SISTEMA DE CONSULTAS DE RUTAS PÚBLICAS BASADO EN SISTEMAS DE INFORMACIÓN GEOGRÁFICOS

CARLOS ENRIQUE GUISAO FRANCO

JOAN VALTIERE RUIZ ESPINOSA

Trabajo de grado presentado como requisito parcial para optar al título de Ingenieros de Sistemas

Asesor: Luis Alberto Giraldo Cuartas

Universidad EAFIT

DEPARTAMENTO DE INFORMÁTICA Y SISTEMAS

MEDELLÍN

2010

Nota de aceptación

_____

_____

_____

_____

_____

Firma del presidente del jurado

_____

Jurado

_____

Jurado

_____

Medellín, 17 de Julio de 2010

AGRADECIMIENTOS

Quisiera agradecer a mis padres, a mi hermano y a las personas que me quieren por todo el apoyo que recibí durante estos años de estudio. Sin ellos no hubiera sido posible llegar hasta este punto. Agradezco también a todos mis compañeros, profesores y amigos, ya que de ellos aprendí muchos temas relacionados con la ingeniería así como enseñanzas para la vida.

Joan V. Ruiz Espinosa

Quiero decirle a mi familia que la amo y que le agradezco inmensamente porque antes de brindarme los medios económicos para poder ser un profesional en ingeniería de sistemas me enseñaron a ser persona, gracias a esto siento que estoy en el lugar correcto en esta vida. Me siento muy agradecido porque en mi vida he conocido grandes personas que me han dejado muchas enseñanzas. En el proceso de este proyecto hicieron parte muchísimas personas, todas aportaron de una u otra manera y para éstas mis más sinceros agradecimientos por tener la disposición de colaborarme en las cosas que necesitaba. Este proyecto de grado me deja una enseñanza y es que no importa el tiempo que éste tome en realizarlo, lo más importante es hacer algo que refleje que el conocimiento que se tiene puede transcender más de lo que se enseña en la universidad, creo que por algo optamos el título de ingenieros.

Carlos Guisao Franco

CONTENIDO

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

GLOSARIO

Jolt Award: Premio que entrega Dr. Doob's anualmente desde el año 1990 a las industrias desarrolladoras de software que crean software rápido, fácil y eficiente[1].

API (Interfaz de programación de aplicaciones): Capa abstracta compuesta de un conjunto de funciones y procedimientos que puede ser utilizada por otro software.

Javascript: Lenguaje de programación basado en objetos, no tipeado.

ORM: mapeo objeto-relacional, es una técnica de programación para convertir datos entre el sistema de tipos utilizado en un lenguaje de programación orientado a objetos y el utilizado en una base de datos relacional.
En la práctica esto crea una base de datos orientada a objetos virtual, sobre la base de datos relacional. Esto posibilita el uso de las características propias de la orientación a objetos (básicamente herencia y polimorfismo) [6].

MVC: Modelo Vista Controlador, es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se ve frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página. El modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio, y el controlador es el responsable de recibir los eventos de entrada desde la vista [6].

Framework: En el desarrollo de software, es una estructura de soporte definida, mediante la cual otro proyecto de software puede ser organizado y desarrollado.

[1] http://www.drdobbs.com/joltawards/

Teóricamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio [6].

ActiveRecord: Es un enfoque al problema de acceder a los datos de una base de datos. Una fila en la tabla de la base de datos se envuelve en una clase, de manera que se asocian las filas únicas de la base de datos con objetos del lenguaje de programación usado. Cuando se crea uno de estos objetos, se añade una fila a la tabla de la base de datos. Cuando se modifican los atributos del objeto, se actualiza la fila de la base de datos. La clase envoltorio implementa métodos de acceso para cada columna de la tabla o vista [6].

CRUD (Create, Retrieve, Update, Delete): Son las operaciones Crear, Obtener, Actualizar y Borrar que se realizan sobre un motor de base de datos.

SQL (Structure Query Lenguaje): Lenguaje utilizado para acceder a una base de datos relacional.

MER (Modelo Entidad Relación): Es el diagrama que representa las entidades, atributos y relaciones de un modelo de datos.

Mysqlimport: Es una herramienta de mysql, que permite la carga rápida de grandes cantidades de datos en poco tiempo.

JSON (Javascript Object Notation): Es un formato ligero basado en el lenguaje Javascript, para el intercambio de datos entre lenguajes.

RESUMEN

El siguiente proyecto de grado describe el proceso de análisis, diseño e implementación de una aplicación donde un usuario entrega dos puntos geográficos (Origen-Destino) al sistema y éste le dirá cuál es la ruta más óptima a pie para desplazarse hacia a dicho lugar e indicará si existen rutas de buses cercanas a la ruta que debe ser transitada por el usuario.

Como parte del proceso se hace énfasis en la construcción del modelo de datos, el algoritmo a implementar y las tecnologías utilizadas que permitieron la construcción de esta aplicación.

INTRODUCCIÓN

Los sistemas de información geográficos (SIG) son herramientas que capturan, interpretan, procesan y representan datos geográficos. En base a estos se puede crear aplicaciones que suplan la necesidad de ubicar a una persona geográficamente trazar rutas que la orienten hacia un lugar específico. Para aquellas personas que poseen un medio de transporte particular, utilizar este tipo de sistemas resulta ser muy atractivo ya que reduce el tiempo y la distancia utilizada para llegar a un lugar de destino. Pero, ¿Qué pasa entonces cuando una persona puede o quiere desplazarse a pie o por medio del sistema de transporte público de su ciudad y no dispone de un SIG que le indique qué ruta debería tomar?

Es esta problemática la que inspira a desarrollar el siguiente proyecto de grado, cuya solución requiere crear una aplicación que irá progresando por las etapas de análisis, diseño e implementación. Esta aplicación implica crear un modelo de datos que relacione la malla vial del municipio de Medellín con su sistema metro y siete rutas de buses ficticias para luego implementar un algoritmo que dé solución a este problema.

La aplicación consta de una interfaz web, fácil de manejar, la cual requiere seleccionar dos puntos geográficos en la ciudad de Medellín, uno de origen y otro de destino para luego retornar una ruta y que esta sea explicada al usuario final.

DEFINICIÓN DEL PROBLEMA

El tiempo en el que la imprenta era una industria vital para la difusión de la información ha quedado atrás, la civilización ha pasado del envío de cartas e impresión de libros y fotografías, a la digitalización de estos convirtiéndolos en recursos públicos que pueden ser accedidos en cuestión de minutos, es decir se ha entrado en la era digital. Es la entrada a esta era lo que ha permitido ampliar el uso de la tecnología en áreas antes impensadas, un ejemplo de esto es la localización geográfica que ha sufrido enormes pero positivas transformaciones. Suponga que un viajero está en una ciudad desconocida y desea orientarse para saber cómo llegar a determinado lugar. Para esto el viajero tiene varias opciones: Leer las señales de tránsito, preguntar a una persona cercana o mirar en un mapa impreso para buscar la forma de llegar, en este último caso el viajero marcaría un punto inicial, que es aquel donde se encuentra, luego señalaría el lugar al que desea ir, procedería a analizar el mapa con sus posibles rutas y por último trazaría una ruta por la cual se desplazaría, corriendo el riesgo de tomar una vía incorrecta o una que tome mucho tiempo en ser recorrida. Viendo que todas las opciones planteadas anteriormente pueden resultar insatisfactorias de alguna manera, algunas en mayor medida que otras, sea porque la persona a la que se le pregunta no tiene conocimiento sobre cómo llegar al lugar deseado o no sabe explicarlo o porque el mapa omite ciertas calles o se hace difícil ubicarse en el, entre otros y teniendo en cuenta que hoy en día se puede aprovechar el desarrollo de la tecnología para hacer estas consultas, se quiere brindar una solución que use tecnologías que faciliten la labor de ubicar y trazar una ruta óptima y confiable en un mapa para el satisfactorio desplazamiento de la persona que use el sistema.

Actualmente la aplicación más famosa sobre la Web en el manejo de mapas es Google Maps, esta ofrece a países desarrollados como Estados Unidos, Francia y Reino Unido una serie de servicios muy útiles como consulta de tráfico en las calles, vista de vías, consulta de establecimientos comerciales y búsqueda de direcciones, sin embargo aunque en los demás países también ofrece servicios similares, no ofrece todos los mencionados anteriormente. En Colombia, más específicamente en la ciudad de Medellín, no hay acceso completo al portafolio de Google Maps mencionado anteriormente. Aunque sí se puede observar, a través de esta aplicación la malla vial de dicha ciudad (callejero) aún se encuentra falencias y no hay información confiable sobre los números de calles y carreras. Esto es negativo para quienes consultan información allí e igualmente evita que la solución a la que se desea llegar en este proyecto de grado pueda referenciarse en su totalidad en esta aplicación. La aplicación a desarrollar requiere fundamentalmente de datos geográficos de buena calidad que no están disponibles gratuitamente, sin embargo, para fines académicos estos pueden ser obtenidos en instituciones educativas. Una vez obtenidos no se puede tener la certeza de que estos sean de excelente calidad, por lo tanto no se puede asegurar la exactitud y/o veracidad de la información entregada por la aplicación para que esta sea interpretada por el usuario final.

En resumen y más específicamente este proyecto de grado busca aprovechar la tecnología para implementar un servicio de consultas en línea para la ciudad de Medellín, en el que se pueda resolver fácilmente la problemática de cómo desplazarse a pie de un lugar a otro en la ciudad y qué rutas de buses cercanas transitan por el camino a ser recorrido a pie.

1. OBJETIVOS DEL PROYECTO

1.1 OBJETIVO GENERAL

- Desarrollar una extensión de una herramienta ya existente de mapas geográficos en línea, para consultar una ruta dado un lugar de origen y un lugar destino en la ciudad de Medellín.

1.2 OBJETIVOS ESPECÍFICOS

- Integrar la herramienta con una aplicación de mapas online como Google Maps.
- Integrar algunas rutas de tránsito de la ciudad de Medellín con la aplicación que se desarrollará, brindando con esto una fuente confiable para el conocimiento de éstas.
- Implementar algoritmos que desplieguen un resultado en un tiempo prudente para el usuario.
- Entregar respuestas al usuario respecto a la ruta que debe tomar, en una forma adecuada, con el fin de que éste la encuentre clara y comprensible.
- Exponer como el framework Ruby on Rails y el API de Google Maps se complementan para agilizar el desarrollo de la aplicación.

2. ALCANCE

Se entregará una aplicación Web que integre el API de Google Maps, la malla vial de la ciudad de Medellín y el framework para aplicaciones Web Ruby on Rails. Mediante un explorador web el usuario podrá acceder a la aplicación y deberá seleccionar en el mapa de la ciudad un lugar de origen y un lugar de destino, con el fin de obtener una respuesta que contendrá información detallada de cómo debe desplazarse a pie por las vías de la ciudad hasta llegar al lugar deseado; además indicará si por dicho recorrido pasan rutas de transporte público cercanas y cuáles son estas.

Se hacen las siguientes aclaraciones:

- Los datos de la aplicación que corresponden a la malla vial de la ciudad de Medellín no se actualizan con información del día a día. Estos corresponden a un periodo de tiempo entre el año 2008 y 2009.
- No se posee una buena calidad de datos de la malla vial, por lo tanto, esto afecta la explicación de la ruta a tomar.
- Las coordenadas geográficas de esta malla vial de la aplicación, no corresponde a las poseídas por Google Maps, por lo tanto a la hora de pintar una ruta, ésta no se ubicará en todos los casos por encima del callejero que provee Google Maps.
- Las consultas hechas a la aplicación sólo devolverán un resultado si estas son sobre la ciudad de Medellín y no sobre las demás ciudades del área metropolitana.
- En lo que concierne a las rutas de transporte, se cuenta con siete rutas de buses ficticias de transporte público, que no corresponden con las rutas reales

de transporte de la ciudad de Medellín. Esto se debe a que los intentos por obtenerla no fueron satisfactorios.

3. JUSTIFICACIÓN

Cuando una persona se encuentra en un lugar y desea ir a un determinado sector, ¿Qué dudas pueden surgir? Posibles preguntas serían: ¿Cómo me voy a ir? (a pie, en transporte público, en transporte particular), ¿Qué vías o rutas de buses puedo tomar?, etc. La experiencia y la costumbre serán las determinantes a la hora de responder estas preguntas planteadas, dependiendo del medio de transporte elegido muy seguramente este recorrerá una trayectoria habitual que lleva a ese lugar deseado. Ahora, ¿Será qué estas rutas utilizadas son siempre las óptimas, existirán otras alternativas?, ¿Acaso conoce cada ciudadano la totalidad de la ciudad donde vive o se encuentra? Probablemente no. Basándose en la suposición que gran parte de los ciudadanos usan transporte público y/o caminan y siendo el medio de transporte para el cual se desea obtener información al respecto, seguramente a la hora de preguntar a otro ciudadano, éste no siempre va a poseer información clara, precisa y concisa de la rutas de transporte público o vías a sugerir, y en caso de creer que éste la tiene no se tiene una garantía del 100% que ésta conducirá al lugar que se desea llegar. Analizando la problemática planteada esta se podría atacar poniendo a la disposición de la persona un sistema de información geográfico que le indique que debe hacer para llegar a dicho lugar. Actualmente hay herramientas en línea como Google Maps que pueden sugerir rutas de transporte, mostrar la ruta más óptima, el estado del tráfico, la panorámica de la vía, entre otros. Empresas como Google, Yahoo y Microsoft no prestan este servicio para las ciudades del territorio colombiano, es por esto que viendo esta brecha se decide desarrollar una herramienta que dé solución al problema planteado anteriormente.

4. MARCO TEÓRICO

Las primeras investigaciones para el desarrollo de los sistemas de información geográficos (SIG) se dieron aproximadamente en los años de 1960 cuando entidades como el Census Bureau creó GBF/DIME (Geographic Base File Dual Independent Map Encoding) y el laboratorio de computación grafica y análisis espacial de la universidad de Harvard creó Symap, siendo estos programas de los primeros sistemas y archivos de mapas digitales.

Dado el avance de la tecnología y el ancho de banda al que se puede acceder se ha permitido que los SIG se puedan acceder como un servicio Web, logrando con esto que las empresas estén empezando a usar los servicios que los SIG proveen para ofrecer a sus clientes nuevos productos y servicios [8].

Esta tecnología está siendo utilizada en países en diversos países con diferentes motivos, como tener información geográfica, estadísticas de determinados temas en diferentes zonas de una ciudad, información de servicios de telefonía, entre otros. Algunos ejemplos de uno de los muchos usos que le han dado ciertos países a esta tecnología son: España lo ha utilizado para los planes de organización urbana, tener información de parcelas agrícolas y oleicas de las provincias. Estados unidos lo utilizan para servicios públicos y refuerzo de fuerza policial. En Bregenze, Austria se utiliza para conocer las necesidades de sus departamentos, el departamento de silvicultura en Irlanda utiliza SIG como una herramienta para tener control de los bosques, entre otros [9].

En Colombia son pocos los SIG que se han implementado. Algunas aseguradoras usan estos sistemas para analizar el terreno de una vivienda y calcular con esto la cuota que

se deberá pagar. El instituto de recursos biológicos Alexander Von Humboldt para sus investigaciones sobre biodiversidad en diferentes zonas del país [10].

Sin embargo, en Colombia a diferencia de muchos países, los sistemas de información geográficos son de uso privativo, por lo que estos no tienen tanta penetración como lo hay en países como Estados Unidos y algunos países europeos, donde la cantidad de servicios prestados por empresas como Google, Yahoo, Microsoft, entre otras, son mucho mayores que en países latinoamericanos.

Tecnologías similares

Google Maps

Este es un servicio prestado por la compañía Google. Google Maps tiene funciones que permiten ver el mapa de diferentes formas, como mapa de relieve, mapa vial, mapa con imágenes satelitales, herramientas para navegar por el mapa con facilidad, he incluso, puede calcular el camino más corto para unir dos puntos dependiendo de la ciudad en que se use [5].

Yahoo Maps

Este servicio prestado por Yahoo, además de permitir al usuario navegar por mapas hechos con imágenes satelitales y geográficas, ofrece otros servicios como consultar el estado de tráfico y consulta del estado del tiempo. Al igual que Google Maps, algunas de sus tareas puede están no estar disponibles dependiendo de la ciudad en la que se utiliza [5].

Bing Maps

Microsoft es el creador de Bing Maps. Una página Web para el uso de mapas donde se puede consultar cuál es el camino más corto entre dos puntos, ver lugares de interés, que los usuarios compartan mapas con conocidos, estado del trafico, entre otras funciones. Este servicio ofrece un modo de vista llamado “Ojo de águila”, el cual sirve

para enfocar una ciudad desde una vista área pero con cierto ángulo, simulando con esto el punto de vista que tendría un ave sobrevolando la ciudad. [5].

OpenStreetMap

Este servicio, a diferencia de los que ya se han mencionado, es un proyecto open source que permite al usuario crear y editar los mapas según su necesidad. Como ventaja tiene que cualquier usuario puede actualizar información que contenga la aplicación, pero a su vez, esta ha sido su desventaja, ya que tiene la cantidad de información que se esperaría al pertenecer a una comunidad con tantos integrantes alrededor del mundo. Esto se ve claramente cuando se compara mapas de ciudades en Estados Unidos o de ciudades en Europa con mapas de ciudades de países donde la tecnología GPS no es frecuentada por los ciudadanos comunes [2][5].

Proyectos similares:

Como ya se mencionó Google Maps, Yahoo Maps y Bing Maps entre muchos servicios que ofrecen, calcular cuál es el camino más corto entre dos puntos dados e informar rutas de transporte que lleven a determinados lugares son los servicios que se quieren implementar en este proyecto de grado. En Colombia se han desarrollado páginas Web que prestan parcialmente estos servicios.

Transporte en línea:

El usuario podrá consultar en esta página que rutas de buses pasan cerca al un punto que se desee y dibujará en el mapa el trayecto de la ruta. Además de esto, la página Web está conectada con otras páginas Web donde se puede consultar si se tiene alguna multa de tránsito, reportes que hace la secretaría de tránsito de Medellín respecto al estado de las calles, entre otras opciones [13]

[2] http://wiki.openstreetmap.org/wiki/FAQ

Ciudad en línea:

Esta página Web permite al usuario consultar sitios de interés como hoteles, embajadas, hospitales, zonas de comercio, sitios turísticos entre otros. Además de esto, ofrece a los usuarios la opción de marcar algún sitio o de convocar a reuniones a gente conocida, enviando un correo electrónico que lleva al mapa, mostrando el lugar de encuentro y una descripción de la convocatoria [11].

Seguridad en línea:

En esta página Web, el usuario podrá marcar sitios donde haya sido víctima de un incidente o lo vea con el fin de alertar a otros usuarios para que estos no sean víctimas de la inseguridad de la zona. Se puede marcar situaciones como homicidios, asaltos, disturbios, entre otros [12].

5. SOLUCIÓN

Una vez se tiene claro cuál es el problema a solucionar, lo siguiente es analizar las tecnologías disponibles referentes al manejo de mapas digitales y los lenguajes de programación enfocados al desarrollo web que permiten la compatibilidad con dicha tecnología. En cuestión de tecnologías hay un abanico de posibilidades, como Google Maps, Nokia Maps, Yahoo Maps, OpenStreetMap y en cuestión de lenguajes de programación Ruby on Rails, Java, .Net, Php. Se elige Google Maps y Ruby on Rails como las herramientas que facilitaron el desarrollo de la aplicación, a continuación se da una explicación del por qué de cada una.

5.1 ¿POR QUÉ GOOGLE MAPS?

Se optó por Google Maps porque su **API**[3] tiene una cantidad de funcionalidades que facilitan el desarrollo de la aplicación, además cuenta con muy buena documentación facilitando su comprensión basado en ejemplos, una comunidad virtual en caso de necesitar ayuda, blogs que recogen las experiencia de desarrolladores de esta tecnología y por último el aspecto más importante a resaltar es la posibilidad de ver las imágenes satelitales del área metropolitana de Medellín y su callejero, ventajas que no poseen las otras tecnologías ya mencionadas y que resulta vital para la solución del problema planteado. Cabe mencionar que Google Maps no limita al programador a utilizar un lenguaje en específico que implemente su API, existen actualmente varias ofertas de API: El API de **Javascript**[4], el API para Flash y el API para Static Maps. Además se ofrece el API de mapplets, pequeñas aplicaciones que se pueden ejecutar

[3] Léase API en el glosario
[4] Léase Javascript en el glosario

dentro de Google Maps. Dependiendo de la necesidad del desarrollador, es posible utilizar una o la combinación de varias de estas API mencionadas.

Dado el caso en que se tenga un proyecto web comercial que genere muchas visitas diarias, se cuenta con un amplio tráfico permitido por día, el cual es de 500.000 visitas, además de tener compatibilidad con los exploradores web más usados como: Firefox, Google Chrome, Internet Explorer y Safari.

Viendo todas estas ventajas que ofrece Google Maps y eligiendo esta tecnología para que sea parte del desarrollo del proyecto, la primera decisión que se debe tomar es el API que se va a utilizar, para este proyecto se elige el API de Javascript por ser un lenguaje altamente usado en la web, de fácil uso y de preferencia por los desarrolladores del proyecto.

El siguiente paso fue buscar un lenguaje de programación para desarrollo Web que permitiera la comunicación con esta tecnología, se encontró que **Ruby On Rails**[5], **Php**[6] y **Java**[7] eran buenos candidatos, después de hacer un análisis se opta por Ruby On Rails.

5.2 ¿POR QUÉ RUBY ON RAILS?

Ruby on Rails, *Rails* o *RoR* es un framework de programación que hace más fácil desarrollar, implementar y mantener aplicaciones web. Desde que se lanzó su primera versión en el año 2006, pasó de ser un juguete a un fenómeno mundial. Ha ganado el **Jolt Award**[8] como la mejor herramienta de desarrollo web y más importante aún, se ha convertido en el framework de preferencia para un gran rango de las llamadas aplicaciones 2.0. [1]

[5] http://rubyonrails.org/
[6] http://php.net/index.php
[7] http://java.sun.com/
[8] Léase Jolt Award en el glosario

Las aplicaciones Ruby On Rails son desarrolladas bajo la arquitectura Modelo Vista Controlador **MVC**[9] **Figura 5.1**, la cual separa los datos, la interfaz de usuario y la lógica de control de la aplicación, haciendo estos componentes independientes y actuando según el estándar [1].

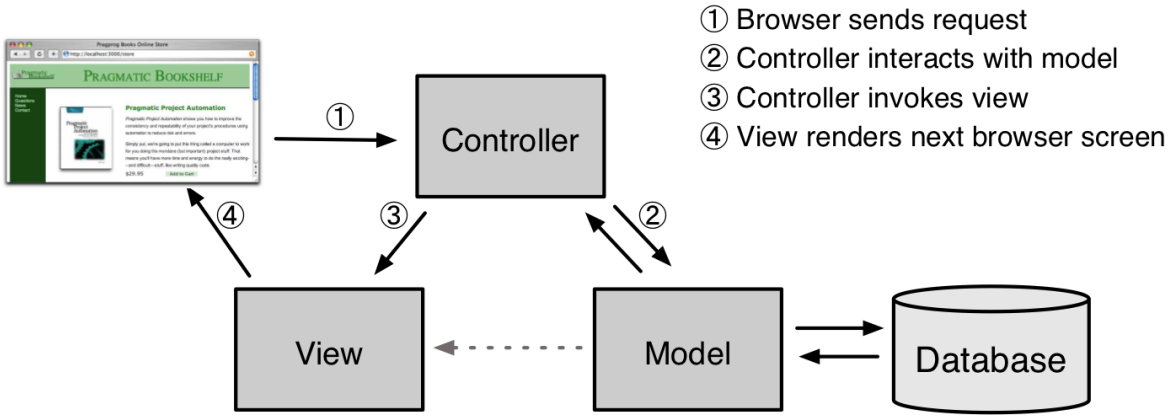


Figura 5.2 1: Modelo Vista Controlador[1]

Otro factor interesante es que es un lenguaje que permite crear código fácil de escribir y de leer, ayudando al programador en su labor de hacer correcciones o mejoras a la aplicación. El manejo[10] de los modelos de datos o tablas de bases de datos es muy simple y tiene compatibilidad con varios motores de bases de datos, como: DB2, Firebird, Frontbase, MySQL, Openbase, Oracle, Postgres, SQLite, SQL Server, y bases de datos Sybase[1] .

Por último se necesita un framework[11] para hacer tests a la aplicación, Ruby on Rails pone a disposición varios framework como **Test-Unit**[12], **Cucumber**[13], **RSpec**[14] entre

[9] Léase MVC en el glosario
[10] Por manejo se hace referencia a todas las operaciones que pueden existir en el manejo de una base de datos, como: Creación, modificación, eliminación entre otros.
[11] Léase Framework en el glosario
[12] http://wiki.rubyonrails.org/testing/test-unit

otros, se opta por Test-Unit porque suple las necesidades básicas que se necesitan en cuanto a testing.

[13] http://cukes.info/
[14] http://rspec.info/

6. DESARROLLO DE LA APLICACIÓN

6.1 DESARROLLO WEB

A continuación se detallan los aspectos más importantes que hicieron parte del desarrollo de la aplicación, se hará un enfoque especial en describir el proceso para la construcción del modelo de base de datos, la integración de Ruby on Rails con el API de Google Maps y los aspectos más relevantes utilizados para crear la interfaz gráfica.

6.1.1 Creación de una aplicación en Rails

Crear una aplicación Web en Ruby on Rails es muy sencillo, simplemente se ejecuta el comando *rails nombreDeAplicacion*[15] y al instante se creará la siguiente lista de directorios:

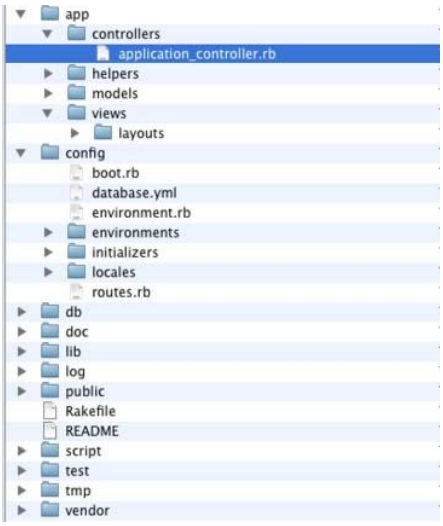


Figura 6.1.1 1 Estructura del directorio de Rails[1]

[15] La palabra varia en base a el nombre del proyecto que se quiera dar.

Básicamente los directorios que más se utilizan en este proyecto son **app/** en el cual están todos los componentes de la aplicación, es decir la representación del modelo vista controlador MVC, **db/** que es donde van ir todos scripts para la creación de los modelos de datos y **public/** que es donde va todo el código Javascript que permite la comunicación entre el API de Google Maps y Ruby on Rails; y no menos importante de mencionar **/config** que es donde va la configuración de la conexión a la base de datos, el entorno de Rails, entre otros y **/script** que es donde están los scripts encargados de subir el servicio web, crear modelos, controladores, etc.; para un ambiente de desarrollo estos dos últimos directorios no necesitan una configuración en especial.

6.1.2 Modelo de datos

Lo primero que se debe tener en cuenta para ver si es viable desarrollar esta aplicación es analizar y diseñar el modelo de base de datos para su posterior implementación. Con Ruby on Rails se tiene la ventaja de que a la hora de configurar la capa que va acceder a los datos que integran la aplicación se requiere de un esfuerzo mínimo. La capa que permite esto se llama **ActiveRecord**[16], basada en el estándar ORM[17], las tablas se traducen en clases, filas a objetos y columnas a atributos de los objetos [1]. Otra ventaja es que al ser ActiveRecord una capa, permite tener una independencia del motor de base de datos a utilizar, para este caso se optó por Mysql.

Con Rails integrar la base de datos a utilizar es simple, basta sólo con cambiar dos parámetros: *adapter* y *socket*; *adapter* recibe el motor de base de datos a utilizar y *socket* la dirección correspondiente en el sistema operativo. Esta configuración puede ser la misma para los tres ambientes que maneja Rails: desarrollo, test y producción.

[16] Léase ActiveRecord en el glosario
[17] Léase ORM en el glosario

Para un ambiente de desarrollo, la configuración es muy sencilla, como se muestra a continuación:

```
development:
  adapter: mysql
  encoding: utf8
  reconnect: false
  database: maps_development
  pool: 5
  username: root
  password:
  host: localhost
  socket: /var/run/mysqld/mysqld.sock
```

Y hacer operaciones CRUD[18] sobre los modelos es muy simple también:

```
class Bus < ActiveRecord::Base
end
bus = Bus.find(1)
bus.company = "Transportes S.A."
bus.save
```

En este ejemplo se ve claramente la funcionalidad de ActiveRecord, ninguna instrucción SQL[19] tuvo que ser ejecutada.

El diseño del modelo entidad relación (MER[20]) que representa el modelo de base de datos de la aplicación se muestra en la figura 6.1.2 1.

[18] Léase CRUD en el glosario
[19] Léase SQL en el glosario
[20] Léase MER en el glosario

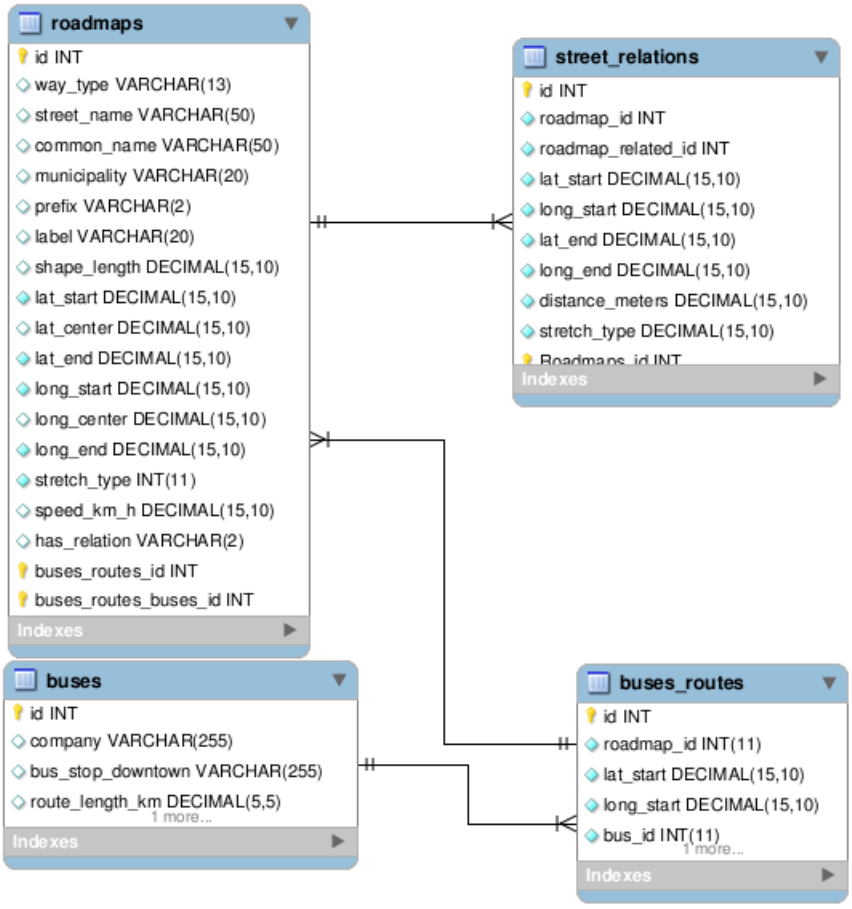


Figura 6.1.2 1: Modelo Entidad Relación

El MER indica las tablas que se necesitan, los campos que las componen y los tipos de datos correspondientes, a continuación se detalla cada modelo (Tabla), explicando que datos debe contener cada uno y al final una simple muestra de cómo se pueden crear a través de código Rails, que luego será traducido a código SQL correspondiente.

6.1.2.1 Modelo roadmaps

Contiene todos los datos geográficos de la ciudad de Medellín.

I. Campos del modelo:

Way_type: Tipo de vía, es decir, si es carrera, calle, diagonal, avenida, transversal, metro, metrocable, trayecto o puente.
Street_name: Es el resto de la nomenclatura de la dirección, por ejemplo, para la dirección Cl 56 - 85, way_type=CL y Street_name=56-85.
Common_name: Nombre común que se le da a las vías de una ciudad, por ejemplo, la ochenta, San Juan.
Municipality: Municipio al cual pertenece el trayecto.
Prefix: Indica si la vía es del sur (Las vías del sur por lo general son del municipio de Envigado) o si es un trayecto del metro indica a qué línea pertenece.
Shape length: Es la longitud en grados decimales del tramo.
Lat_start: Latitud inicial del tramo.
Lat_center: Latitud central del tramo.
Lat_end: Latitud final del tramo.
Long_start: Longitud inicial del tramo.
Long_center: Longitud central del tramo.
Long_end: Longitud final del tramo.
Stretch_type: Tipo de tramo, 1 para vías de la ciudad, 2 para trayectos del metro, 3 para trayectos donde inicia una estación del metro y 4 para trayectos que conectan el metro con vías de la ciudad.
Speed_km_h: Indica la velocidad promedio en kilómetros por hora del trayecto.
Has_relation: Indica si el registro tiene relación con otros registros de este modelo.

II. Verificar la integridad de los datos y su unicidad:

- Los campos latitud inicial-final, longitud inicial-final no deben ser nulos, iguales a menos uno, o ser equivalentes en valores ya que se busca que haya una distancia mayor que cero en el tramo.
- La latitud inicial-central-final, longitud inicial-central-final y Shape Length deben manejar cifras decimales por punto y no por coma.
- Con respecto a la unicidad no es necesario tener registros que compartan la misma latitud inicial-final y longitud inicial-final, así el resto de los campos sean diferentes.
- Para este modelo de datos no se tiene información sobre la orientación de la vía y por lo tanto son datos innecesarios a la hora de formar el grafo como se explicará más adelante.

III. Importar el archivo de datos hacia el modelo:

Una herramienta muy útil a la hora de hacer importación de archivos hacia modelos es mysqlimport[21] esta sencilla instrucción lleva a cabo la importación de aproximadamente 43000 registros en pocos segundos:

```
sudo mysqlimport --delete --fields-terminated-by="," --
columns=way_type,street_name,common_name,municipality,prefix,label,shape_length,lat_start,l
at_center,lat_end,long_start,long_center,long_end,stretch_type,speed_km_h,has_relation --
user=root --local maproad_development /tmp/roadmaps.csv;
```

Como parámetros recibe el separador de columna, las columnas del archivo, el usuario y el nombre de la base de datos y la ruta del archivo a importar.

[21] Léase mysqlimport en el glosario

6.1.2.2 Modelo street_relations

Contiene todas las relaciones que hay entre los registros de *roadmaps*.

I. Definir los campos del modelo:

Roadmap_id: Es el nodo inicial que tendrá relación con un nodo final.
Roadmap_related_id: Es el nodo final que se relaciona con un nodo inicial.
Lat_start: Es la latitud inicial del tramo.
Lat_end: Es la longitud final del tramo.
Long_start: Es la longitud inicial del tramo.
Long_end: Es la longitud final del tramo.
Stretch_type: Es el tipo de tramo del nodo inicial.
Distance_meters: Es la distancia en metros, obtenida por la fórmula del haversine[2].

$$\operatorname{haversin}\left(\frac{d}{R}\right) = \operatorname{haversin}(\varphi_1 - \varphi_2) + \cos(\varphi_1)\cos(\varphi_2)\operatorname{haversin}(\Delta\lambda).$$

Donde:
d = Distancia entre los dos puntos latitud-longitud inicial y latitud-longitud final.
R = Es el radio de la esfera, en este caso, el de la tierra que equivale a 6371000 metros.
$\boldsymbol{\varphi}_{1=}$ Es la latitud en radianes del punto inicial.
$\boldsymbol{\varphi}_{2}$ **=** Es la latitud en radianes del punto final.
$\boldsymbol{\Delta\lambda}$ **=** Es el diferencial entre la longitud final y la longitud inicial.
haversin(θ) = sen^2 (θ/2).

II. Armar todas las relaciones existentes:

Cada registro en el modelo Roadmaps es un tramo del área metropolitana compuesto por coordenadas geográficas, el siguiente paso es analizar cómo se conectan todos los *tramos* o *nodos* por medio de sus coordenadas angulares (latitud, longitud).

La lógica utilizada para armar todas las conexiones es hacer la suposición que existe un enlace entre dos tramos (*A*,B) siempre y cuando la latitud-longitud final del tramo *A* es igual a la latitud-longitud inicial del tramo *B* y viceversa, será llamada condición *inicial-final* para facilitar la comprensión de los métodos utilizados de aquí en adelante. A continuación se muestra la figura 6.1.2.2 1 y la tabla 6.1.2.2 1 que representan cómo es una conexión correcta y cómo deben quedar los registros en el modelo street_relations respectivamente.

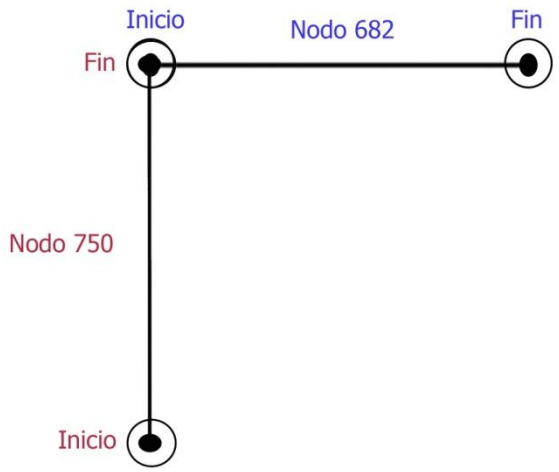


Figura: 6.1.2.2 1 Nodos con conexión correcta

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
1	*682*	*750*	*6.1624*	*-75.605571*	*6.1632*	*-75.605904*	*102.3653*	*1*

Tabla 6.1.2.2 1: Representación en datos de la figura 6.1.2.2 1

Pero, no se puede garantizar que todas las conexiones se puedan armar bajo esta premisa, ya que hay registros que se conectan solamente por latitud-longitud inicial véase figura 6.1.2.2 2 o por latitud-longitud véase final 6.1.2.2 3. En términos

generales armar este tipo de conexiones es una de las tareas más dispendiosas del proyecto de grado. Todo lo descrito en los pasos *a* y *b* se hizo por medio de instrucciones SQL.

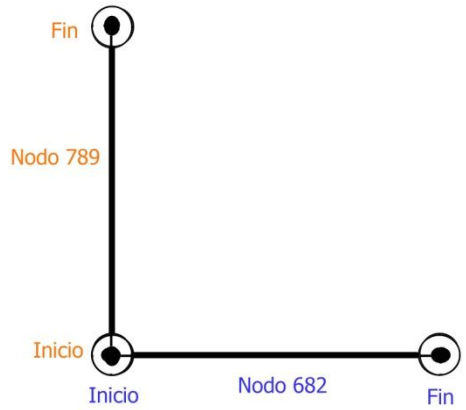


Figura: 6.1.2.2 2: Nodos conectados por latitud longitud inicial

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
5	789	682	6.176078061	-75.6448333	6.17607806	-75.64483328	0	1
6	682	789	6.176078061	-75.6448333	6.17607806	-75.64483328	0	1

Tabla 6.1.2.2 2: Representación en datos de la figura 6.1.2.2 2

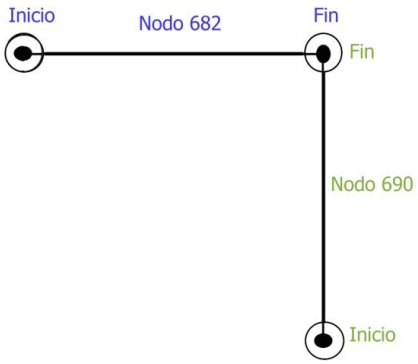


Figura: 6.1.2.2 3: Nodos que se conectan por latitud longitud final

a. Crear nuevos registros para los tramos que se conectan por latitud-longitud final

Este paso es simple, como se muestra en la figura 6.1.2.2 3 hay una conexión entre estos tramos por medio de latitud-longitud final, la idea es insertar en el modelo roadmaps un nuevo registro (40034) que tendrá las coordenadas geográficas donde se conecten estos dos tramos, para identificar estos registros el campo *label* tendrá el valor de FINAL.

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
2	2037	40034	6.17612506	-75.6443373	6.17635006	-75.64436127	25.1593	1
4	2051	40034	6.176276061	-75.6450583	6.17635006	-75.64436127	77.49145	1

Tabla 6.1.2.2 3: Creación de un nuevo para conectar registros que se conectan por latitud longitud final

Luego crear la tabla temporal *nodes_by_lat_long_end* que contendrá las relaciones realizadas bajo la condición "inicial-final" entre los tramos que se conectan por latitud-longitud final y los tramos que se crearon, descritos en el paso anterior.

id	*way_type*	*street_name*	*on_name*	*municipality*	*prefix*	*label*	*shape_length*	*lat_start*	*lat_center*	*lat_end*	*long_start*	*long_center*	*long_end*
2037	*KR*	*1-Mar*		*MEDELLIN*	*E*	*KR 3 E*	*0.00022628*	*6.17612506*	*6.17623756*	*6.17635006*	*-75.644337*	*-75.644349*	*-75.644361*
2051	*CL*	*1A-2C*		*MEDELLIN*	*E*	*CL 1A*	*0.00070093*	*6.17627606*	*6.17631181*	*6.17635006*	*-75.645058*	*-75.64471*	*-75.644361*
40034	*CL*	*1A-2C*		*MEDELLIN*	*E*	*FINAL*	*0.00070093*	*6.17635006*	*6.17631181*	*6.17635006*	*-75.644361*	*-75.64471*	*-75.644361*

Tabla 6.1.2.2 4: Relaciones creadas en la tabla street_relations para nodos que se conectan por latitud longitud final

b. Arreglar conexiones para todos los tramos que se conecten por latitud-longitud inicial

Para llevar a cabo esta tarea se debe crear una serie de tablas temporales que se describen a continuación:

- Crear la tabla temporal *street_relations_temp* que contendrá las siguientes relaciones: Los tramos que se conectan por la condición *inicial-final*, los tramos que se conectan

sólo por latitud-longitud inicial y todas las relaciones de la tabla *nodes_by_lat_long_end*.

- Crear la tabla *st_good_ones* que contendrá los tramos que se conectan por la condición *inicial-final* y los registros de la tabla *nodes_by_lat_long_end.*

- Crear la tabla *st_fix_records_zero*, esta tabla requiere de dos pasos, primero es insertar todos los registros que tengan el campo distance_meters en cero y sus *roadmap_related_id* correspondientes:

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
20	19499	19445	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1
21	19499	19446	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1
22	19499	19500	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
23	19499	19501	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
24	19499	45228	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1
25	19500	19499	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
26	19500	19501	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
27	19500	45251	6.261870815	-75.6189179	6.26192731	-75.61930483	43.22495	1
28	19501	19499	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
29	19501	19500	6.261870815	-75.6189179	6.26187082	-75.61891792	0	1
30	19501	45246	6.261870815	-75.6189179	6.26182527	-75.61820717	78.72247	1

Tabla 6.1.2.2 5: Ejemplo de la tabla st_fix_records_zero[22]

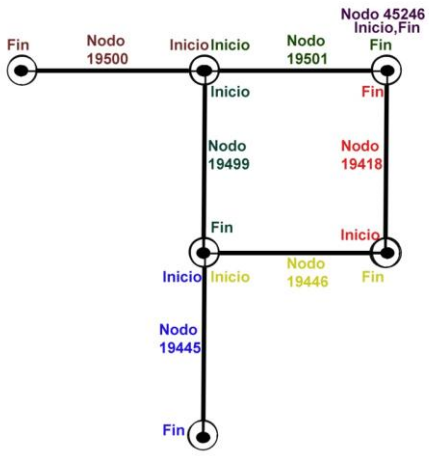


Figura: 6.1.2.2 4: Representación gráfica de la tabla 6.1.2.2 5

[22] El roadmap_id 19499, 19500, 19501 se conectan sólo por latitud-longitud inicial. Temporalmente su distancia será cero hasta que ésta sea arreglada

Y segundo se deben crear conexiones virtuales bajo el siguiente criterio: Dado un conjunto A que contiene sólo aquellos *roadmap_id* que se relacionan entre sí por latitud-longitud inicial y dado un conjunto B que contiene los *roadmap_related_id* del conjunto A, si para un *roadmap_id* del conjunto A no tiene conexión con algún *roadmap_related_id* del conjunto B, entonces esta conexión debe ser creada.

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
100835	*19499*	*19445*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*
100836	*19499*	*19446*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*
100837	*19499*	*19500*	*6.261870815*	*-75.6189179*	*6.26187082*	*-75.61891792*	*0*	*1*
100838	*19499*	*19501*	*6.261870815*	*-75.6189179*	*6.26187082*	*-75.61891792*	*0*	*1*
100839	*19499*	*45228*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*
100840	*19499*	*45246*	*6.261870815*	*-75.6189179*	*6.26182527*	*-75.61820717*	*78.72247*	*1*
100841	*19499*	*45251*	*6.261870815*	*-75.6189179*	*6.26192731*	*-75.61930483*	*43.22495*	*1*

Tabla 6.1.2.2 6: Los nodos *45246 y 45251 tienen conexión con 19501 y 19500 respectivamente, pero no con 19499*

- Crear la tabla *records_to_update* que contendrá un registro único entre los registros que se relacionan por latitud-longitud inicial, ya que la idea es tomar un punto único que abarque todas las relaciones de todo el conjunto de nodos, por ejemplo para el siguiente conjunto de datos:

roadmap_id	*roadmap_related_id*
19499	19500
19499	19501
19500	19499
19500	19501
19501	19499
19501	19500

Tabla 6.1.2.2 7: Conjunto de nodos conectados por latitud-longitud inicial

Se debe obtener el siguiente resultado:

roadmap_id	*roadmap_related_id*
19501	*19499*
19501	*19500*

Tabla 6.1.2.2 8: Reducción de la tabla 6.1.2.2 7

Ahora, ¿Qué hacer con estos registros?, Como el nodo 19501 va a ser el *roadmap_id* candidato que va a contener todas las relaciones existentes entre 19499 y 19500, entonces hay que actualizar en la tabla *st_buenos,* aquellos registros que no van a ser parte de las nuevas relaciones, en este caso los nodos 19499 y 19500. Por ejemplo, para los siguientes registros que están en *st_buenos*:

id	roadmap_id	roadmap_related_id	lat_start	long_start	lat_end	long_end	distance_meters	stretch_type
100	19499	45228	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1
101	19500	45251	6.261870815	-75.6189179	6.26192731	-75.61930483	43.22495	1
102	19501	45246	6.261870815	-75.6189179	6.26182527	-75.61820717	78.72247	1
103	19499	19445	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1
104	19499	19446	6.261870815	-75.6189179	6.26164019	-75.61892179	25.64846	1

Tabla 6.1.2.2 9: Conjunto de datos de la tabla st_buenos

Debe quedar actualizado así:

id	*roadmap_id*	*roadmap_related_id*	*lat_start*	*long_start*	*lat_end*	*long_end*	*distance_meters*	*stretch_type*
110	*19501*	*45228*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*
111	*19501*	*45251*	*6.261870815*	*-75.6189179*	*6.26192731*	*-75.61930483*	*43.22495*	*1*
112	*19501*	*45246*	*6.261870815*	*-75.6189179*	*6.26182527*	*-75.61820717*	*78.72247*	*1*
113	*19501*	*19445*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*
114	*19501*	*19446*	*6.261870815*	*-75.6189179*	*6.26164019*	*-75.61892179*	*25.64846*	*1*

Tabla 6.1.2.2 10: Conjunto de datos resultantes de la tabla 6.1.2.2 9

Por último se debe hacer dos inserciones en la tabla *street_relations*, la primera son los datos actualizados de la tabla *st_buenos* y la segunda son todas las conexiones del

roadmap_id candidato de la tabla st_fix_records_zero, que según el ejemplo que se viene siguiendo es el 19501, ya que insertar las conexiones de 19499 y 19500 sería poner en la tabla resultante datos redundantes

6.1.2.3 Modelo buses

Contiene toda la información de las rutas de transporte.

I. Definir los campos del modelo

Company: Es la compañía a la cual pertenece la ruta de bus.
Vehicle_type: Es el tipo de vehículo, ya se bus, buseta, colectivo.
Bus_stop_downtown: Indica si las vías del centro de la ciudad donde el vehículo hace paradas.
Route_length_km: Indica la distancia en kilómetros de la ruta.
Routes_taken: Indica las vías o lugares principales por donde pasa el vehículo.

6.1.2.4 Modelo buses_routes

Contiene todos los datos geográficos de las rutas de transporte.

I. Definir los campos del modelo

Roadmap_id: Es el punto por el cual pasa la ruta de transporte y que a su vez se encuentra en la tabla street_relations.
Lat_start: Es la latitud inicial del tramo.
Long_start: Es la longitud inicial del tramo.
Bus_id: Es el identificador de la ruta de bus.

Esta tabla debe contener exactamente la latitud-longitud inicial de la tabla *Roadmaps.*

6.1.2.5 ¿Cómo se crean modelos con Rails?

Para cada modelo con la instrucción *ruby script/generate migration nombreDelModelo* se crea un archivo en **db/migrate**, que contiene las definiciones a ser aplicadas en la base de datos, en el método *up* está el script que crea la tabla con sus campos y tipos de datos correspondientes y en el método *down* el script que la destruye:

```
class CreateBusesRoutes < ActiveRecord::Migration
 def self.up
  create_table :buses_routes do |t|
   t.column :roadmap_id, :integer, :null => false
   t.column :lat_start, :decimal, :precision => 15, :scale => 10, :null=>false
   t.column :long_start, :decimal, :precision => 15, :scale => 10,:null=>false
   t.column :bus_id, :integer, :null => false
  end

 end

 def self.down
  drop_table :buses_routes
 end
end
```

Luego con la instrucción *rake db:migrate* creará el modelo BusesRoutes en la base de datos y *rake:db:rollback* eliminará esta tabla de la base de datos.

6.1.3 ¿Cómo integrar Rails y el API de Google Maps?

Lo primero que se debe hacer es ir a la dirección Web *http://code.google.com/intl/es-419/apis/maps/signup.html* y registrarse para obtener una clave única que permitirá utilizar el API de Google Maps en el proyecto. Para un ambiente de desarrollo no se

necesita tener un dominio en la Web, en dirección URL[23] se debe poner *localhost* como se muestra en la figura 6.1.3 1.

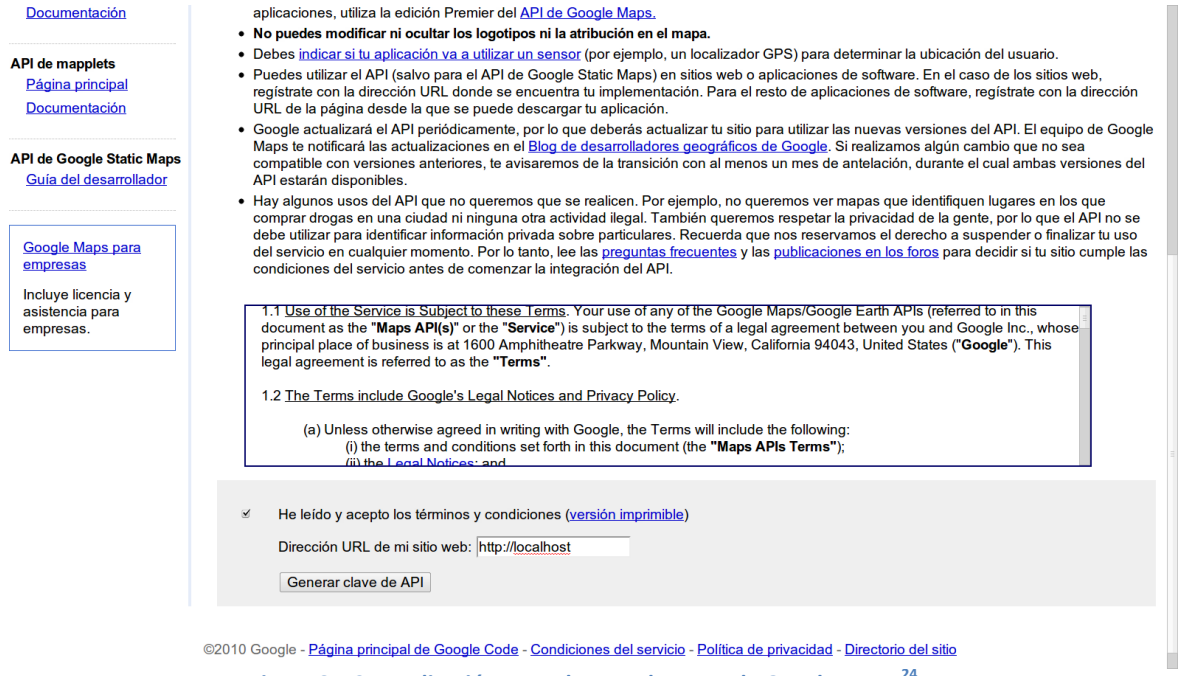


Figura 6.1.3 1: Indicación para obtener clave API de Google Maps [24]

Una vez obtenida la clave se debe incluir la siguiente directriz en la vista que va a contener el mapa:

```
<script
src="http://maps.google.com/maps?file=api&amp;v=2&amp;sensor=false&amp;key=ABQIAAAAUdGTwlglQe
27tBhbUDZYHhT2yXp_ZAY8_ufC3CFXhHIE1NvwkxSXcSyrjqkn7IGwmIX03cgwZAbuCA"
type="text/javascript"></script>
```

Ahora, mostrar el mapa es muy sencillo, en el <body> de la vista, se crea el elemento html que lo contendrá:

```
<body onload="load()" onunload="GUnload()">
  <div id="map" style="width: 500px; height: 300px"></div>
</body>
```

[23] URL(Uniform Resource Locator): Es un conjunto de caracteres para identificar un recurso en la web.
[24] Imagen tomada de: http://code.google.com/intl/es/apis/maps/signup.html

Y por último en **public/javascript/application.js** se hará el uso de todas las funcionalidades del API:

```
function load() {
   if (GBrowserIsCompatible()) {
      var map = new GMap2(document.getElementById("map")); //Crea el mapa
      map.setCenter(new GLatLng(37.4419, -122.1419), 13);  //Posiciona el mapa en una coordenada
     map.setUIToDefault(); //Crear los controles por defecto que tiene maps.google.com
   }
}
```

Como se puede apreciar, el código necesario para crear un mapa en la web no requiere de una configuración extensa y compleja. La figura 6.1.3 2 representa el código fuente anterior.

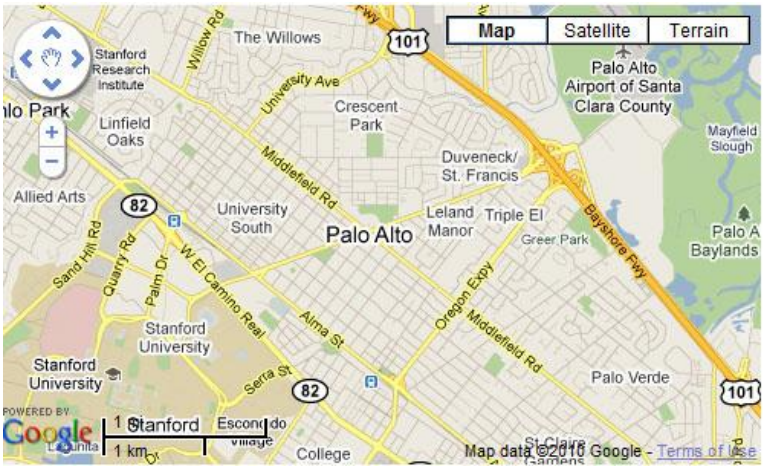


Figura 6.1.3 2 Ejemplo simple de la creación de un mapa

6.1.3.1 ¿Cómo hacer peticiones al servidor de Rails a través del API de Google Maps?

Para hacer una comunicación con el servidor, se tienen dos alternativas, utilizar el objeto que provee Google Maps llamado GXmlHttp o utilizar el objeto Ajax.request de librería Prototype. Para este proyecto se elige el objeto GXmlHttp para evitar incluir librerías alternas y para explorar más las funcionalidades de esta tecnología.

Una vez obtenidos los datos del *form*[25] que van a ser procesador por el servidor, se procede a hacer el llamado a este así:

```
function findRoute(){


 var init_lat_lng = init_lat+","+init_lng;
 var end_lat_lng = end_lat+ "," + end_lng;
 var getVars = "?initial_point="+init_lat_lng+"&end_point="+end_lat_lng;

 var request = GXmlHttp.create();
 request.open('GET', 'calculate'+getVars,true);
 request.onreadystatechange = function() {
  if(request.readyState == 4){
    var success=false;
    var content="Error contacting web service";
    try{
       res=eval("(" + request.responseText + ")" ); //Evalua los valores enviados por el método
                                                    // calcúlate a el controlador Map
       content=res.content;
       success=res.success;
    }catch (e){
     success=false;
    }
    if(!success) {
     alert(content);
    }
    else{
       parseContent(content);
    }
  }
 }
 request.send(null);
 return false;
}
```

La función *findRoute(),* obtiene los valores del *form* y concatena estos Strings para entregárselos al objeto GXmlHttp quien hace un request[26] bajo el método GET. Este objeto debe verificar qué sucedió con la petición cuando el controlador envió el resultado en formato JSON[27], esto lo hace mediante el método *onreadystatechange*, durante la petición la propiedad *readyState* puede tomar los siguientes valores:

[25] Un form es un formulario HTML que contiene datos que luego serán enviados al servidor
[26] Un request es una petición que se hace al servidor para solicitar un conjunto de datos.
[27] Léase JSON en glosario

0: uninitialized
1: loading
2: loaded
3: interactive
4: completed

La que debe ser evaluada es la número 4, que indica cuando la petición se completó para luego analizar el resultado enviado por el controlador. La figura 6.1.3.1 1 [3] representa lo anteriormente descrito.

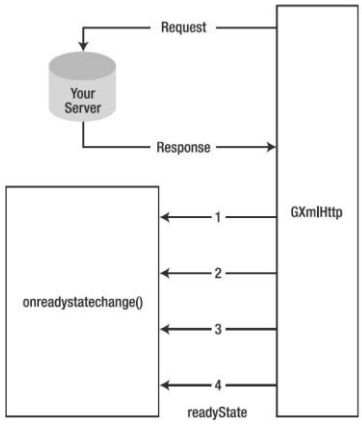


Figura 6.1.3.1 1: Diagrama de flujo de un GXmlHttp request[3]

Por otro lado, enviar el resultado obtenido por el método *calculate* es muy simple, estos valores se envían en formato JSON mediante la siguiente línea de código *render :text=variable.to_json,* por ejemplo:

```
class MapController < ApplicationController
  def calculate
    #Obtiene los parámetros enviados mediante el método GET y ejecuta la función que necesitará estos
parametros
    params_initial_point = params[:initial_point]
    params_end_point = params[:end_point]
    result = dijkstra(params_initial_point,params_end_point)
    #Si se obtuvo datos del método dijkstra entonces success y content obtendrán los valores correspondientes

    if result
      res={:success=>true, :content=>infoPath, :bus=>infoBus}
    else
      res={:success=>false,:content=>"Could not get the route"}
    end
```

```
    render :text=>res.to_json
  end
end
```

Adicional en este método se ejecuta al algoritmo de Dijkstra, el encargado de devolver un conjunto de nodos que luego deberán ser analizados sintácticamente. Las consideraciones y todo lo referente a este se explica a continuación.

6.1.4 Solución del camino más corto

6.1.4.1 Introducción a la teoría de grafos

Un grafo es un conjunto representado de la forma G = <V, E>, siendo V un conjunto de vértices o nodos y E un conjunto de aristas. Las aristas son un par no ordenado de vértices con el cual que se denota que dos elementos del conjunto V están conectados.

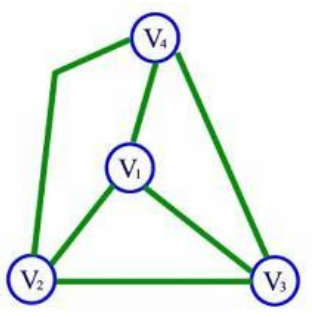


Figura 6.1.4.1 1: Grafo de ejemplo[28].

Este grafo representaría de la siguiente forma:

G = <V, E>

[28] Imagen tomada de http://www.ugr.es/~rruizb/cognosfera/sala_de_estudio/ciencimetria_redes_conocimiento/images/grafo.jpg

$V = \{V1, V_2, V_3, V_4\}$

$E = \{ (V_1, V_2), (V_2, V_1), (V_1, V_3), (V_3, V_1), (V_1, V_4), (V_4, V_1), (V_2, V_3), (V_3, V_2), (V_2, V_4), (V_4, V_2), (V_3, V_4), (V_4, V_3) \}$

Algunas propiedades de los grafos son[29]:

Adyacencia: Se dice que dos vértices son adyacentes si existe una arista que los relacione.

Caminos: Es un conjunto de vértices distintos donde dos vértices consecutivos son adyacentes.

Grafo convexo: Un grafo es convexo si para todo par de vértices distintos del grafo existe por lo menos un camino que los una.

Grafo completo: Un grafo es completo si cada vértice del grafo es adyacente a los demás vértices.

Grafo dirigido: Se dice que un grafo es dirigido cuando las aristas solo pueden recorrerse en un solo sentido.

Grafo múltiple: Un grafo es múltiple si en el grafo se presenta que dos vértices están conectados por más de una arista.

Subgrafo: Un subgrafo es aquel cuyo conjunto de vértices y de aristas son un subconjunto de un conjunto de vértices y aristas.

Peso o costo de una arista: Es un valor que se le da a una arista para que el grafo represente mejor el modelo.

Para la representación de un grafo computacionalmente, existen varios métodos:

Matriz de adyacencia: Un grafo se representa como una matriz de adyacencia creando una matriz cuadrada de orden n, siendo n el número total de vértices del conjunto V. Cada fila y cada columnas es un vértice de la matriz y si dos vértices a y b no están conectados, entonces el valor de (a, b) será 0. En caso contrario, el valor de (a, b) será 1 o si es un grafo donde las aristas tienen peso, entonces el valor de (a, b) será el peso de la arista que une los vértices a y b. Si el grafo es no dirigido, la matriz de adyacencia es simétrica y se puede optimizar si se crea como una matriz triangular[30].

[29] Agustín Gravano. Estudio de problemas, propiedades y algoritmos en grafos arco-circulares y circulares. Disponible en Internet: http://www.glyv.dc.uba.ar/agustin/files/gravano_2001.pdf

[30] Weisstein, Eric W. "Adjacency Matrix." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/AdjacencyMatrix.html

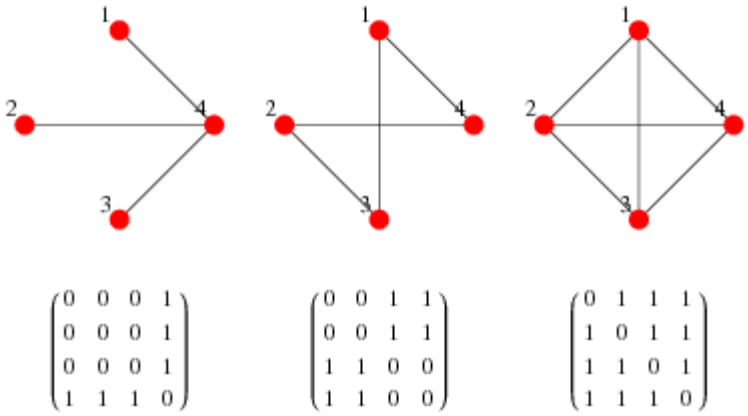


Figura 6.1.4.1 2: Ejemplo de matriz de adyacencia.

- Lista de adyacencia: Una lista de adyacencia es una lista de donde cada nodo de la lista es un vértice de la matriz y esta tiene una lista de los vértices con los que se es adyacente[31].

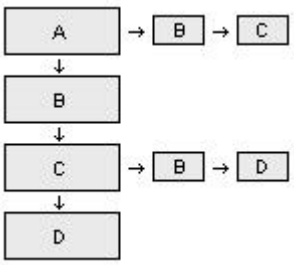


Figura 6.1.4.1 3: Ejemplo de una lista de adyacencia[32].

[31] Weisstein, Eric W. "Adjacency List." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/AdjacencyList.html

[32] Imagen tomada de http://www.brokendust.com/Projects/Images/3DGraphRepresentationAdjacencyList.jpg

6.1.4.2 Tratamiento del grafo de la ciudad

La representación gráfica más común de un grafo es dibujar los vértices como puntos y las aristas como líneas que conectan los vértices. Si se mira un grafo que tenga como conjunto de vértices V todas las esquinas de la ciudad y conjunto E las calles que unen las esquinas y el peso de las aristas como la distancia entre dos esquinas, se obtiene que un grafo modela de forma óptima las conexiones viales de una ciudad.

La mejor forma de representar la ciudad en un grafo sería como un grafo dirigido, múltiple, convexo y adyacente, pero lamentablemente se encontró que los datos con los que contaba para establecer el grafo de la ciudad estaba incompleto ya que la información de las calles no cumplía con la propiedad de grafo dirigido. Esto significó que el programa no sea capaz de reconocer el sentido real de las calles ni las prohibiciones de giro en las intersecciones, entre otras situaciones particulares a la malla vial.

Para la representación del grafo computacionalmente se eligió la lista de adyacencia. El motivo por el cual se rechazó la matriz de adyacencia es que si se mira una ciudad, esta puede tener cientos o miles de calles y cada una de éstas tienen entre dos a seis conexiones, por lo que la matriz demandaría mucho espacio en memoria para posibles conexiones que no ocurren.

A la representación de la lista de adyacencia se le hizo unas modificaciones: Además de incluir el vértice con el que se conecta, se agregó también el peso de la arista y la latitud y la longitud del vértice que se esté analizando.

A pesar de tener los datos en una base de datos, se optó por tener un archivo plano en disco con la lista de adyacencia del grafo por motivos de rapidez en la ejecución del programa

Como la lista tiene varios valores, se hizo un parser (analizador sintáctico) que leyera el archivo plano y lo convirtiera en una lista enlazada. El archivo plano tiene como patrón que el nodo que se analice sea seguido de dos puntos (:), la arista y su peso estén separados con una coma (,), las diferentes aristas estén separadas con un punto y coma (;), la coordenada esté separada con un símbolo de ampersand (&) y cada nodo de la lista esté separado por un salto de línea. A continuación se verá un ejemplo de cómo se vería un elemento de la lista:

A: B,x;C,y;D,z&lat,long

Siendo A el nodo que se analiza, B, C y D los nodos con los que se tiene una arista siendo x, y, z el peso de cada arista y por último la coordenada (lat, long) asignada al nodo A.

El proceso de análisis sintáctico del parser se hace de la siguiente forma:

1. Se lee una línea del archivo como una cadena de caracteres.
2. Se separa la línea con el carácter de dos puntos (:), creando con esto dos nuevas cadenas de caracteres. La primera cadena de caracteres es el nodo y la segunda cadena de caracteres son las aristas, los pesos y la coordenada del nodo.
3. Se separa la nueva cadena de caracteres con el carácter de ampersand (&), separando con esto la coordenada de las aristas y sus pesos.
4. Se separa la coordenada con el carácter de coma (,) y se le asignan los valores a la latitud y longitud del nodo.
5. Se separa la cadena de caracteres de aristas y pesos con el carácter de punto y coma (;), creando con esto una lista de cadenas de caracteres que contiene a las aristas y sus pesos.

6. Cada elemento de la lista de aristas y pesos se separa con el carácter de coma (,) y se agrega a una lista de enlaces como objetos separados.

7. Se guarda las variables en un objeto y se agrega a la lista del grafo.

8. Si no es el final del archivo, repita 1

6.1.4.3 Solución de cómo encontrar el camino más corto utilizando el algoritmo de Dijkstra

Una vez terminado el proceso de análisis sintáctico, se procede a buscar un camino entre los dos puntos. Los dos puntos que se tomarán en cuenta son la coordenada inicial y la coordenada final que el usuario ingresa por la interfaz gráfica del programa.

Es probable que los usuarios seleccionen como punto inicial una casa o lugar que esté entre dos esquinas; dado que el sistema establecido sólo tiene las coordenadas de las esquinas de la ciudad, se tomará el dato que el usuario ingrese y se calculará cuál es la esquina más cercana a este punto para establecerla como punto inicial.

Para realizar este cálculo, se tomará la latitud y la longitud inicial que ingrese el usuario y se hará una búsqueda en la base de datos para encontrar los ID de los registros que estén a un radio de 500 metros de la coordenada del usuario, utilizando la fórmula de Haversine y se utilizará el vértice que esté más cerca. El mismo proceso se hará con la latitud y la longitud final del usuario y con esto se obtendrá los dos puntos con los cuales el sistema calculará el camino más corto.

El problema del camino más corto tiene varias soluciones y en el momento las mejores formas de resolver este son usar el algoritmo de Bellman-Ford-Moore o el algoritmo de Dijkstra. El algoritmo de Bellman-Ford-Moore es el más rápido que se conoce en el momento si el grafo tiene pesos negativos, mientras que el algoritmo de Dijkstra

ofrece mejores resultados si el grafo sólo tiene pesos no negativos. Para el grafo del problema, una situación con una distancia negativa es ilógica, por lo que se optó por utilizar el algoritmo de Dijkstra [14].

Un pseudo código del algoritmo original es el siguiente [15]:

```
función Dijkstra(Grafo g, vértice fuente, vértice final)
  Para cada vértice en g
    dist[v] := infinito
    prev[v] := nulo
  fin para
  dist[fuente] := 0
  Q := lista de todos los nodos de g
  Mientras que Q no este vacío
    u := vértice de menor distancia en dist[]
    si dist[u] = infinito
      terminar
    fin si
    eliminar u de Q
    para cada vecino v de u
      si dist[u]+peso(u,v) < dist[v]
        dist[v] := [u]+peso(u,v)
        prev[v] := u
      fin si
    fin para
    retornar dist[]
fin Dijkstra
```

Descrito en lenguaje accesible, el algoritmo de Dijkstra lo que hace es recorrer todos los vértices del grafo g empezando por los que tengan aristas con los pesos menores, determinando con cuáles vértices conecta éste y se actualiza qué distancia hay desde el vértice de origen al vértice que se esté revisando. Si se encuentra un camino que tenga menor peso que el que se tiene en el arreglo de distancia, entonces se actualiza esta información.

Para la solución del problema, se le realizaron modificaciones a este algoritmo. El primero fue que no se revisaría todo el grafo sino un subgrafo de vértices que estarían a una distancia variable, dependiendo del vértice inicial y del vértice final. Para encontrar el subgrafo, se calcularía primero cuál es la distancia que hay entre el vértice inicial y el vértice final con la función de Haversine y luego se le sumaría un tercio de ésta a la distancia y con esto se sacaría la distancia máxima a la que puede estar un vértice del vértice inicial para ser evaluado.

Por ejemplo, si el vértice inicial tiene como coordenada 6.200821,-75.577762 y el vértice final es 6.210400, -75.571110, la distancia calculada por la formula Haversine es 1.294 Km., luego la distancia se divide por 3 lo que da 0,43133km. Con estos datos, se encuentra que la distancia máxima sería 1.294 + 0,43133km = 1.72533km. Cualquier vértice que esté a más de 1.72533km del vértice inicial, no sería evaluado por el algoritmo de Dijkstra, evitando con esto evaluar vértices que no sean necesarios para encontrar el camino óptimo. Una forma que se puede confirmar esto es tomando como vértice inicial un punto en el centro de la ciudad y establecer como el vértice final un punto al norte de ésta. El camino más corto no se verá afectado si no revisamos vértices que estén ubicados en los barrios al sur del punto establecido como inicial, lo que resulta en una disminución en el tiempo de ejecución del algoritmo ya que el algoritmo no recorrerá todos los nodos del grafo, sino un conjunto de estos.

Otro cambio que se le hizo al algoritmo fue reemplazar la estructura de datos Q del algoritmo por una estructura Heap en lugar de una lista. Esto se hace ya que el algoritmo supone que la búsqueda del vértice con menor distancia sea O(1), pero si se maneja una lista, esto tendrá O(n), lo que afecta gravemente la complejidad aumentando el número de ciclos que se tienen que ejecutar en el algoritmo. La estructura Heap es una estructura de datos donde, para este caso, el menor siempre estará en el tope de la estructura, permitiendo con esto que encontrar el menor valor sea O(1), cumpliendo con esto la condición que pide el algoritmo.

El último cambio que se hizo es que el algoritmo no devuelve el arreglo de distancias, sino el camino más corto entre el vértice inicial y el vértice final. Esto se logra tomando los previos de cada vértice, empezando desde el vértice final hasta llegar al vértice inicial. El pseudo código de esta operación es el siguiente:

```
v:= vértice final
Mientras que prev[v] ≠ vértice fuente
  Insertar en camino el vértice v
  v:=prev[v]
Fin mientras
Insertar en camino el vértice inicial
```

Hasta este punto, se tiene el camino más corto entre el vértice inicial y el vértice final, el paso siguiente sería encontrar qué rutas de buses son óptimas para conectar estos vértices.

6.1.4.4 Búsqueda de rutas de transporte para el usuario

Este procedimiento se hace en dos etapas: encontrar un único bus que conecte el vértice inicial y el final y en caso que no exista, encontrar cuáles son los buses que más se adaptan al camino óptimo que entrega el algoritmo de Dijkstra.

Para encontrar si existe un bus que conecte el vértice inicial y el vértice final, se hace nuevamente una búsqueda en la base de datos como se hizo para encontrar los vértices que estuvieran cerca a los puntos que el usuario ingresó, sólo que esta vez no se toma el que más cerca esté sino un conjunto de nodos cercanos. Luego, se toma cada elemento de la búsqueda de vértices y se busca cuáles rutas de buses pasan por

dicho vértice. El mismo procedimiento se hace para el vértice final. El último paso consiste en hacer una comparación entre las búsquedas de buses. Si existe una o más rutas de buses que conecten los vértices cercanos al inicio y al final, se pasará a dibujar en el mapa el trayecto completo de la ruta para que el usuario pueda ver el recorrido que hace el bus, en que esquina se toma el bus y en que esquina se debe bajar de éste. Si por el contrario no se encontró ninguna ruta de bus que pasara cerca al inicio y al final, se procede a buscar una combinación de rutas de buses que juntas sean las más aproximadas al camino más corto.

Para encontrar la combinación de rutas de buses óptima se toma todos los vértices del camino más corto, una vez más se hace una búsqueda en la base de datos que retorne los nodos cercanos al vértice que se está analizando y luego se busca cuál ruta de buses pasan cerca a éste, dando como resultado un arreglo de buses, donde cada ruta de bus tiene como característica que una parte de su recorrido es similar en alguna parte de su recorrido con camino más corto. Con esto, se puede determinar si el usuario debe tomar dos o más rutas de buses para ir desde su origen hasta su destino recomendando las rutas de buses que sean lo más similares posibles al camino más óptimo.

6.1.5 Interpretación de datos

Una vez ejecutado el algoritmo de Dijkstra, este devolverá un conjunto de nodos que representaran el camino a seguir desde el lugar de origen hasta el lugar de destino, para cada nodo se debe obtener los datos disponibles en el modelo de datos como sus coordenadas geográficas, nomenclatura de la dirección, entre otros. Pero no se tiene la orientación y la dirección a la que se debe girar.

Para obtener la orientación en grados geométricos se debe aplicar la siguiente fórmula[33]:

$$\theta = atan2(\ sin(\Delta long).cos(lat_2), cos(lat_1).sin(lat_2) - sin(lat_1).cos(lat_2).cos(\Delta long)$$

Donde:

atan2: Es la variación de la función arcotangente, recibe 2 argumentos (x,y) y el resultado es devuelto en ángulos radianes.

Δlong: Es el diferencial de dos puntos longitud, dados en radianes

lat_1: Es la latitud inicial dada en radianes

lat_2: Es la latitud final dada en radianes

Estos grados se deben traducir en direcciones Norte, Nororiente, Oriente, Suroriente, Sur, Suroccidente, Occidente, Noroccidente. La siguiente tabla muestra los limites dados en grados para ubicar cada dirección.

Orientación(Grados geométricos)	***Dirección***
Entre 0 y 22.5 ó entre 337.5 y 360	Norte
Entre 22.5 y 67.5	Nororiente
Entre 67.5 y 112.5	Oriente
Entre 112.5 y 157.5	Suroriente
Entre 157.5 y 202.5	Sur
Entre 202.5 y 247.5	Suroccidente
Entre 247.5 y 292.5	Occidente
Entre 292.5 y 337.5	Noroccidente

Tabla 6.1.5 1 Codificación de grados geométricos a direcciones cardinales

Hallados estos dos campos, se necesita saber por dónde se debe girar dada una dirección inicial y una dirección final. Por ejemplo, la lógica utilizada para saber hacia dónde girar estando en dirección norte se representa en la siguiente tabla.

[33] http://www.movable-type.co.uk/scripts/latlong.html

Dirección Inicial	*Dirección Final*	*Giro hacia*
Norte	Oriente	Derecha
Norte	Noroccidente o Suroccidente	Ligeramente hacia la izquierda
Norte	Nororiente o Suroriente	Ligeramente hacia la derecha
Norte	Occidente	Izquierda

Tabla 6.1.5 2 Lista de opciones a tomar tomando como dirección inicial Norte

6.1.6 Interfaz gráfica

Como se mostró anteriormente, crear un mapa resulta ser una tarea muy sencilla. Gracias a las extensas funcionalidades del API de Google Maps se pueden crear funciones que permiten la creación de elementos gráficos sobre el mapa, a continuación se hace énfasis en aquellos elementos más relevantes que hicieron parte de la elaboración de la interfaz gráfica de este proyecto.

Menú desplegable: Este menú, es de vital importancia para esta aplicación, el cual permitirá crear de forma fácil los dos puntos (Origen y Destino) necesarios por la aplicación, los cuales serán enviados al servidor para que luego estos puedan ser procesados.

```
//Evento para desplegar menú cuando se hace click izquierdo
GEvent.addListener(map,"singlerightclick",function(pixel,tile) {
  var clickedPixel = pixel;
  var x=pixel.x;
  var y=pixel.y;
  if (x > map.getSize().width - 120) { x = map.getSize().width - 120 }
  if (y > map.getSize().height - 100) { y = map.getSize().height - 100 }
  var pos = new GControlPosition(G_ANCHOR_TOP_LEFT, new GSize(x,y));
  pos.apply(contextmenu);
  contextmenu.style.visibility = "visible";
  point=map.fromContainerPixelToLatLng(clickedPixel);
  lat=point.lat();
  lng=point.lng();
```

}

El anterior fragmento de código, captura un evento cuando el usuario hace click derecho sobre el mapa y obtiene la posición x,y del pixel correspondiente del mapa. Luego crea el objeto *GControlPosition*, quien recibe 2 parámetros, la posición relativa donde va a estar y el tamaño en pixeles del panel, una vez creado el HTML que compone el menú desplegable, este se aplica mediante *pos.apply(contexmenu)* y con *contextmenu.style.visibility = "visible";* para que este sea visible. Por último *point=map.fromContainerPixelToLatLng(clickedPixel);* obtiene la latitud longitud del punto donde hizo click. La figura 6.1.6 1 muestra el menú que se crea cuando el usuario hace click derecho sobre el mapa.

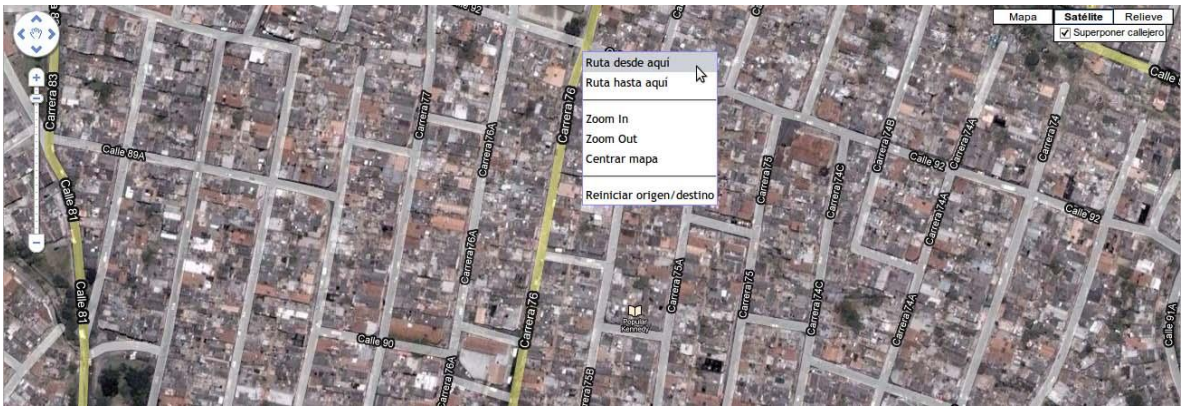


Figura 6.1.6 1: Menú desplegable

Markers: Los markers o marcadores son elementos que identifican puntos sobre un mapa, a través de estos se puede obtener la latitud y la longitud de un punto dado. Entre las muchas funciones que abarcan, las utilizadas en este proyecto fue cambiar los iconos que poseen por defecto Google Maps para diferenciar el punto inicial del punto final, además de permitir que este pueda ser arrastrado por todo el mapa. En el fragmento de código Javascript siguiente se crea un objeto *GIcon* donde se configura todo lo relacionado a la parte visual del icono como la imagen que va a contener y su sombra correspondiente (*image, shadow*), su tamaño en pixeles (*iconSize*), y su posición

relativa al mapa (*anchor*), una vez hecho esto el objeto *GMarker* recibe la coordenada geográfica donde se va a crear el marker y adicional va a permitir que este pueda ser arrastrado, cuando se arrastre se lanzará un evento que obtendrá de nuevo la coordenada latitud-longitud correspondiente del marker.

```
var init_icon = new GIcon();
init_icon.image = "http://www.google.com/mapfiles/dd-start.png";
init_icon.shadow = "http://www.google.com/mapfiles/shadow50.png";
init_icon.iconSize = new GSize(20,34);
init_icon.iconAnchor = new GPoint(9,34);
init_icon.shadowSize = new GSize(37, 34);
init_icon.infoWindowAnchor = new GPoint(24,24);
var initial_marker = new GMarker(point,{draggable:true,icon:init_icon});
map.addOverlay(initial_marker);
GEvent.addListener(initial_marker, "dragend", function() { //Lanza un evento si se arrastra el marker
  init_lat=initial_marker.getPoint().lat();
  init_lng=initial_marker.getPoint().lng();
}
```

En la figura 6.1.6 2 muestra la creación de dos markers con iconos personalizados.

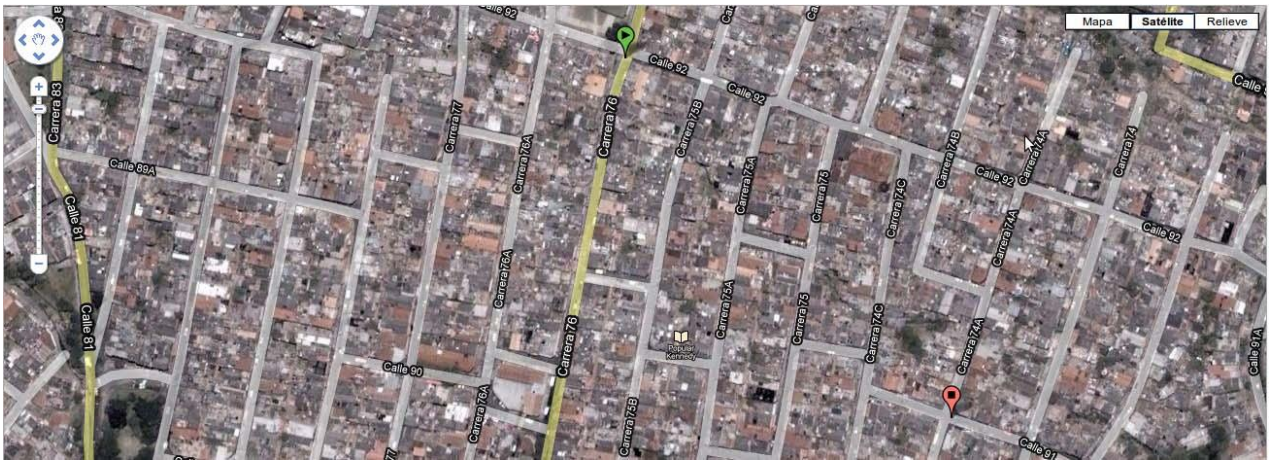


Figura 6.1.6 2: Creación de markers

Polylines: Esta es una superposición de mapa que dibuja una polilínea sobre este utilizando para ello las funciones de dibujo vectorial del navegador si las hay o, si no, una superposición de imagen desde los servidores Google[34]. Estas polilíneas van a ser la representación del conjunto de coordenadas geográficas de los nodos devueltos por el algoritmo de Dijkstra. La siguiente función muestra como pintar una polilínea:

```
function drawPolyline(latlng_street){
  var polyline = new GPolyline(latlng_street,'#FF6633',7,1);
  map.addOverlay(polyline);
}
```

El objeto *GPolyline* recibió tres parámetros, *points* que es el array latlng_street que contiene todos los objetos *GLatLng*, quienes poseen todas las coordenadas geográficas del conjunto de nodos a pintar, *color* que es el valor en hexadecimal del color a representar, *weight* corresponde a la anchura de la línea en pixeles y *opacity* que representa el grado de transparencia de la línea, variando desde el número cero hasta el uno. La figura 6.1.6 3 representa un conjunto de datos a pintar.

Figura 6.1.6 3: Polilíneas

[34] http://code.google.com/intl/es/apis/maps/documentation/mapplets/reference.html#GPolyline

CONCLUSIONES

- Se desarrolló una aplicación web en base una herramienta de mapas en línea, en este caso Google Maps, donde un usuario elige dos puntos de la ciudad y ésta entrega una ruta óptima que debe ser transitada a pie.

- Se integró satisfactoriamente las rutas de buses al modelo de datos de la aplicación, mediante un algoritmo el usuario podrá ver que rutas de buses transitan cerca a la ruta sugerida por la aplicación.

- Se logra dar respuesta clara al usuario sobre que ruta es la más adecuada para llegar a su destino.

- Para que el algoritmo de Dijkstra logre un buen desempeño, hace falta del uso de una estructura de datos más avanzada que una lista simple, por ejemplo una cola de prioridad o un montículo (*heap)*. En este proyecto se utilizó la estructura de datos (*heap)*, la cual ayudó considerablemente a reducir los tiempos de procesamiento del algoritmo, por ende la respuesta al usuario se entrega en un tiempo prudente.

- Para cualquier aplicación y sobre todo para las aplicaciones que manejan bases de datos con información geográfica, un buen análisis previo de la calidad de la información contenida en la base de datos es crucial para desarrollar una aplicación de este tipo.

- En base a la experiencia de uso de otros lenguajes de programación, se evidenció que el framework utilizado (Rails) redujo el tiempo necesario que

puede necesitar otros frameworks de aplicaciones web basado en lenguajes como Java, Php y .Net para crear una comunicación entre los componentes que hacen de una aplicación web que implementa el API de Google Maps.

- La aplicación desarrollada se programó para que sus componentes fueran manejados por capas, un ejemplo de esto fue el modelo de datos, en el cual se tiene la ventaja que si se actualizan los datos de la malla vial de Medellín no es necesario hacer cambios drásticos en el modelo.

- Como mejora, se podría integrar más información sobre las rutas de buses e ingresar el sistema metro de la ciudad, para que la aplicación ofrezca más posibilidades a la hora de elegir una ruta de transporte.

- Para manejar información geográfica es aconsejable utilizar base de datos espaciales si se quiere mejorar los resultados de búsqueda, pero existen consultas que SQL que bien diseñadas pueden generar resultados en tiempos óptimos.

BIBLIOGRAFÍA

[1] Dave Thomas, David Hansson, Leon Breedt, Mike Clark, James Duncan Davidson, Justin Gehtland, Andreas Schwarz. Agile Web Development with Rails. Tercera Edición. The Pragmatic Programmers LLC.2006.

[2]Gellert W., S. Gottwald, M. Hellwich, H. Kästner y Küstner H., La Enciclopedia concisa de las Matemáticas VNR , 2 ª ed., Cap. 12 (otras unidades: New York, 1989).

[3] Andre Lewis, Michael Purvis, Jeffrey Sambells and Cameron Turner, Beginning Google Maps Applications with Rails and Ajax From Novice to Professional. Primera Edición. Jason Gilmore. 2007.

[4] Dave Thomas, Chad Fowler, Andy Hunt. Programming Ruby The Pragmatic Programmers' Guide Second Edition. Segunda Edición. The Pragmatic Programmers LLC. 2005.

[5] José Carmona, Julián Gámez, Luis Giradlo. Campus Móvil. Universidad EAFIT. 2009.

[6] Juan Muñoz, Juan Pemberthy. Visualización del mercado de acciones en 3D. Universidad EAFIT. 2009.

[7] Mike Williams. Google Maps API Tutorial. 2010. Disponible en Internet: http://econym.org.uk/gmap/index.htm.

[8] Brown university. Using GIS and the Internet to Produce a Cultural Resource Inventory for South Kingstown, RI. Disponble en Internet: http://envstudies.brown.edu/oldsite/Thesis/2001/james/gishistory.html.

[9] The guide of Geographic Informations Systems. Disponible en Internet: http://www.gis.com/es/content/who-uses-gis.

[10] Instituto de Investigación de Recursos Biológicos Alexander von Humboldt. *LABORATORIO DE BIOGEOGRAFÌA Y ANÀLISIS ESPACIAL.* Disponible en internet: http://www.humboldt.org.co/humboldt/mostrarpagina.php?codpage=7000.

[11] Ciudad en linea.net. Disponible en Internet: http://www.ciudadenlinea.net.

[12] Seguridad en línea .net. Disponible en Internet: http://www.ciudadenlinea.net.

[13] Transporte en línea .net Disponible en Internet: http://www.ciudadenlinea.net.

[14] BV Cherkassky, AV Goldberg, T Radzik: Shortest paths algorithms: theory and experimental evaluation. Disponible en Internet: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.752.

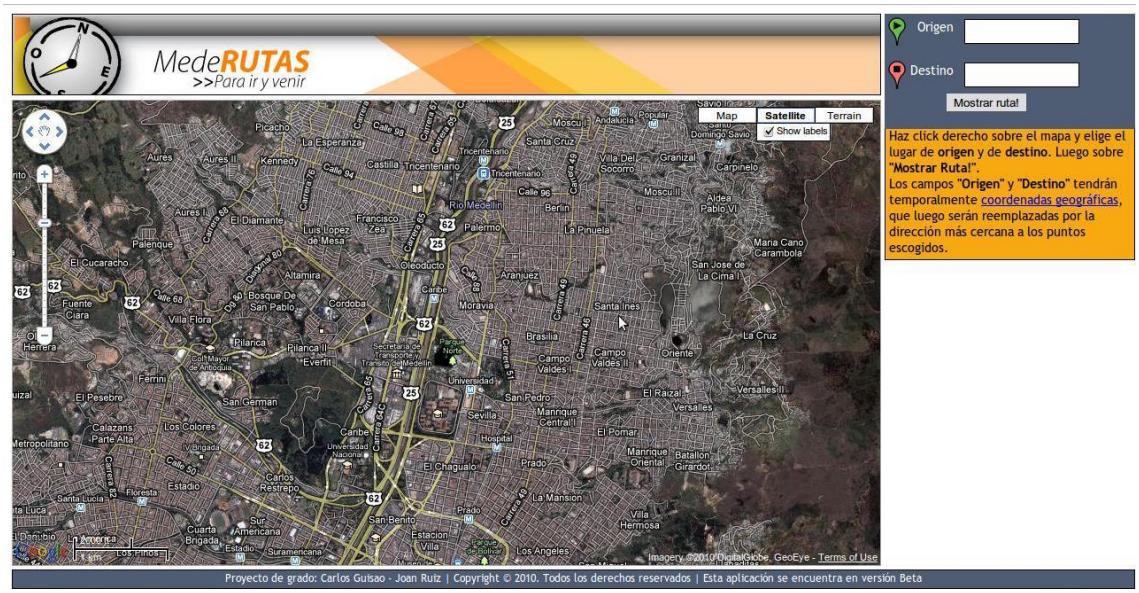
[15] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Introduction to Algorithms. Segunda edición. McGraw-Hill.2002.

[16] Alexander Rubin. Geo/Spatial Search with MySQL. MySQL. 2006. Disponible en internet: http://www.scribd.com/doc/2569355/Geo-Distance-Search-with-MySQL.

MANUAL DE USUARIO

Esta aplicación funciona para calcular cuál es el camino más corto entre dos puntos de la ciudad que el usuario seleccione y encontrar qué rutas de buses debería de tomar para su desplazamiento

Cuando el usuario ingrese en la aplicación, deberá ver en su navegador lo siguiente:



La aplicación tiene varias opciones y menús para facilitarle el movimiento por la ciudad en el mapa y el uso de la aplicación. Empezando por la esquina superior izquierda del mapa se podrá ver una opción con la que se podrá desplazar por el mapa al presionar las flechas. También se puede mantener presionado el botón izquierdo del mouse y moverse por el mapa a gusto.

Debajo de la opción de navegación, se encontrará una barra con la que se puede acercarse o alejarse en el mapa. Este *zoom* servirá para encontrar puntos más específicos en el mapa o ampliar la vista del mapa.

En la esquina superior derecha del mapa hay tres botones: *mapa, satélite* y *relieve*. Con el botón de *map* se podrá ver un mapa plano de la ciudad, mostrando sólo las calles como una línea blanca y lo demás como áreas grises. El botón *satélite* convierte el mapa en imágenes que son fotos aéreas de la ciudad. Ésta es la opción que viene predeterminada cuando se inicia la aplicación. El botón *satélite* tiene una opción más llamada "Superponer callejero". Con ésta se podrá ver la cuadricula de las calles de una forma más clara.

A la derecha de la aplicación se puede ver dos campos de texto donde irá la dirección del punto inicial, la dirección del punto final y un campo en blanco donde luego se explicara la ruta optima. Estos campos se llenarán con la información de los puntos que se seleccionen con el menú de click derecho que se explicará más adelante.

Debajo del campo donde se verán los puntos seleccionados, se encuentra un área en blanco. Esta se usara para explicar el camino una vez sea calculado.

Si se hace click derecho sobre el mapa, saldrá un nuevo menú:

Con este menú, se selecciona el punto inicial si se hace click sobre la opción “Ruta desde aquí”, el punto final seleccionando la opción “Ruta hasta aquí”. Las opciones “Zoom in” y “Zoom out” funcionan de la misma manera que la barra de zoom que ya se explicó. La opción “Centrar mapa aquí”, como su nombre lo indica, establece como centro del mapa el punto donde se hizo click derecho para sacar el menú. La opción “Reiniciar origen y destino” sirve para borrar los puntos seleccionados anteriormente.

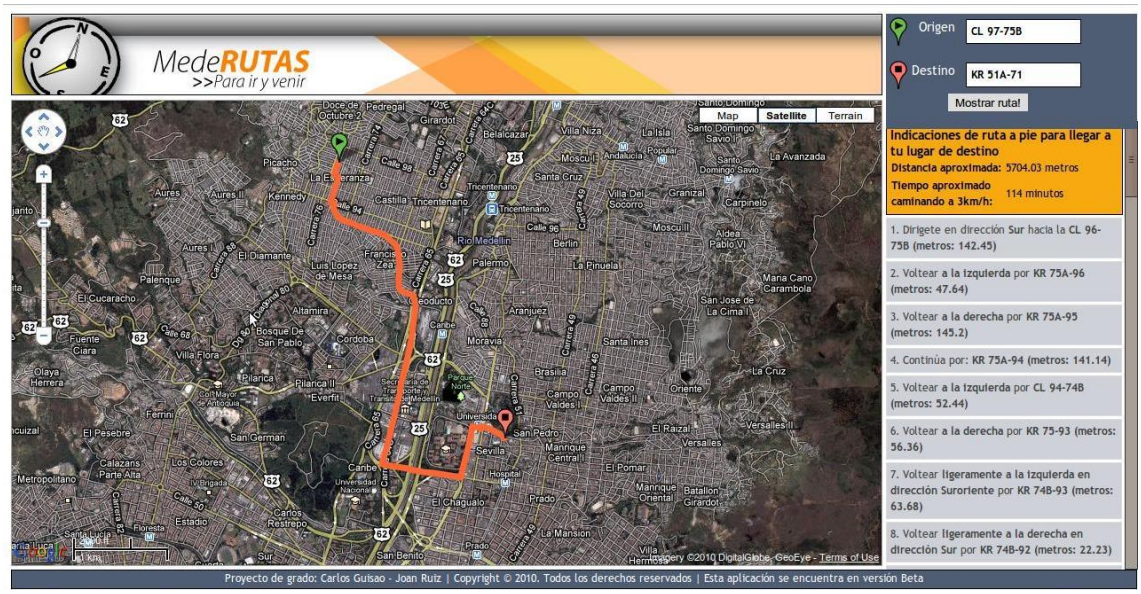
Para el cálculo del camino más corto entre dos puntos, se deberá seleccionar el punto inicial y el punto final con el menú que se despliega al hacer click derecho sobre el mapa.

Por ejemplo, si se quiere ir desde una calle del barrio La Esperanza hasta una calle cercana a la estación del metro Universidad. Para esto, se debe seleccionar el punto inicial con la opción “Ruta desde aquí” en la calle del barrio La Esperanza, luego se debe seleccionar el destino con la opción “Ruta hasta aquí”. El punto inicial quedará marcado con un pin de color verde y el punto final con un pin de color rosado.

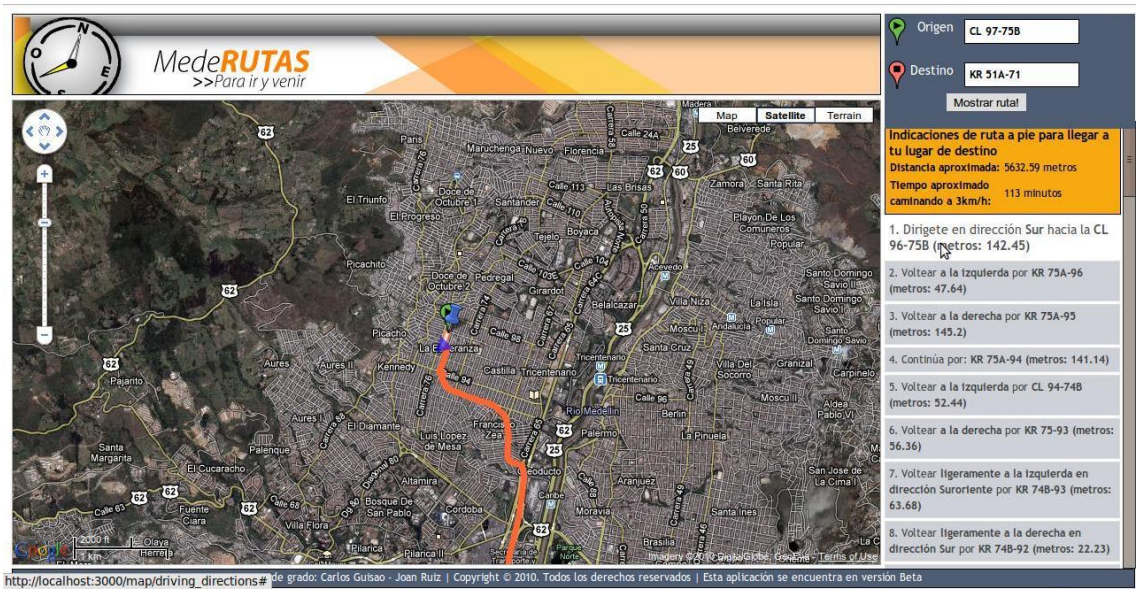
Ahora, se ejecuta la aplicación presionando el botón “Mostrar ruta” el cual se encuentra al lado derecho de la aplicación. El cálculo puede tomar unos cuantos segundos.

Cuando se termine el cálculo del camino más corto, este aparecerá sobre el mapa como una línea naranja y en el campo en blanco que había antes al lado derecho,

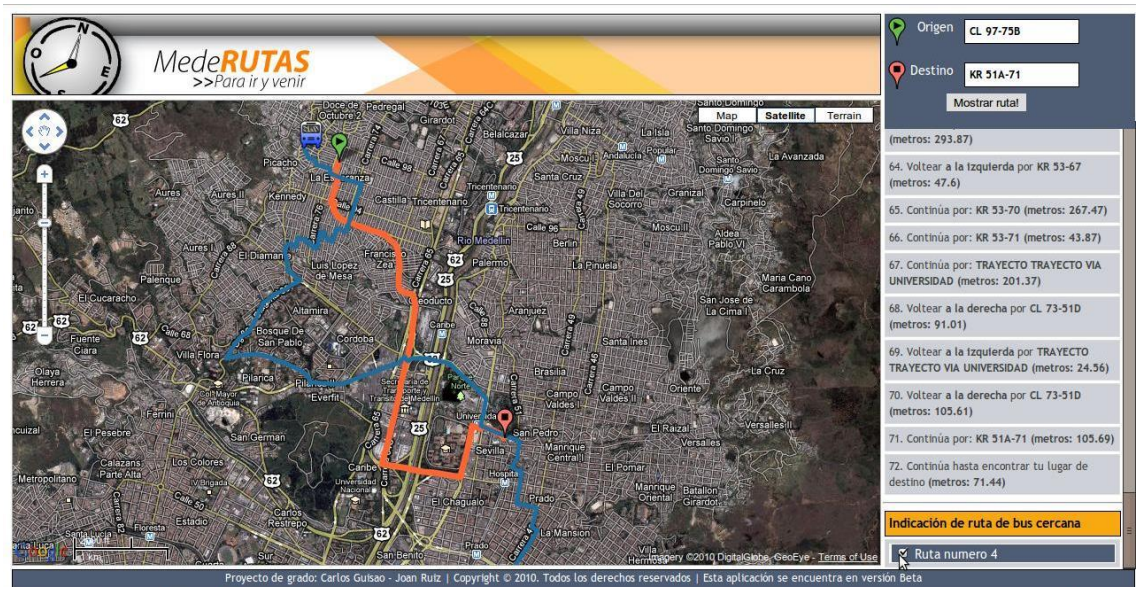
aparecerá una aproximación de la distancia y del tiempo que tomará recorrer este camino caminando a una velocidad de 3 Km/h que es la velocidad promedio de una persona. Además se verá una explicación detallada del camino.



En el desglose se podrá hacer click sobre cada elemento del trayecto, haciendo con esto que se centre el mapa sobre la parte del trayecto seleccionado y a la vez, este se resaltará con un color blanco, un pin azul aparecerá en el punto inicial del punto seleccionado y un triangulo azul sobre el final.



Al final del desglose punto por punto del camino más corto, aparecerán las rutas de buses que se podrán tomar para el desplazamiento. Al hacer click sobre el cuadro de selección de la ruta, en el mapa aparecerá una línea de color donde se mostrara todo el recorrido de ésta.



En caso que se deban tomar varias rutas de buses, el usuario podrá seleccionarlas todas y ver así más claramente cómo será su recorrido en las rutas de buses.