y estructurales.

*Estructurales:* son pruebas que se concentran en lo que hay codificado o diseñado a bajo nivel por lo que no es necesario reconocer la especificación de requisitos. Su cometido es comprobar los flujos de ejecución dentro de cada función o módulo.

*Funcionales:* son pruebas que parten de los requisitos funcionales aplicándolas sobre el sistema empleando un determinado conjunto de datos de entrada y observando las salidas que produce, de manera que se contraste con las especificaciones y determinar si está desempeñando correctamente su función.

A continuación se describen algunas de las pruebas realizadas para la plataforma, tanto funcionales como estructurales.

## 5.1 Pruebas estructurales

Para la realización de estas pruebas se eligieron módulos específicos de la plataforma de cada aplicación, se elaboraron casos de prueba y se realizó la verificación de las mismas.

### 5.1.1 Aplicación de usuario

**Módulo**: Buscar un lugar de la ciudad

**Descripción**: Se escribe un lugar cuyo nombre existe en otras partes del mundo, ya que la API de Google Places retorna todas las coincidencias de los lugares el usuario deberá seleccionar una. El módulo limita los márgenes de búsqueda solo dentro de la ciudad de Puebla.

**Resultado esperado**: Sólo se mostrarán las opciones de autocompletado que corresponden a lugares de la ciudad de Puebla.

**Código**:

```
private void openAutocompleteActivity() {
    try {
        // La actividad de AutoComplete require que los Servicios de Google Play
estén instalados. De no ser el caso el constructor lanza una excepción.
        Intent intent = new
PlaceAutocomplete.IntentBuilder(PlaceAutocomplete.MODE_FULLSCREEN).setBoundsBias(new
LatLngBounds(
                new LatLng(19.041292, -98.330306),
                new LatLng(19.056869, -98.039168)))
                .build(this);
        startActivityForResult(intent, REQUEST_CODE_AUTOCOMPLETE);
    } catch (GooglePlayServicesRepairableException e) {
        // Indica que los Servicios de Google Play no están instalados.
        GoogleApiAvailability.getInstance().getErrorDialog(this,
e.getConnectionStatusCode(),
                0 /* requestCode */).show();
    } catch (GooglePlayServicesNotAvailableException e) {
        // Indica que los Servicios de Google Play no están disponibles.
        String message = "Google Play Services is not available: " +
                GoogleApiAvailability.getInstance().getErrorString(e.errorCode);

        android.util.Log.e(TAG, message);
        Toast.makeText(this, message, Toast.LENGTH_SHORT).show();
    }
}
```

**Resultado obtenido**: Mientras el usuario iba escribiendo en el cuadro de búsqueda solo se iban mostrando lugares que se encuentran dentro de la ciudad de Puebla, debido a que en el constructor del cuadro de texto se delimitaron los límites de búsqueda a sólo la ciudad de Puebla.

### 5.1.2 Aplicación de conductor

**Módulo**: Autenticar conductor

**Descripción**: Se ingresa un usuario correcto y se ingresa la contraseña, pero la contraseña no está completa, pues sólo se escribieron los primeros 4 caracteres. Antes de que se establezca la conexión con la Base de datos, el módulo se encarga de validar que se ha ingresado información en ambos campos, pero además se encarga de que la información ingresada cumpla con las características: el usuario debe ser un correo electrónico y la contraseña debe ser mayor a 4 caracteres.

**Resultado esperado**: Ya que no se escribió completamente la contraseña, al querer iniciar sesión antes de establecer la conexión a la base de datos, debe colocarse un mensaje en el campo de la contraseña diciendo que es demasiado corta.

**Código**:

Los siguientes dos métodos verifican la validez de los campos:

```
private boolean isEmailValid(String email) {
    return email.contains("@");
}

private boolean isPasswordValid(String password) {
    return password.length() > 4;
}
```

El siguiente método invoca a los anteriores y se encarga de mostrar los errores en los campos respectivos, así como de conectarse a la base de datos en caso de que la información sea válida.

```
private void attemptLogin() {
    if (mAuthTask != null) {
        return;
    }

    // Reinicia los errores.
    mEmailView.setError(null);
    mPasswordView.setError(null);

    // Almacena los valores al momento que se intenta iniciar sesión.
    String email = mEmailView.getText().toString();
```

```
    String password = mPasswordView.getText().toString();

    boolean cancel = false;
    View focusView = null;

    // Verifica una contraseña válida si es que el usuario ingresó una.
    if (!TextUtils.isEmpty(password) && !isPasswordValid(password)) {
        mPasswordView.setError(getString(R.string.error_invalid_password));
        focusView = mPasswordView;
        cancel = true;
    }

    // Verifica una dirección de email válida.
    if (TextUtils.isEmpty(email)) {
        mEmailView.setError(getString(R.string.error_field_required));
        focusView = mEmailView;
        cancel = true;
    } else if (!isEmailValid(email)) {
        mEmailView.setError(getString(R.string.error_invalid_email));
        focusView = mEmailView;
        cancel = true;
    }

    if (cancel) {
        // Hubo un error, y se enfoca en el primer campo.
        focusView.requestFocus();
    } else {
        // Muestra un spinner de progreso mientras comienza una tarea en Segundo
plano para autenticar al usuario en la base de datos.
        showProgress(true);
        mAuthTask = new UserLoginTask(email, password);
        mAuthTask.execute((Void) null);
    }
}
```

**Resultado obtenido**: Antes de iniciar conexión con la base de datos, la aplicación puso un mensaje de que la contraseña es demasiado corta en su respectivo campo.

### 5.1.3 Aplicación de administrador

**Módulo**: Dibujar ruta más buscada en el mapa

**Descripción**: Del listado de rutas más buscadas se selecciona a la primera de ellas, sin embargo para dibujarla es necesario borrar lo que este actualmente, ya que son muchos puntos. El módulo limpia el mapa y entonces dibuja la ruta seleccionada.

**Resultado esperado**: Ver trazada la ruta seleccionada y además los lugares más buscados, aún si se selecciona otra ruta, estos deben conservarse.

**Código**:

```
lvRutas.setOnItemClickListener(new AdapterView.OnItemClickListener() {
    @Override
    public void onItemClick(AdapterView<?> parent, View view, final int position,
long id) {
        mMap.clear();
        int idr = getResources().getIdentifier("Route_" +
Integer.toString(buscarIndice(rutas.get(position).toString())), "array",
getPackageName());
        String[] latlon = getResources().getStringArray(idr);
        polopt = new PolylineOptions();
        polopt.geodesic(true);
        for(int k=0; k<latlon.length/2; k++) {
            polopt.add(new LatLng(Double.parseDouble(latlon[(k*2)+1]),
Double.parseDouble(latlon[k*2])));
        }
        mMap.addPolyline(polopt.color(0xCCF7A700));
        mMap.addMarker(new MarkerOptions().position(new
LatLng(Double.parseDouble(latlon[1]),
Double.parseDouble(latlon[0]))).icon(BitmapDescriptorFactory.fromResource(R.drawable
.place_pin)).title(rutas.get(position).toString()));
        mostrarLugares();
    }
});
```

**Resultado obtenido**: Se visualiza la ruta trazada sobre el mapa así como los marcadores de todos los lugares más buscados.

## 5.2 Pruebas funcionales

Para la realización de estas pruebas se seleccionaron las funcionalidades a probar de cada aplicación de la plataforma, se elaboraron los casos de prueba correspondientes y se realizó la validación de los resultados obtenidos.

### 5.2.1 Aplicación de usuario

**Función**: Realizar búsqueda para ir de un lugar a otro

**Descripción**: Se ingresa un lugar de partida y un lugar destino, y dentro de una lista se muestran las posibilidades para llegar.

**Resultado esperado**: Una lista de todas las posibilidades para realizar dicho trayecto, que permita visualizar las rutas sobre el mapa cuando sean seleccionadas.

**Vista**:

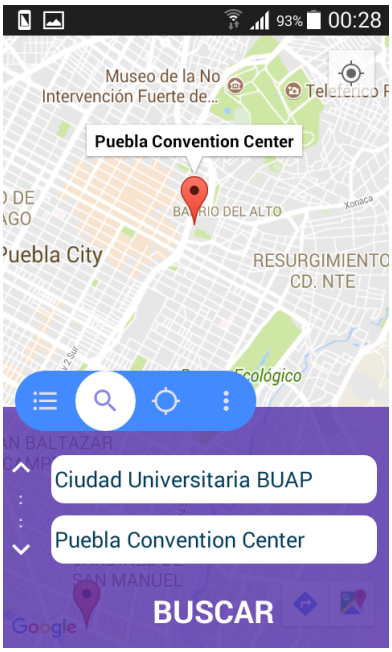



*Figura 5. 1 Inserción de lugares para búsqueda*

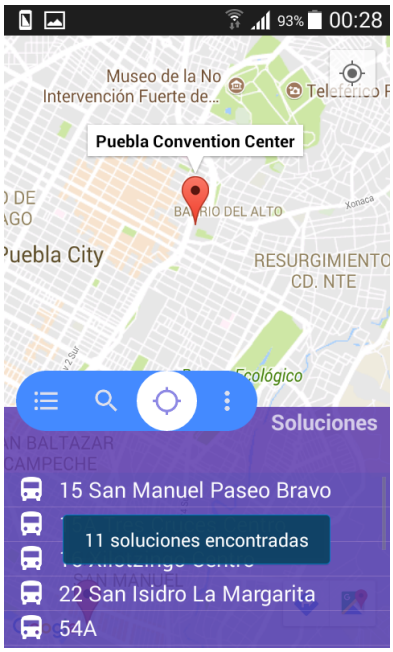



*Figura 5.2: Soluciones*

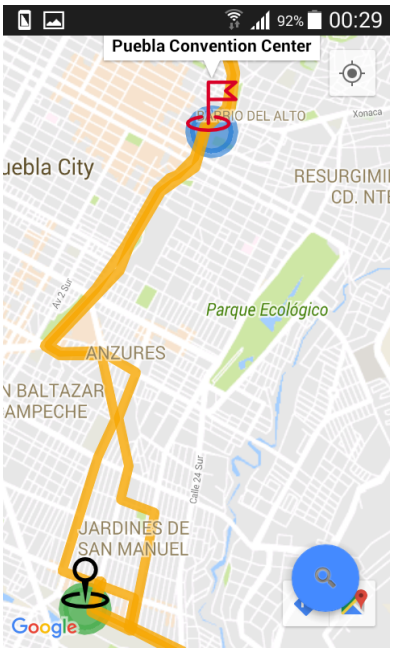



*Figura 5.3: Ruta "Boulevard CU"*

**Resultado obtenido**: Al realizar una búsqueda para ir de Ciudad Universitaria al Centro de Convenciones (figura 5.1), se encontraron 11 rutas para llegar (figura 5.2). Se seleccionó la de "Boulevard CU" y esta se dibujó sobre el mapa, conservando los marcadores del punto de partida y del punto destino (figura 5.3).

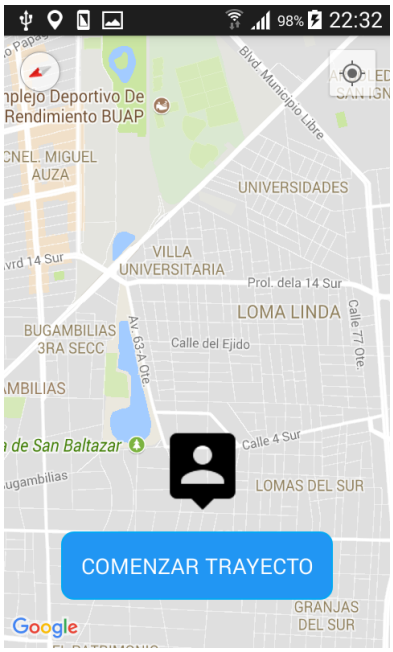
### 5.2.2 Aplicación de conductor

**Función**: Transmisión de trayecto de ruta

**Descripción**: Cuando se inicia el trayecto se hace en base a la información cuando se autenticó el usuario, así se sabe a qué ruta y qué unidad se hace la transmisión de las coordenadas del recorrido.

**Resultado esperado**: Mostrar un marcador en el mapa correspondiente al geo posicionamiento del vehículo actual.

**Vista**:

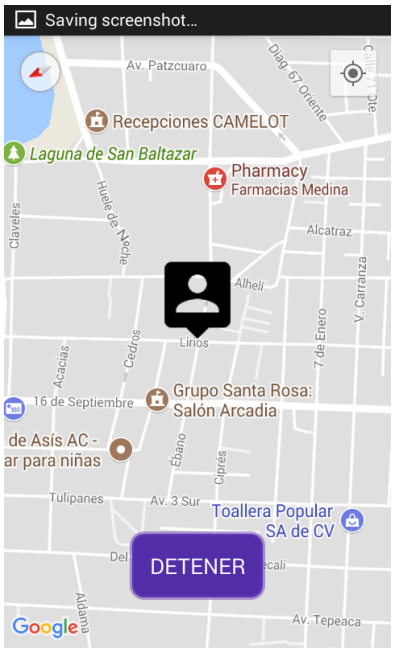




*Figura 5. 4 Inicio de sesión* *Figura 5.5: Pantalla para comenzar trayecto* *Figura 5.6: Transmisión de la ruta*

**Resultado obtenido**: Cuando se inició sesión como conductor (figura 5.4), se obtuvo la ruta y unidad a la que debe transmitir. Entonces se permitió acceder a la pantalla que comienza la transmisión de un trayecto (figura 5.5). Sobre el mapa se muestra un marcador que corresponde a la posición en tiempo real del conductor (figura 5.6).

### 5.2.3 Aplicación de administrador

**Función**: Visualizar propuestas de mejora en rutas

**Descripción**: Se pide cotejar las rutas y lugares más buscados, y de encontrar varios de estos lugares cercanos a una ruta se enlistan como posibles mejoras.

**Resultado esperado**: Dentro de una lista se encuentran las rutas que poseen posibilidades de mejora, y cuando se selecciona una de ellas se dibuja sobre el mapa dicha ruta y los lugares cercanos más demandados por los que puede pasar.

**Vista**:

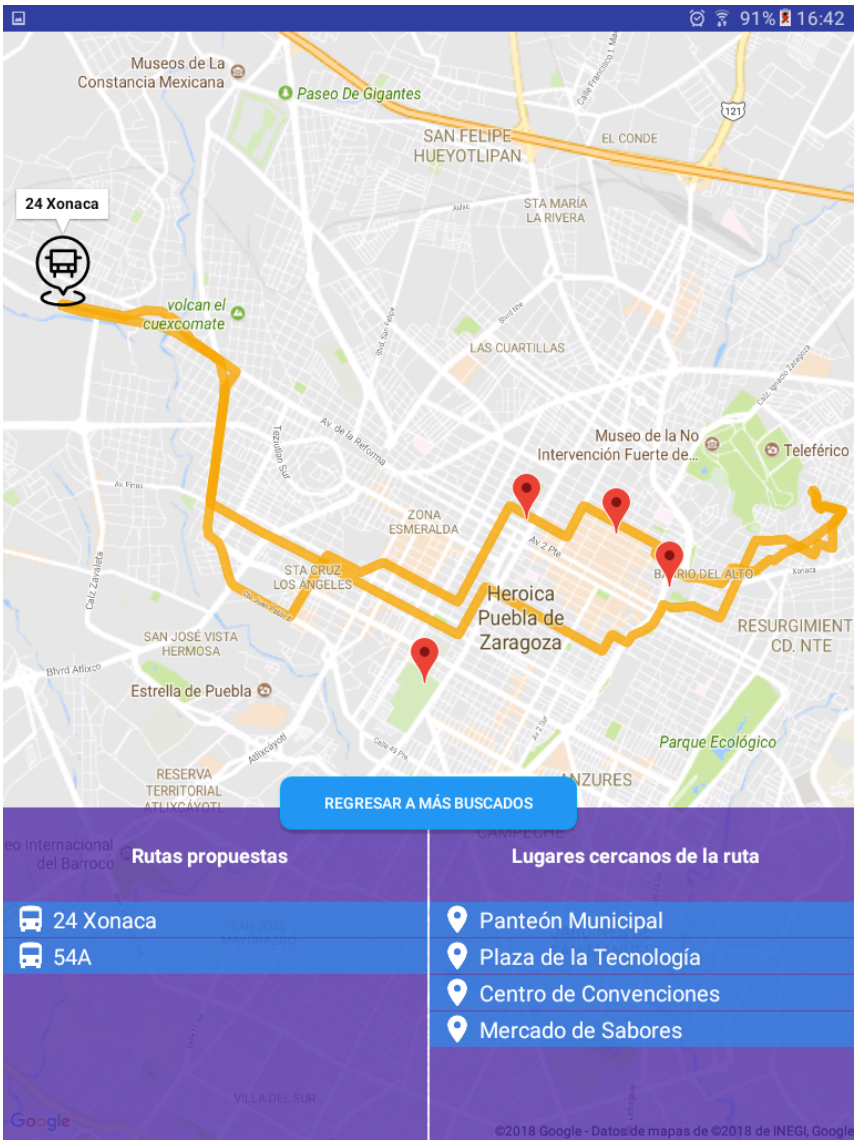



*Figura 5. 7 Propuesta de mejora en la ruta "24 Xonaca"*

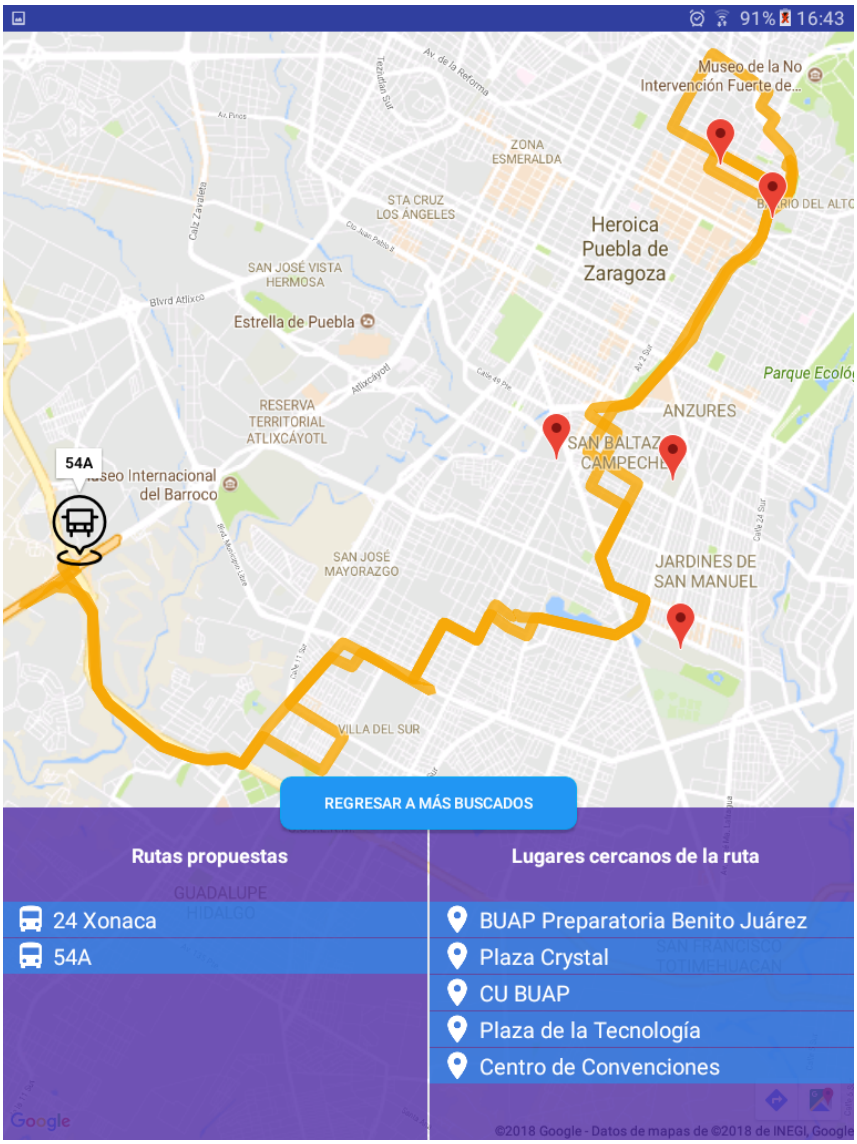



*Figura 5. 8 Propuesta de mejora en la ruta "54A"*

**Resultado obtenido**: Dadas las rutas y lugares más buscados por los usuarios, se cotejaron estos datos de manera que del lado izquierdo aparece el nombre de la ruta y del lado derecho aparecen los lugares cercanos de ella, que son más demandados por los usuarios. De igual manera se muestran sobre el mapa dichas propuestas. Tal como es posible observar en las figuras 5.7 y 5.8.

# CONCLUSIONES Y TRABAJO FUTURO

El uso de los Smartphones y el mercado de las aplicaciones móviles permiten un sin fin de posibilidades cuando se trata de facilitar tareas de la vida cotidiana. Sin embargo, un problema común en la ciudad de Puebla, es que el servicio de transporte público actual es deficiente debido en su gran parte a la mala planeación de las rutas de transporte público, y a la falta de información continua acerca de la demanda del servicio por parte de los usuarios.

A pesar de las aplicaciones actuales que permiten hacer búsquedas para ir de un lugar a otro haciendo uso del servicio de transporte público, ninguna de ellas ha sido diseñada para ofrecer una visualización en tiempo real de las unidades de cada ruta, ni mucho menos usan la información generada para proponer mejoras. Esto, aparte de permitir conocer a qué distancia se encuentra la unidad, también permite a los usuarios saber si la ruta ha sido desviada, ya que a veces cuando cierran calles debido a obras públicas u otras circunstancias, ni siquiera los mismos conductores saben que deberán desviarse.

La plataforma de 3 aplicaciones móviles y los algoritmos utilizados en este estudio, demostraron una alternativa de implementación tecnológica poco costosa y confiable, con la capacidad de facilitar a los usuarios la información en tiempo real acerca de cada ruta de transporte público en la ciudad de Puebla. Por otro lado, almacena la información de las rutas así como los lugares más buscados por los usuarios, sirviendo de muestra al momento de querer mejorar una ruta o crear otra. A continuación, se presentan las contribuciones que arrojó este estudio, atado a las limitaciones expuestas que dejan campo de estudio abierto para trabajo futuro que complemente el estudio.

## Contribuciones

Las principales contribuciones de este estudio se indican a continuación:

**Visualización en tiempo real de las unidades**: En este estudio se implementó la visualización de las unidades de cada ruta de transporte público, utilizando el GPS del celular del conductor. De esta manera las coordenadas de los conductores son almacenadas constantemente en la base de datos, y luego son consultadas por los usuarios, cuando desean saber por dónde va en ese momento la unidad de la ruta.

**Búsquedas de rutas para llegar de un lugar a otro**: Se diseñaron algoritmos para la obtención de las rutas de transporte público que mejor se amoldan a un origen y un destino. En este aspecto, el algoritmo tuvo que determinar primero si era posible llegar al destino haciendo uso de una sola ruta, en caso contrario comparaba todas las rutas cercanas al punto origen con todas las rutas cercanas al punto destino y verificaba si algunas de ellas se cruzaban en algún punto de sus trayectos a no más de 300 metros de distancia.

**Obtención de rutas y lugares más buscados**: En este estudio se demostró la capacidad de conocer las rutas más buscadas por los usuarios, así como los lugares que más ingresan. Y esto es muy importante, considerando que si buscan la ruta en la app es porque esa ruta la usan muy seguido.

**Despliegue de propuestas de mejora**: Se presentó mejoras en las rutas actuales, con el fin de aprovechar las que ya existen y en base a las búsquedas se pueda atender las necesidades de los usuarios, cubriendo la demanda de la población. Dichas mejoras sólo son sobre aquellas rutas que pasen por un gran número de puntos lugares que los usuarios más demandan.

## Limitaciones

Ya que la implementación real de este proyecto depende de la Secretaría de Transportes, así como de los concesionarios de las diferentes rutas de transporte público en la ciudad, se impidió la propuesta de mejoras reales sobre las rutas actuales, pues para esto se necesita que la población real use la app y así conocer la demanda de las rutas en base a las necesidades actuales de los usuarios.

Por otro lado, una limitante para haber hecho una prueba mayor de conductores, es que se requiere de un celular para cada unidad de transporte. Esto realmente no es una desventaja al momento de la implementación real de ser aprobada por el gobierno, pero de forma privada sería un poco costoso para los conductores e incluso para hacer pruebas .

Además, una limitación pequeña en cuanto a los usuarios de transporte público, es que la app sólo funciona para Smartphones con SO android. Es una limitante pequeña pues solo una pequeña parte de los usuarios posee un iPhone.

**Trabajo a futuro**

Algunas propuestas de trabajo futuro que contribuirían al trabajo realizado en esta tesis son:

**Nuevas rutas**: La creación de nuevas rutas, basadas en la demanda actual de los usuarios.

**Atención de solicitudes de abordaje en tiempo real**: Ofrecer a los usuarios la característica de solicitar su abordaje a la unidad más próxima. Con un conteo automático de usuarios por unidad, en base a los ascensos y descensos, se puede hacer saber a los usuarios mucho antes de abordar si la unidad más próxima tiene sobrecupo.

**Sistema de cobro**: Implementar un sistema de cobro que permita a los usuarios pagar desde su cuenta, así no hay problema al no llevar efectivo, incluso es más seguro para los conductores al no distraerse para cobrar.

**Paradas de ascenso inteligentes**: Los puntos de abordaje son establecidos por la plataforma en base a la demanda de usuarios y a los lugares estratégicos que no generen tránsito. Además, en base a las peticiones de los usuarios para abordar una unidad, el trayecto que efectúa el conductor se ve influenciado por estas. Permitiendo seguir su camino sin detenerse en caso de no haber usuarios esperando abordar.

## BIBLIOGRAFÍA


1. Brandes, U. & Erlebach, T. (2005). *Network Analysis: Methodological Foundations.* Germany: Springer.
2. Taha, H. (2012). Investigación de operaciones. México: Pearson.
3. Cuello, J. & Vittone, J. (2013). Diseñando apps para móviles. Argentina: Catalina Duque Giraldo.
4. Amaro, J. (2013). *El gran libro de programación avanzada con Android*. México: Alfomega.
5. DiMarzio, J. (2008). Android: A Programmer's Guide. U.S.: Mc Graw Hill.
6. Sommerville, I. (2011). Ingeniería de Software. México: Pearson.
7. Beck, K. (2012). Extreme Programming Explained. U.S.: Pearson Education.
8. Šmite, D. (2010). Agility Across Time and Space: *Implementing Agile Methods in Global Software Projects*. Germany: Springer.
9. Brunner, R; Emery, S. & Hall, R. (2009). Do You Matter?: How Great Design Will Make People Love Your Company. México: Pearson Education.
10. Silberschatz, A; Korth, H. & Sudarshan, S. (2002). Fundamentos de Bases de Datos. España: Mc Graw Hill.
11. Pérez de Celis, M. & Somodevilla, M. (2012). El Modelo Entidad Ralación. En M. Bonne. (Eds.). Objetos de Aprendizaje para la Enseñanza de Bases de Datos. (pp. 19-31). Puebla.
12. Ambrosio, A. & Ochoa, J. (2012). El Proceso de Normalización. En M. Bonne. (Eds.). Objetos de Aprendizaje para la Enseñanza de Bases de Datos. (pp. 34-55). Puebla.


### Webliografía


13. Análisis de redes de transporte https://es.wikipedia.org/wiki/An%C3%A1lisis_de_redes
14. Transporte público https://web.archive.org/web/20110429134312/http://www.publictransportation.org:80/about.asp
15. Redes de transporte

Litman, T. (2009). Public transportation's impact on rural and small towns: A vital mobility link. Victoria Transport Police Institute. www.vtpi.org

16. Google Maps Transit
    https://maps.google.com/landing/transit/cities/index.html
17. Moovit
    https://play.google.com/store/apps/details?id=com.tranzmate&hl=en
18. Rutadirecta
    https://play.google.com/store/apps/details?id=com.buzity.android&hl=en
19. Sanders, Peter. (2009). Fast route planning.
    Recuperado de: https://www.youtube.com/watch?v=-0ErpE8tQbw
20. Material Design
    https://material.io/guidelines/