SITAPP: UNA APLICACIÓN INTELIGENTE PARA DISPOSITIVOS MÓVILES DEL SISTEMA DE RUTAS DE TRANSPORTE URBANO DEL MUNICIPIO DE PASTO

JUAN JOSÉ CHAMORRO
MARLON FABRICIO PASUY ESPINAL

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2018

SITAPP: UNA APLICACIÓN INTELIGENTE PARA DISPOSITIVOS MÓVILES DEL SISTEMA DE RUTAS DE TRANSPORTE URBANO DEL MUNICIPIO DE PASTO

JUAN JOSÉ CHAMORRO
MARLON FABRICIO PASUY ESPINAL

Trabajo de Grado presentado como requisito parcial para optar al título de Ingeniero de Sistemas

RICARDO TIMARÁN PEREIRA, PHD.

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE SISTEMAS
PROGRAMA DE INGENIERÍA DE SISTEMAS
SAN JUAN DE PASTO
2018

NOTA DE RESPONSABILIDAD

“Las ideas y conclusiones aportadas en la tesis de grado, son responsabilidad exclusiva de sus autores”.

Artículo 1o del acuerdo No 324 del 11 de octubre de 1966, emanado del honorable Consejo Directivo de la Universidad de Nariño.

“La Universidad de Nariño no se hace responsable de las opiniones o resultados obtenidos en el presente trabajo y para su publicación priman las normas sobre el derecho de autor”

Artículo 13, Acuerdo N. 005 de 2010, emanado del Honorable Consejo Académico.

Nota de aceptación:

_____
_____
_____
_____
_____
_____
_____

_____

Firma Presidente del Jurado

_____

Firma del Jurado

_____

Firma del Jurado

San Juan de Pasto, Abril de 2018

AGRADECIMIENTOS

Expresamos nuestro agradecimiento a la Universidad de Nariño por acogernos y formarnos como Ingenieros.

Al Sistema de Investigaciones de la Universidad de Nariño por el apoyo financiero otorgado para la ejecución de este proyecto.

Al asesor del trabajo de grado, profesor Ricardo Timaran, PhD, por su constante acompañamiento y por permitirnos formar parte del grupo de investigación GRIAS para iniciar nuestra vida investigativa.

A los profesores que fueron parte de nuestra formación, por todas las enseñanzas y conocimientos compartidos.

A los compañeros Yamid Estrada, David Eraso y Wilmer Escobar, por su colaboración y por los buenos momentos compartidos en el grupo de investigación.

A todos los compañeros, amigos y demás personas que formaron parte de nuestra formación académica, por el apoyo y colaboración en esta importante etapa.

DEDICATORIA

A mis padres, Yuliet y Fabricio por sus enseñanzas, constante apoyo, ser mis guías y mi fortaleza siempre, son la fuente de inspiración para superarme cada día más.

A mi hermano Sergio, mis abuelos Campo e Iraida, mis tíos y tías, todas las enseñanzas han sido fundamentales en mi formación como persona.

A mis primos Silvia, Iván, Daniel, Valery y Camila, ha sido fundamental su compañía en cada momento de mi vida.

A mis amigos Deivyd y Diana por acompañarme durante estos años de amistad y estar en los momentos difíciles.

A Cristina, por su compañía, amor y apoyo incondicional en los momentos compartidos en esta etapa crucial de mi formación profesional y personal.

Marlon Fabricio Pasuy Espinal

DEDICATORIA

A mi madre, María Isabel Chamorro. Mi constante fuente de inspiración. La persona que con su buen ejemplo y sus sabias palabras me ha ayudado a ser una persona de bien, a esforzarme por alcanzar mis metas, pero más que todo por el amor y el apoyo incondicional. La persona que ha estado en cada momento de mi vida. A ti madre, gracias por todo. Hoy logramos cumplir una meta compartida, tu sueño de ver a uno de tus hijos ser profesional y mi anhelo por ser ingeniero.

Juan José Chamorro

RESUMEN

En este documento se presentan los resultados del proyecto de investigación cuyo objetivo fue construir un APP inteligente para dispositivos móviles Android, bajo software libre, para el sistema estratégico de transporte público del municipio de Pasto (SETP) denominada SITApp.

SITApp está acoplada con una ontología de zonas de la ciudad y un buscador semántico, que facilita a los usuarios a partir de su sitio origen, encontrar la ruta más apropiada y óptima para llegar a su sitio destino. Esta aplicación cuenta también con un módulo administrativo para la completa actualización de información de rutas y paraderos del sistema.

Esta aplicación permite a los usuarios del transporte público obtener fácilmente, información en tiempo real sobre rutas y paraderos de la ciudad así como de los sitios de interés que se encuentran alrededor de cada paradero del SETP, ubicándolos sobre una cartografía del municipio de Pasto (Colombia).

ABSTRACT

The present work shows the results obtained in the research proyect whose objective was to build an intelligent APP for mobile devices Android, under open source for the strategic system of public transport of municipality of Pasto (SETP).

SITApp is coupled with an ontology of areas of the city and a semantic search engine which facilitates users finds the most appropriate and optimal route to reach from their origin to their destination site. This application also has an administrative module for the full update of information on routes and stops of the system.

This application allows users of public transport to obtain easily real-time information about routes and stops in the city as well as the sities of interest that are around each stops of the SETP, locating them on a map of the municipality of Pasto (Colombia).

***Keywords*:** SITApp, Android, Semantic search engine, Web Service, Satellite tracking.

TABLA DE CONTENIDO

LISTA DE FIGURAS

LISTA DE TABLAS

INTRODUCCIÓN

El crecimiento demográfico genera, entre otros, una necesidad de formas de transporte cada vez más eficientes. En las ciudades donde el transporte público es deficiente, los habitantes hacen uso de medios de transporte no formales, los cuales fueron creados por las deficiencias del sistema de transporte que posee la ciudad, generando mayores niveles de contaminación y aumentando los índices de accidentalidad y mortalidad en accidentes de tránsito.

Las empresas, sin importar cuál sea su razón social, deben hacer uso de las nuevas tecnologías para lograr llegar a más personas y, en el caso específico de las empresas de transporte público, hacer uso de las TIC con el fin de informar a los usuarios, en el menor tiempo posible, sobre cambios o situaciones anormales que hayan surgido durante el día. Al hacer esto los usuarios del transporte público estarán informados de los cambios del sistema y así podrán tomar las precauciones del caso o hacer uso de otra ruta.

Con el auge de las aplicaciones móviles, las personas pueden tener toda la información en la “palma de la mano”, es por ello que tiene fundamental importancia el hecho de que las empresas utilicen nuevas formas para informar a sus usuarios. Además, el uso de transporte público contribuye a bajar los niveles de contaminación por el hecho de que por una persona que haga uso de este servicio será, al menos, un vehículo menos rodando por las calles de la ciudad. Una forma eficiente y poco costosa para informar a los usuarios, y que además genera cero contaminaciones, es el uso de aplicaciones móviles.

Dada la importancia de informar, en todo momento, a los usuarios del transporte público y la enorme popularidad que tienen, en la actualidad, las aplicaciones móviles es necesario que el Sistema Estratégico de Transporte Público del municipio de Pasto (SETP) tenga una aplicación móvil que muestre información, en tiempo real, de rutas y paraderos.

Tema de investigación

Esta investigación comprende la construcción de un aplicativo móvil, el cual permite al usuario del Sistema Estratégico de Transporte Público de Pasto (SETP) la visualización, en tiempo real, de rutas y paraderos. Para permitir que el sistema persista en el tiempo y que este lo más actualizado posible, se creó un módulo administrativo para realizar operaciones CRUD (Create, Read, Update and Delete) sobre la base de datos del aplicativo.

Área de investigación

El desarrollo de la investigación involucra la apropiación y aplicación de conocimiento sobre Sistemas de Información Geográfica, Gestión de Bases de Datos, Funcionamiento y Control de Sistemas de Transporte Público.

Línea de investigación

La presente investigación corresponde a la Línea de Software y Manejo de Información, con enfoque al procesamiento y análisis de información almacenada en bases de datos.

Descripción del problema

La ciudad de Pasto, ha tenido un crecimiento demográfico importante en los últimos años, esto ha hecho que movilizarse dentro de la ciudad sea cada vez más difícil. Una buena opción para movilizarse en la ciudad es la utilización del transporte público.

El SETP cuenta con 23 rutas y 746 paraderos; las rutas están divididas en Estratégicas (7) y Complementarias (16) y los paraderos están asociados a las rutas dependiendo del recorrido seguido por una determinada ruta.

Hace algunos años la nomenclatura de las rutas del SETP de la ciudad de Pasto cambió y esto generó, entre los habitantes, un grado considerable de incertidumbre puesto que era complicado saber cual ruta llegaba hasta cierto punto de la ciudad, la empresa encargada de administrar el SETP tomó como medida la ubicación del nombre antiguo bajo el nuevo nombre pero aun así las personas continuaban en incertidumbre. Para conocer la ruta adecuada, para trasladarse a diferentes puntos de la ciudad, los usuarios optaron por detener el bus y preguntar al conductor que ruta los llevaba hacia el lugar al cual ellos querían ir, esto además de ser incómodo para las personas que estaban haciendo uso de ese bus, contribuía al entorpecimiento de la movilidad vehicular en la ciudad ya que el bus era detenido solo para dar información.

Los cambios en las rutas y paraderos están sujetos a las diferentes situaciones que se presenten en la ciudad como por ejemplo obras de adecuación de la infraestructura vial, desplazamiento de grupos sociales como movimientos estudiantiles, gremios del sector productivo, necesidades de medios de transporte en algunos sectores, etc. Esto hace que los cambios que se realicen sobre el SETP no se informen a tiempo a la ciudadanía generando estados de molestia sobre los usuarios del sistema.

Este proyecto de investigación pretende dar solución a esta problemática construyendo un App de código abierto, el cual hace uso de una ontología de

sitios de interés de la ciudad de pasto. Este aplicativo permitirá que los usuarios conozcan, en tiempo real, información sobre rutas y paraderos. Además, con la implementación de paraderos autorizados, por parte de AVANTE-SETP, se informará al usuario hacia donde debe dirigirse para tomar el bus que lo traslade hacia su punto de destino.

Formulación del problema

¿Cómo mostrar información actualizada y en tiempo real del Sistema Estratégico de Transporte Público de la ciudad de Pasto a los usuarios?

OBJETIVOS DE LA INVESTIGACIÓN

- **General**

Desarrollar un APP inteligente gratuita de código abierto para dispositivos móviles Android, del SETP de la ciudad de Pasto, que permita al usuario del transporte público, conocer información en tiempo real sobre rutas y paraderos de este sistema.

- **Específicos**

I. Investigar documentación bibliográfica acerca de ontologías, búsquedas semánticas, cartografía, georreferenciación de rutas de transporte público, además de mapas de las rutas de transporte urbano de la ciudad de Pasto y programación para dispositivos móviles en plataforma Android.
II. Diseñar una base de datos que almacene información de las rutas de transporte urbano, paraderos del sistema de transporte SETP y sitios de interés de la ciudad de Pasto.
III. Construir una base de datos que almacene información geocodificada del SETP y de sitios de interés de la ciudad de Pasto.
IV. Diseñar una ontología de sitios de interés de la ciudad de Pasto.
V. Construir una ontología de sitios de interés de la ciudad de Pasto.
VI. Diseñar SITApp, una aplicación móvil para dispositivos android, a fin de mostrar información a los usuarios sobre el SETP.
VII. Construir SITApp con un visor cartográfico, integrando la base de datos y la ontología sobre un Web Service.
VIII. Diseñar el Módulo Administrativo para SITApp que permita realizar operaciones CRUD sobre la base de datos espacial.
IX. Construir el módulo administrativo de SITApp.
X. Realizar pruebas de la aplicación SITApp.
XI. Realizar pruebas del módulo administrativo de SITApp.

Justificación

Con la ejecución de este proyecto se pretende resolver el problema de confusión y desconocimiento de los usuarios de transporte público urbano del municipio Pasto, acerca de las diferentes rutas y paraderos del SETP, al contar con una aplicación para dispositivos móviles, tales como celulares y tabletas, que facilite a los usuarios, la búsqueda y conocimiento de las rutas y paraderos más cercanos a su sitio de origen y a su sitio de destino en la ciudad de Pasto.

Esta App fue desarrollada con la ayuda de herramientas de software libre, lo cual permitió al Grupo de Investigación Aplicada en Sistemas GRIAS-KDD, la distribución gratuita de esta aplicación, generando bienestar social y mejorando la movilidad de los ciudadanos de la ciudad de Pasto.

Con la realización de este proyecto se formarán dos estudiantes como investigadores, adquiriendo conocimiento en temas asociados al proyecto, tales como georreferenciación, ontologías, búsquedas semánticas, metodologías ágiles de desarrollo de software, para el caso de este proyecto SCRUM, así como también en desarrollo de aplicaciones para dispositivos móviles.

Delimitación

El área geográfica que cubre el municipio de Pasto está conformada por 12 comunas (área urbana) y 17 corregimientos (área rural). Con el desarrollo de esta investigación se define el proceso de búsqueda de rutas del SETP de lugares que estén ubicados en el municipio de Pasto.

MARCO TEÓRICO

1. ANTECEDENTES

Para lograr una contextualización de la investigación se realizó una búsqueda de elementos que contribuyan a este aspecto. Durante este proceso de búsqueda se logró identificar algunos proyectos que hacen uso de cartografías para la representación de información georreferenciada, así.

A nivel internacional se encontraron los siguientes elementos.

Google transit[1], aplicación desarrollada por Google. Esta aplicación esta acoplada con los servicios de Google maps. Google maps brinda diferentes formas de trasladarse desde un punto a otro, por ejemplo en carro, a pie, en bicicleta y ahora para las ciudades de CALI, BOGOTÁ y MEDELLÍN, ofrece la posibilidad de dar indicaciones de como trasladarse en bus. Esta herramienta Brinda la opción de seleccionar punto inicio y fin, ya sea escribiendo el lugar, dirección o seleccionando los puntos sobre el mapa. La respuesta dada al usuario muestra información sobre la ruta que debe tomar, en el caso de tener que hacer trasbordo indica donde debe bajarse y que distancia debe caminar, además muestra tiempo estimado de duración del viaje, frecuencia con la que pasa el bus, etc. En el caso de que el servicio no inicie o llegue exactamente al lugar de inicio o destino, respectivamente, el sistema le indica al usuario que distancia debe desplazarse hasta llegar a la estación en la cual deberá tomar el bus o bajarse de él. Está disponible desde la versión 1.6 de Android. Este trabajo brindó a la investigación la idea para la creación de una de las funcionalidades de la aplicación, tocar dos puntos en el mapa, además se tuvo en cuenta otros detalles tales como la distancia que debe desplazarse el usuario para llegar al paradero en el cual debe esperar el bus y la distancia que debe caminar para llegar a su lugar de destino esto último aplica en el caso de que el bus no pase exactamente por el sitio de destino del usuario.

Moovit[2], es una empresa, de talla mundial, cuyo objetivo principal es trabajar por el transporte público. Está presente en alrededor de 67 países y en más de 1200 ciudades. Esta empresa tiene una comunidad a nivel mundial, quienes constantemente están haciendo actualizaciones. Usa como gestor de mapas a OpenStreetMaps. Para mostrar información sobre el transporte, la empresa, ha desarrollado un aplicativo móvil y una plataforma web, en los cuales el usuario, para iniciar a utilizar estos servicios, lo primero que debe hacer es seleccionar el país. Para el caso de Colombia, en la única ciudad en la cual está presente es Bogotá. Los Bogotanos pueden hacer uso de los servicios prestados por Moovit, ya sea descargando el App o entrando a su sitio web. En el caso del

App, el sistema hace uso del GPS y pregunta al usuario cual es el destino, internamente el App realiza el cruce de rutas y muestra al usuario información de que ruta debe tomar, si debe tomar un Alimentador para llegar a la estación de transmilenio más cercana, muestra una tabla completa sobre las diferentes acciones que debe realizar el usuario para llegar a su sito de destino como por ejemplo que distancia debe caminar, donde debe bajar para realizar un trasbordo, de ser necesario. Al igual que Google transit, en el caso de que no exista un servicio directo, Moovit indica al usuario que distancia debe caminar para tomar el bus y que distancia debe trasladarse para llegar a su sitio de destino.

La disponibilidad de esta aplicación depende del dispositivo, el sitio web es accesible por cualquier usuario con acceso a internet y un navegador instalado. Este proyecto, al igual que Google transit, contribuyo para la creación de una funcionalidad en SITApp, búsquedas por sitios de interés. Al igual que moovit, SITApp hace uso del GPS del dispositivo móvil y usa las coordenadas del télefono como punto inicial del viaje. Además SITApp también hace uso de OpenStreetMaps para mostrar los resultados espaciales de las consultas realizadas por los usuarios.

A nivel nacional, se encontró la siguiente herramienta:

TRANSMISIT, Es la aplicación más popular en la ciudad de Bogotá y con la cual MOVILIXA[3] entro en el negocio del desarrollo de aplicaciones para el transporte.

Las funcionalidades que ofrece esta aplicación son:

Cálculo de ruta, El sistema busca la ruta óptima entre dos puntos de la ciudad, en el caso de no existir un servicio directo, la aplicación brinda opciones al usuario como por ejemplo hacer trasbordo o desplazarse hasta el lugar donde debe tomar la ruta adecuada.

Mapas, Los usuarios pueden conocer el sistema y visualizar las rutas que este tiene en un mapa de google maps. Para resaltar, el sistema guarda una imagen del mapa y de esta manera los usuarios a pesar de no contar con acceso a internet podrán hacer uso de los mapas en la aplicación.

Detalles de recorrido, rutas favoritas, búsquedas, horarios, uso del GPS y datos en tiempo real para el usuario final.

Tarjetas del sistema, puntos de compra de tarjetas de transporte, lectura de NFC (solo para celulares que tengan la tecnología).

Puntos de parada, ubicación por GPS, búsquedas y datos en tiempo real de llegadas de los servicios.

Noticias y más, el App esta enlazada con las redes oficiales de las empresas de transporte.

La mayoría de las opciones funcionan sin tener acceso a Internet o un plan de datos.

Está disponible desde la versión 4.0.3 de Android.

Este trabajo generó la idea para la creación de una nueva funcionalidad en la aplicación, recorridos de rutas. SITApp muestra el recorrido que realiza una determinada ruta del SETP. Permite que los usuarios conozcan las diferentes zonas de la ciudad por las cuales pasa el bus. Además este proyecto dio un acercamiento a cuál sería la ruta óptima para el usuario. Para el caso de SITApp se tomó como criterio el hecho de que el usuario deba desplazarse, a pie, la menor distancia posible para llegar a su sitio de destino desde su ubicación en la ciudad.

2. FUNDAMENTOS TEÓRICOS

2.1. SISTEMAS DE INFORMACIÓN GEOGRÁFICA

Un Sistema de Información Geográfica (SIG) puede ser definido como un sistema de hardware, software y procedimientos diseñados para soportar la captura, manejo, manipulación, análisis, modelado y visualización de datos espacialmente referenciados para resolver problemas complejos de planificación y administración[4]**.**

Los SIG están diseñados para datos relacionados con características del mundo real o fenómenos descritos en términos de ubicación, también conocidos como datos geográficos. Estos datos deben seguir varios criterios: deben estar conectados con un sistema de coordenadas geográficas aceptado de la superficie de la Tierra, deben estar representados en una escala geográfica, y deben describir las interrelaciones espaciales entre sí que describen cómo se unen [5].

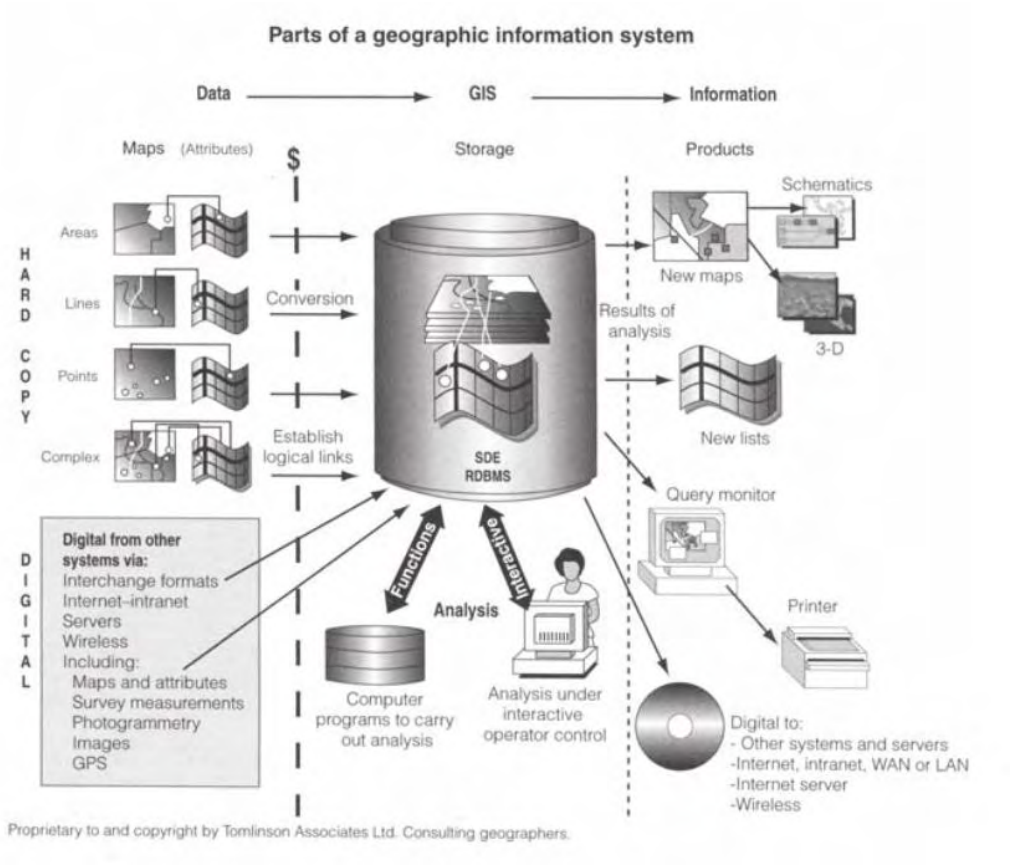
Según[6], los SIG están compuestos por 3 subsistemas, ver Figura 1.

Subsistemas de un SIG.

Es común hablar de tres subsistemas, en la Figura 1 se puede observar los subsistemas de un SIG.

- **Subsistema de datos**. Se encarga de las operaciones de entrada y salida de datos, y la gestión de estos dentro del SIG. Permite a los otros subsistemas tener acceso a los datos y realizar sus funciones en base a ellos.
- Subsistema de análisis. Contiene métodos y procesos para el análisis de los datos geográficos.
- **Subsistema de visualización y creación cartográfica**. Crea representaciones a partir de los datos (mapas, leyendas, etc.), permitiendo así la interacción con ellos. Entre otras, incorpora también las funcionalidades de edición[6].

Figura 1. Subsistemas de un SIG



Fuente: [7].

Sistema de coordenadas

Es una creación artificial que permite la definición de una manera analítica de la posición de un objeto o un fenómeno.

Existen diferentes maneras de definir analíticamente la ubicación de un punto o un fenómeno sobre la superficie de la tierra. Matemáticamente todos los

sistemas de coordenadas son admisibles y el único motivo para seleccionar uno u otro es la conveniencia.

Desde el punto de vista práctico se escogen los sistemas de coordenadas, que permitan representar la cuestión objeto de estudio de una manera física y geométricamente interpretable y susceptible de ser medida[8].

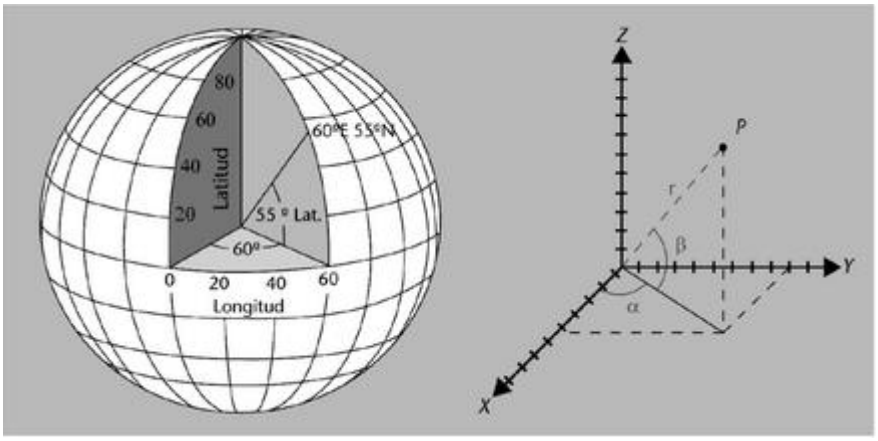
Sistemas de referencia espacial

La ubicación de puntos, objetos o fenómenos sobre la superficie terrestre obedecen a sistemas de referencia espacial, mediante los cuales se puede medir exactamente la ubicación de cualquiera de los elementos anteriores sobre la superficie de la tierra. Dentro de estos sistemas se encuentran el sistema de coordenadas geográficas o geodésicas y el sistema de proyecciones cartográficas.

Sistema de coordenadas geográficas o geodésicas

Un sistema de coordenadas geográficas o geodésicas utiliza una superficie esférica tridimensional para definir las localizaciones sobre la superficie terrestre. En este sistema cualquier punto sobre la Tierra puede representarse por dos ángulos, medidos desde el centro de la tierra denominados LATITUD y LONGITUD. La latitud de un punto es el ángulo medido desde el centro de la tierra hacia el norte, entre el ecuador y la posición de un punto sobre la superficie terrestre. La longitud de un punto es el ángulo medido a lo largo del ecuador desde cualquier punto de la tierra [8], ver Figura 2.

Figura 2. Coordenadas geodésicas

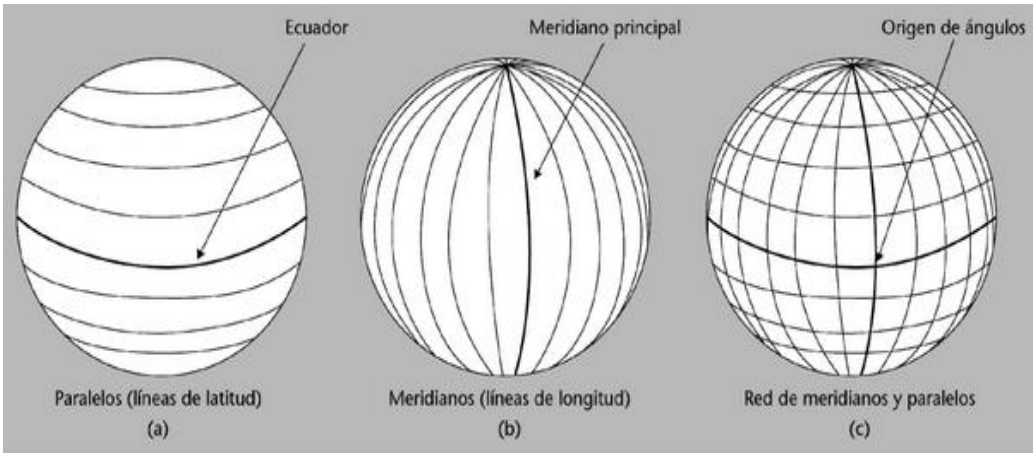


Fuente: [8]

Las líneas formadas a partir de las longitudes son denominadas meridianos, el meridiano principal o meridiano de Greenwich divide la superficie terrestre en dos hemisferios, oriental y occidental. Las líneas formadas por las latitudes se

denominan paralelos, el paralelo principal o Ecuador divide la superficie terrestre en dos hemisferios, norte y sur [8]. El punto con coordenadas (0,0) u origen está ubicado justo en el la intersección entre el meridiano y el paralelo principal. Ver Figura 3.

Figura 3. Elementos de los sistemas de coordenadas geográfcas: Paralelos (a), Meridianos (b) Origen de coordenadas (c)

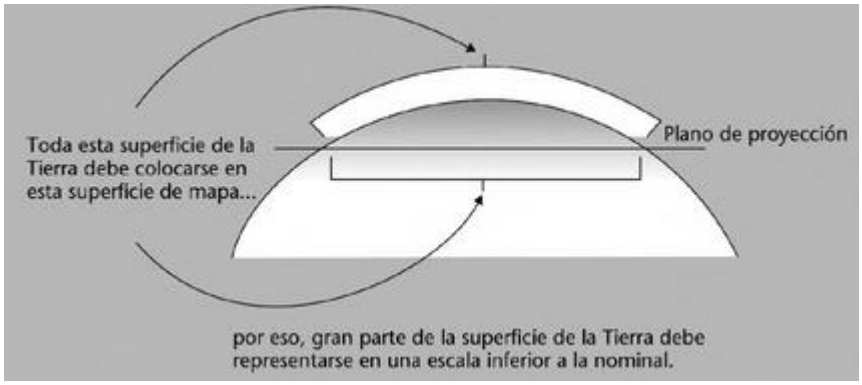


Fuente: [8].

Sistema de proyecciones cartográficas

Son transformaciones matemáticas que permiten representar (proyectar) a la esfera en el plano, y convertir las coordenadas geográficas (latitud & longitud) en coordenadas cartesianas (x & y). Este proceso conlleva distorsiones de la superficie original en 3-dimensiones, al convertirse a una superficie plana de dos dimensiones [9]. Ver Figura 4.

Figura 4. Deformación de los atributos de una superficie tridimensional al adaptarse a una bidimensional.



Fuente [8].

Tipos de proyecciones

Al momento de representar la esfera en el plano se presentan deformaciones. En la Figura 5, se puede observar los diferentes tipos de deformaciones.

Figura 5. Proyecciones cartográficas.

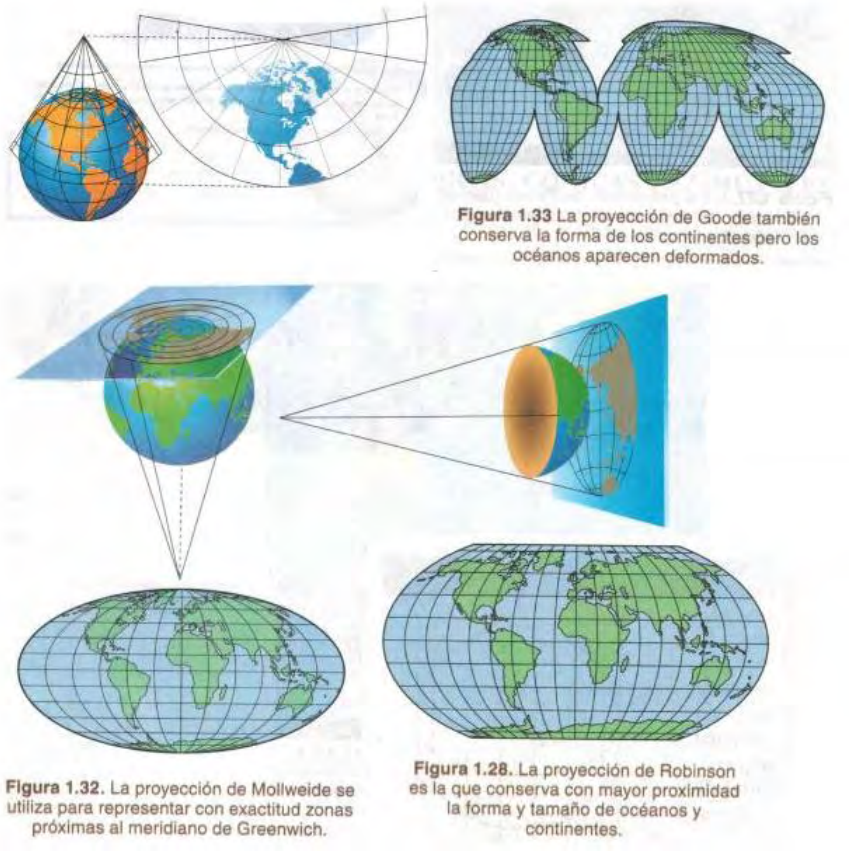


Figura 1.33 La proyección de Goode también conserva la forma de los continentes pero los océanos aparecen deformados.

Figura 1.32. La proyección de Mollweide se utiliza para representar con exactitud zonas próximas al meridiano de Greenwich.

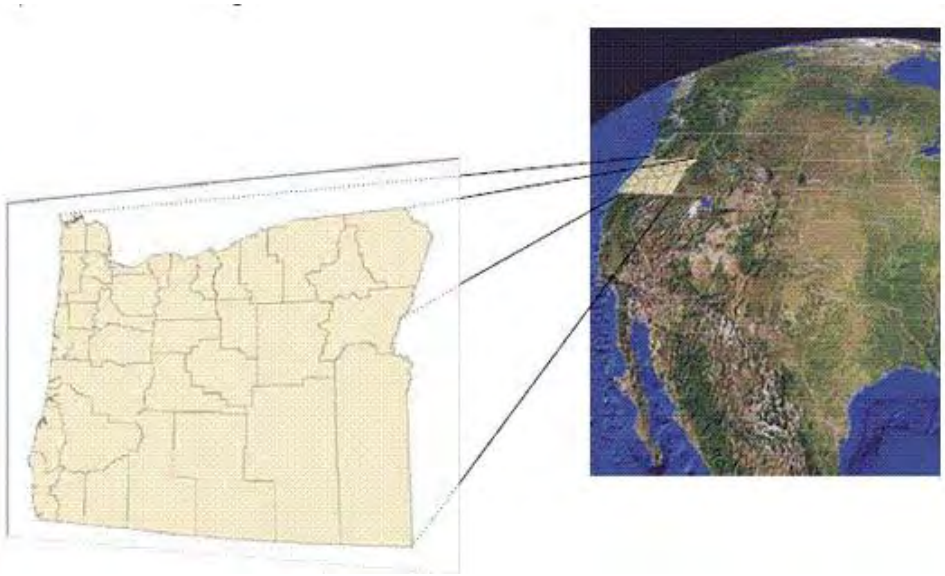
Figura 1.28. La proyección de Robinson es la que conserva con mayor proximidad la forma y tamaño de océanos y continentes.

Fuente: [10]

Georreferenciación

[8] Define georreferenciación como el proceso que se utiliza para relacionar la posición de un objeto o superficie en un plano con su posición sobre la superficie terrestre. De este modo, se podría afirmar que la georreferenciación consiste en ubicar el objeto sobre el mapa y asignarle una ubicación espacial con el uso de coordenadas geográficas para luego poder identificar este objeto haciendo uso de un GIS. En la Figura 6, se puede apreciar un ejemplo de este proceso.

Figura 6. Georreferenciación.



Fuente: [11]

Geolocalización

Según [12] Geolocalización es una herramienta que permite obtener la ubicación geográfica real de cualquier tipo de objeto o persona, esto se realiza a través de, por ejemplo, un dispositivo móvil o un computador conectado a internet, el término geolocalización está intrínsecamente ligado al uso de sistemas de posicionamiento, teniendo mayor énfasis en una determinada posición significativa. La geolocalización se emplea para ubicar un dispositivo electrónico, el cual debe estar conectado a Internet, en tiempo real.

Sistema de posicionamiento Global, GPS

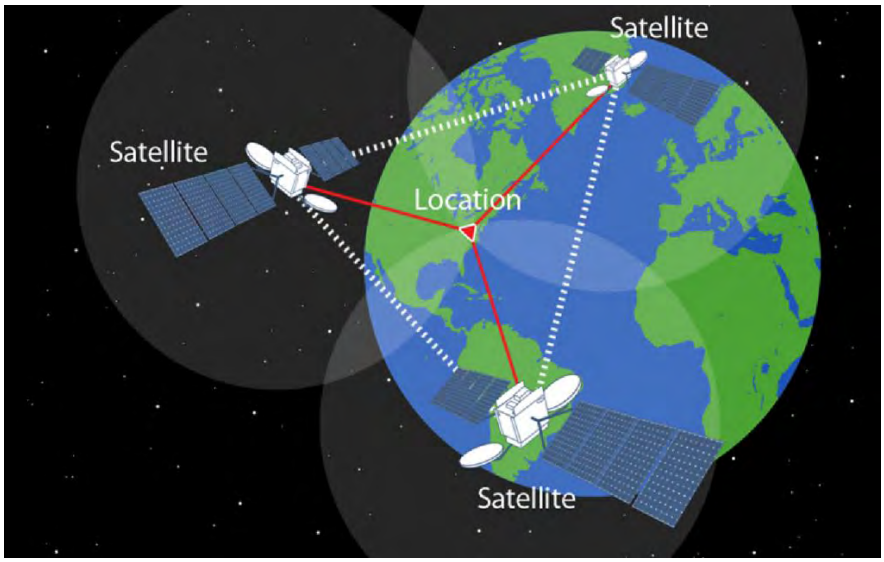
El Sistema de Posicionamiento Global o GPS, aunque su nombre correcto es NAVSTAR-GPS1, es un sistema global de navegación por satélite que permite determinar en todo el mundo la posición de un objeto, una persona, un vehículo o una nave. Se puede alcanzar una precisión hasta de centímetros, usando el GPS diferencial, pero lo habitual son unos pocos metros.

Aunque su invención se atribuye a los gobiernos de Francia y Bélgica, el sistema fue desarrollado e instalado por el Departamento de Defensa de los Estados Unidos, del que actualmente se encarga.

El GPS funciona mediante una red de 27 satélites (24 operativos y 3 de respaldo) en órbita a 20.200 km sobre el globo terráqueo, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar una posición, el receptor que se utiliza para ello localiza

automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la posición y el reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del Sistema de Posicionamiento y calcula el retraso de las señales; es decir, la distancia al satélite. Por “triangulación” los tres satélites calculan la posición en que el GPS se encuentra, ver Figura 7. La triangulación en el caso del Sistema de Posicionamiento Global se basa en determinar la distancia de cada satélite respecto al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición [13].

Figura 7. Triangulación GPS.



Fuente: [14]

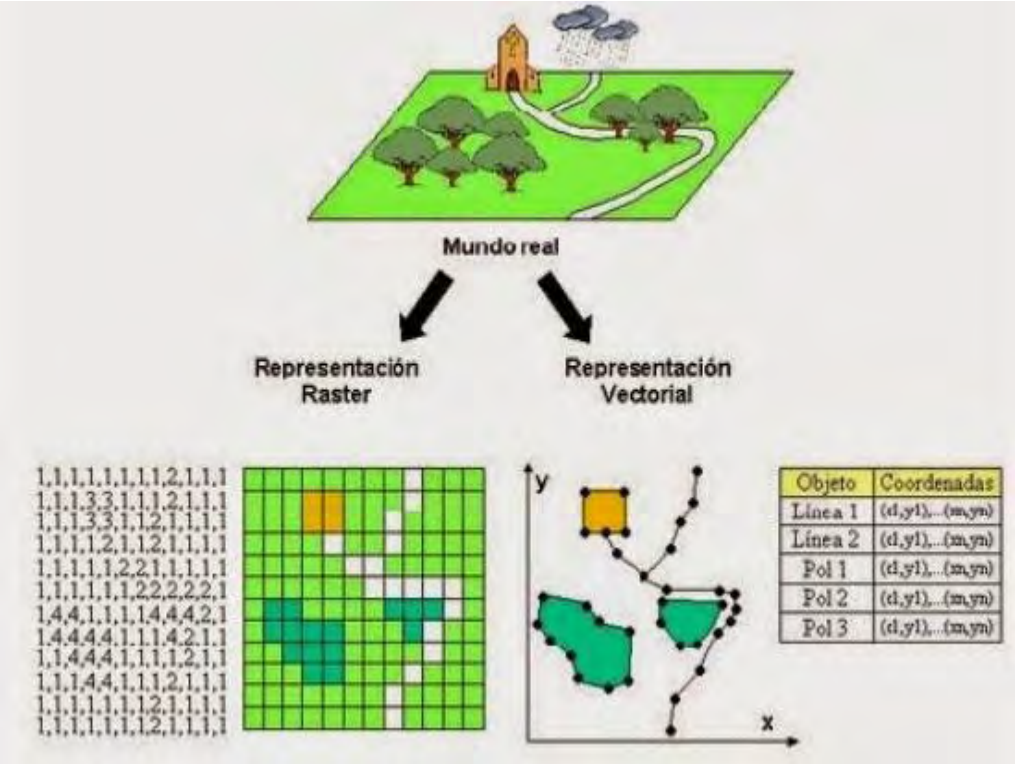
Bases de datos espaciales

En [15] se le llama base de datos a los bancos de información que contienen datos relativos a diversas temáticas y categorizados de distinta manera, pero que comparten entre sí algún tipo de vínculo o relación que busca ordenarlos y clasificarlos en conjunto. En el caso de los SIG, las bases de datos utilizadas son las llamadas bases de datos espaciales, estas según [16] pueden definirse como conjuntos de objetos en el espacio. Cada objeto tiene identidad, propiedades, localización y se relaciona con otros objetos del espacio. Con Objetos se hace referencia a distintas entidades ubicadas en el espacio, cada una con su propia descripción geométrica. De esta forma, se pueden modelar ciudades, ríos, bosques, etc. [16]**.**

Una base de datos espacial debe permitir la descripción de los objetos espaciales mediante tres características básicas. La primera es describir sus Atributos, esto es saber qué es un objeto mediante la representación de sus características. La segunda es describir su Localización, esto es saber dónde está el objeto y qué espacio ocupa. La tercera corresponde a la descripción de su Topología, esto es mejorar la interpretación semántica del contexto y establecer jerarquías de elementos a través de sus relaciones [16].

Las bases de datos espaciales puede almacenar dos tipos de datos, imágenes, generalmente satelitales, o datos correspondientes a formas geométricas (puntos, líneas y polígonos). Para lograr almacenar esta información sobre bases de datos espaciales los SIG hacen uso de dos modelos de representación geográficos, el modelo raster y el modelo vectorial, en la Figura 8 se puede ver las dos formas de representar un espacio del mundo real.

Figura 8. Formas de representar la realidad bajo los dos modelos de representación de información geográfica.



Fuente: [17]

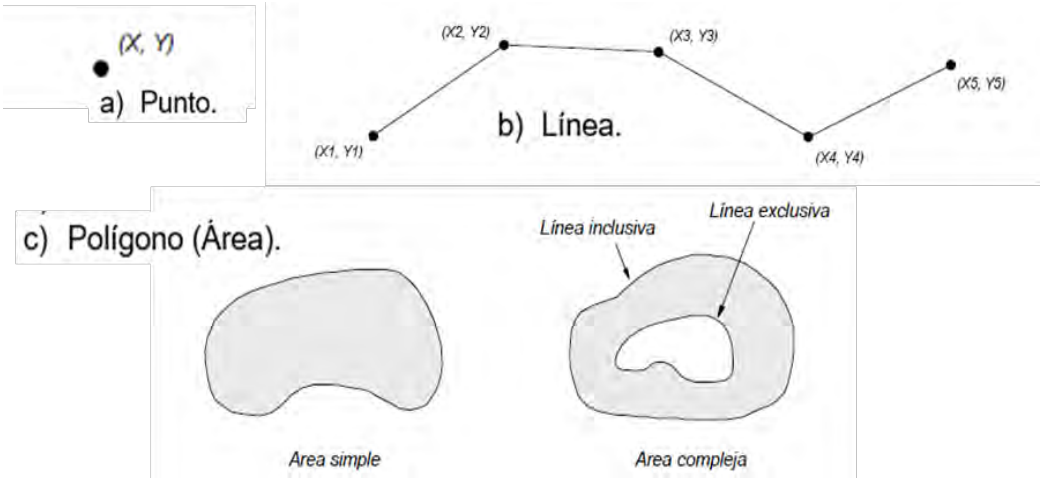
Modelo de datos vectorial

Son aquellos Sistemas de Información Geográfica que utilizan vectores definidos por pares de coordenadas relativas a algún sistema cartográfico para la descripción de los objetos geográficos. Este modelo usa el sistema de

coordenadas (X, Y) y emplea representaciones geométricas de punto, línea y polígono [18], en la Figura 9 se muestra la representación de estos elementos.

- Punto: es la representación geométrica constituida sólo por un par de coordenadas (X, Y) [19]. Un punto puede representar pozos, postes de energía, lugares de interés, paraderos de buses, entre otros.
- Línea: es la representación geométrica constituida por una serie de dos o más pares distintos de coordenadas (vértices) ligados secuencialmente [19]. Una línea puede representar, caminos, ríos, líneas de alta tensión, túneles, entre otros.
- Polígono (Área): es la representación geométrica delimitada por una línea cerrada o serie de líneas que cierran [19].

Figura 9. Representación de elementos en el modelo de datos vectorial.

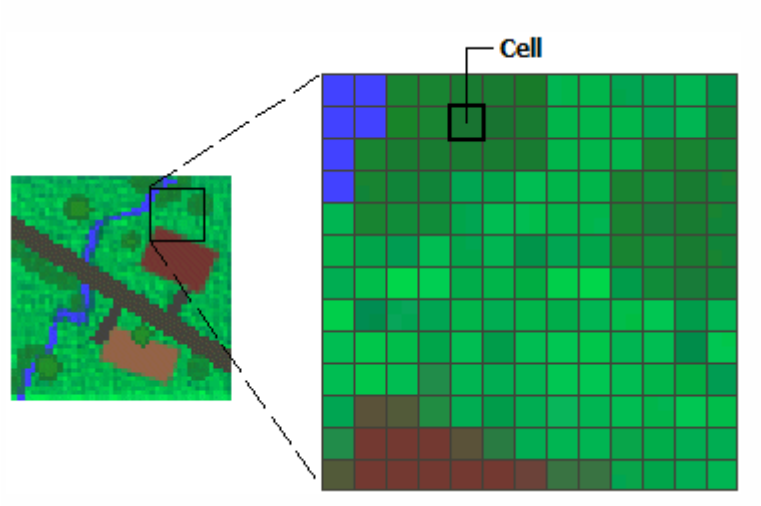


Fuente: [19]

Modelo de datos raster

En su forma más simple, un raster consta de una matriz de celdas (o píxeles) organizadas en filas y columnas (o una cuadrícula) en la que cada celda contiene un valor que representa información, como la temperatura. Los rasteres son fotografías aéreas digitales, imágenes de satélite, imágenes digitales o incluso mapas escaneados [20], En la Figura 10, se puede ver la representación del mundo real bajo este modelo.

Figura 10. Representación del mundo real bajo el modelo de datos raster.



Fuente: [20]

Geocodificación

La Geocodificación es el proceso de convertir lugares informales, tales como direcciones de calles, en las coordenadas geográficas formales, ver figura 11. Ubicaciones se les da un conjunto único de coordenadas para marcar su lugar exacto en la tierra. La tecnología GPS que muchos han llegado a depender es posible gracias a la codificación geográfica y la creación de bases de datos de información geográfica [21].

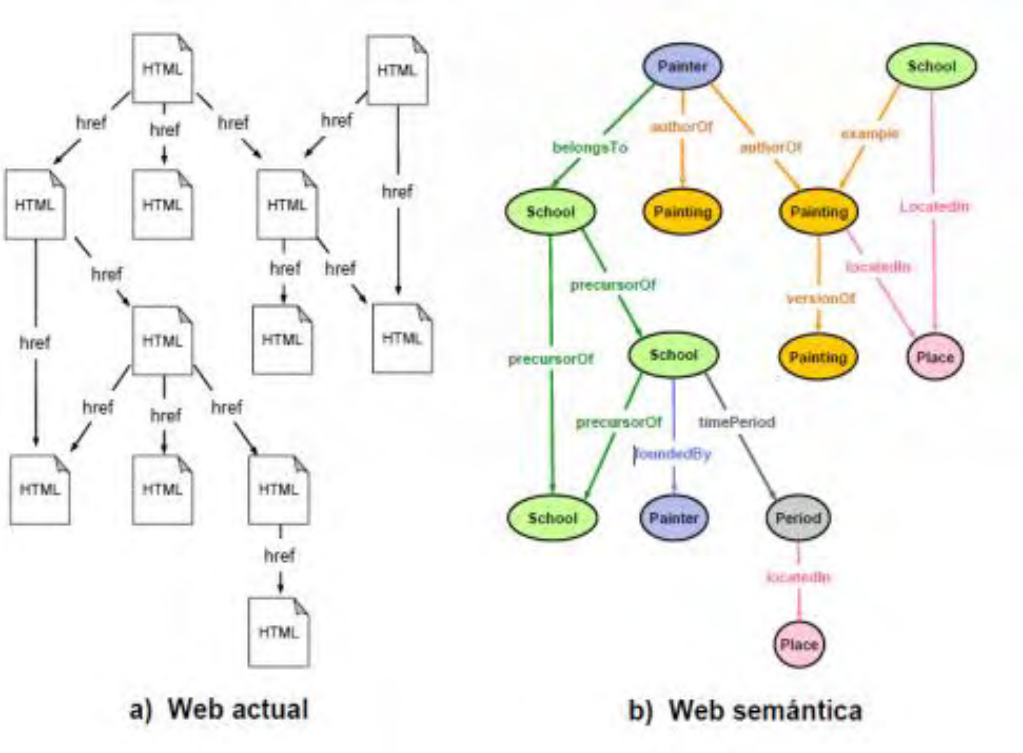
Figura 11. Proceso de geocodificación

2.2. WEB SEMÁNTICA

Siendo una extensión de la web actual, se propone superar las limitaciones que actualmente se tiene en los contenidos web ya que estos para las máquinas no poseen significado, tienen una estructura de metadatos que por sí solos no dicen mucho y se encuentran organizados de manera caótica para este propósito la web semántica plantea clasificar, estructurar y darle semántica a los contenidos para que estos sean procesables por las máquinas. La idea es que la web semántica esté formada (al menos en parte) por una red de nodos tipificados e interconectados mediante clases y relaciones definidas por una ontología compartida por sus distintos autores[22].

En la Figura 12, se observa una comparación de la web actual y la web semántica.

Figura 12. Web actual vs. Web semántica



Fuente: [22]

Redes semánticas: una red semántica es una forma de representar el conocimiento lingüístico en la que los conceptos y sus interrelaciones se representan mediante un gráfico. Se utilizan, entre otras cosas, para representar mapas conceptuales y mentales. En una red semántica se representan esos elementos semánticos mediante nodos. Cuando hay una relación semántica entre dos elementos semánticos se representará en la

red semántica como una línea, flecha o enlace entre ambos[22]. Estas redes pueden ser representadas mediante redes IS-A, grafos conceptuales o redes de marcos.

Búsqueda semántica: la búsqueda semántica pretende comprender las expresiones proporcionadas por el usuario en su consulta. Ahora se trata de devolverle al usuario unos resultados orientados a su búsqueda, más específicos, sin necesidad de una gran intervención por parte del usuario haciendo uso del contexto y por ello, del significado[23].

Desde un punto de vista técnico, un buscador semántico es una aplicación que comprende las búsquedas de los usuarios y los textos de los documentos de la web mediante el uso de algoritmos que simulan comprensión o entendimiento, y que a partir de éstos proporciona resultados correctos sin que el usuario tenga que abrir el documento e inspeccionarlo por sí mismo[24].

Ontología: las ontologías se definen como conceptualizaciones que determinan el significado de un grupo de conceptos para un determinado dominio. Esta conceptualización debe ser representada de manera formal, legible y utilizable por los ordenadores [25].

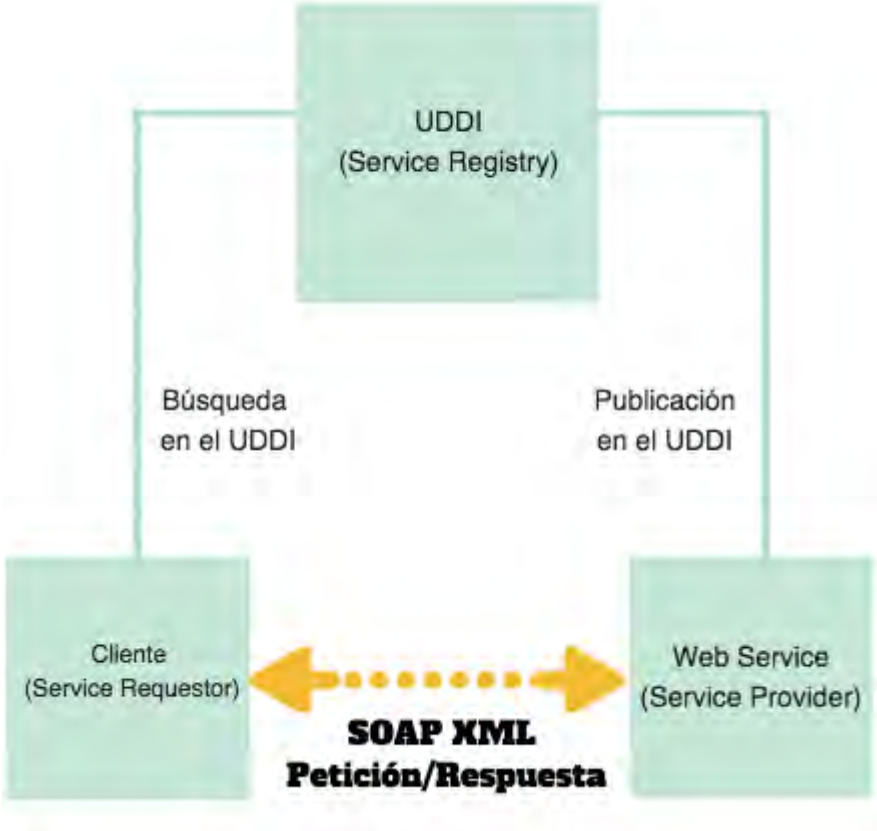
2.3. WEB SERVICES

Un Web Service, o Servicio Web, es un método de comunicación entre dos aparatos electrónicos en una red (ver figura 13). Es una colección de protocolos abiertos y estándares usados para intercambiar datos entre aplicaciones o sistemas. Las aplicaciones escritas en varios lenguajes de programación que funcionan en plataformas diferentes pueden utilizar web services para intercambiar información a través de una red. La inter-operatividad, por ejemplo entre Java y Python o Windows y Linux se debe al uso de estándares abiertos.

Como sistema de mensajes se utiliza XML estandarizado. El protocolo más simple para el intercambio de información entre ordenadores es XML-RPC, que emplea XML para llevar a cabo RPCs. RPC, Remote Procedure Call, es un protocolo de red que permite a un programa ejecutar código en una máquina remota. Los XML-RPC requests son una combinación entre contenido XML y headers HTTP. La simpleza de los XML-RPC hizo que el estándar evolucionase a SOAP, uno de los componentes básicos de los Web Services.

SOAP es un protocolo escrito en XML para el intercambio de información entre aplicaciones. Es un formato para enviar mensajes, diseñado especialmente para servir de comunicación en Internet, pudiendo extender los HTTP headers. Es una forma de definir qué información se envía y cómo mediante XML. Básicamente es un protocolo para acceder a un Web Service [26].

Figura 13. Arquitectura Web Services.



Fuente: [26]

2.4. APLICACIONES MÓVILES

Una aplicación móvil no es más que software, es decir, es un software desarrollado para ser ejecutado sobre un equipo móvil como por ejemplo un teléfono o una Tablet (ver figura 14), o gracias al avance de la tecnología móvil, un Smart TV o incluso en un Smart-watch. Pero en síntesis sigue siendo software. Las aplicaciones móviles son a los celulares como el software a los computadores.

El desarrollo de las aplicaciones tiene en cuenta las limitaciones de los dispositivos, como la batería o el software y comienzan probándose utilizando un emulador, para después ponerse al mercado en versión de prueba.

Estas aplicaciones pueden ser gratuitas o de pago, donde el 20-30% del coste de la aplicación se destina al distribuidor y el resto es para el que la desarrolla. Pueden adquirirse a través de diferentes plataformas según el sistema operativo del aparato, como Google Play (antes Android Market), App Store, Windows Phone Store y BlackBerry World, entre otros, como puede ser Amazon Appstore, sólo disponible para Android [27].

Figura 14. Apps móviles

Fuente: [27]

2.5. SETP PASTO.

En la ciudad de San Juan de Pasto, la ciudad Sorpresa de Colombia y capital del departamento de Nariño, se está desarrollando un proyecto el cual tiene como objetivo principal mejorar la movilidad en la ciudad haciendo uso de un sistema de transporte público más eficiente que el que tenía hace un par de años.

En la ciudad de Pasto, el servicio de transporte público era prestado por cuatro empresas (TRANSPORTE EJECUTIVOS S.A. TESA, COOPERATIVA AMERICANA DE TRANSPORTADORES LTDA COAMETRAN, COOPERATIVA DE TRANSPORTADORES URBANOS CIUDAD DE PASTO LTDA. COOTRANUR y EMPRESA TRANSPORTADORA AUTOBUSES DEL SUR LTDA) y cada una de ellas era administrada por una entidad diferente, esta situación daba paso para que el servicio prestado a la comunidad Pastusa sea de mala calidad puesto que se presentaban situaciones tales como recorridos demasiado largos, demasiada oferta de rutas para un mismo destino o por el contrario poca oferta para sitios que requerían mayor cantidad de vehículos prestando el servicio; otra situación que se presentaba era el tiempo de servicio, en ocasiones los usuarios debían esperar demasiado tiempo o sucedía que los buses realizaban el recorrido vacíos puesto que el anterior ya había recogido a los pasajeros.

Las cuatro empresas, en el año 2001 haciendo uso del decreto 170 del Ministerio de Transporte, mediante el cual se permite que las empresas puedan efectuar alianzas temporales, decidieron realizar una unión temporal llamada UNIÓN TEMPORAL CIUDAD SORPRESA con el fin de lograr unificar la prestación del servicio de transporte público de pasajeros en aras de lograr un mejor servicio hacia los pastusos.

Para seleccionar el tipo de sistema de transporte público que debía tener la ciudad de Pasto se hizo uso del documento CONPES 3546, según el cual “El Plan Nacional de Desarrollo, 2006 – 2010 “Estado comunitario: desarrollo para

todos", estableció en el programa de Ciudades Amables, la estrategia para desarrollar diferentes acciones que permitirían estructurar movilidades eficientes. Para ello, la Nación ha cofinanciado los Sistemas Integrados de Transporte Masivo, para las ciudades de más de 500.000 habitantes. En ese mismo sentido el Gobierno Nacional apoya el desarrollo de sistemas de transporte para las ciudades que tienen una población entre 500.000 y 250.000 habitantes, mediante la implantación de Sistemas Estratégicos de Transporte Público – SETP." [CONPES 3549].

Pasto tiene una población aproximada de 480.000 habitantes, esto hizo que el sistema de transporte público para esta ciudad sea el SETP. Para el caso de la ciudad de Pasto el SETP tiene los siguientes elementos.

- Dos topologías de rutas.
- Once topologías de paraderos.

Topología de rutas.

El SETP de Pasto cuenta con dos tipos de rutas: Complementarias y Estratégicas, para realizar los recorridos en la ciudad. Según la sección de Operaciones de AVANTE, Las rutas Complementarias son más cortas y rápidas que las rutas estratégicas, esta situación, por ahora, no es claramente visible puesto que el sistema está constantemente cambiando y, en este momento, algunas de las rutas complementarias son más largas que las Estratégicas y por ende el tiempo de recorrido es más largo. Se espera que al finalizar la implementación y puesta en marcha de todo el SETP de la ciudad de Pasto, los recorridos sean mucho más eficientes y el servicio prestado a la comunidad pastusa sea de excelente calidad.

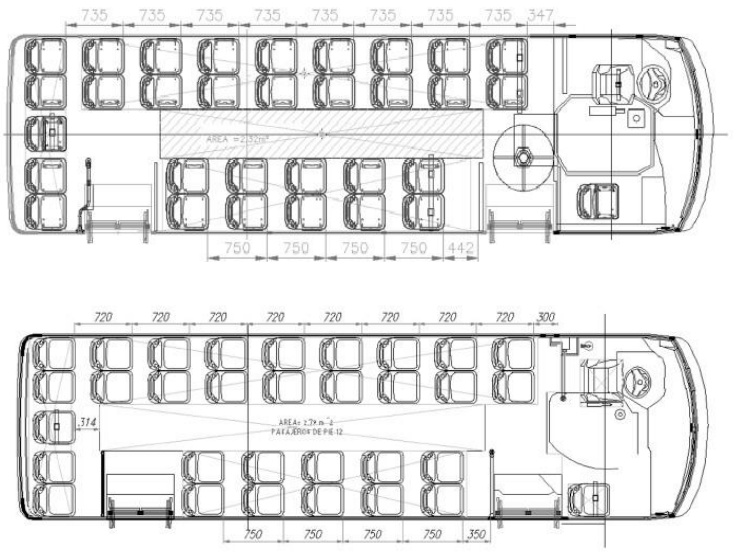
Para realizar los recorridos, la empresa hace uso de dos topologías de buses, ver figuras 15 y 16.

Topología 1:

- Longitud: 9 metros +/- 10%.
- Número de asientos: 30 a 32 asientos para pasajeros.
- Asientos Preferenciales: 2
- Capacidad Total: 40 pasajeros
- Número de Puertas de Servicio: 2
- Norma aplicable: NTC 5206
- Dos puertas de ancho libre mínimo de 650 mm, cada una.

A continuación, se presenta un esquema de distribución genérico del vehículo recomendado, tomando como base una longitud total de 9 metros:

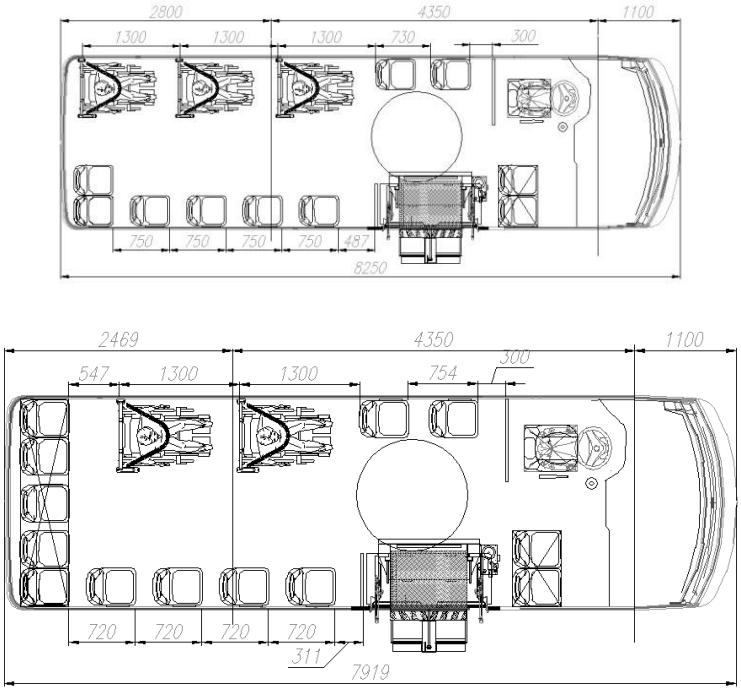
Figura 15. Topología de buses número 1.



Fuente: [28].

Topología 2:

Figura 16. Topología de buses número 2.



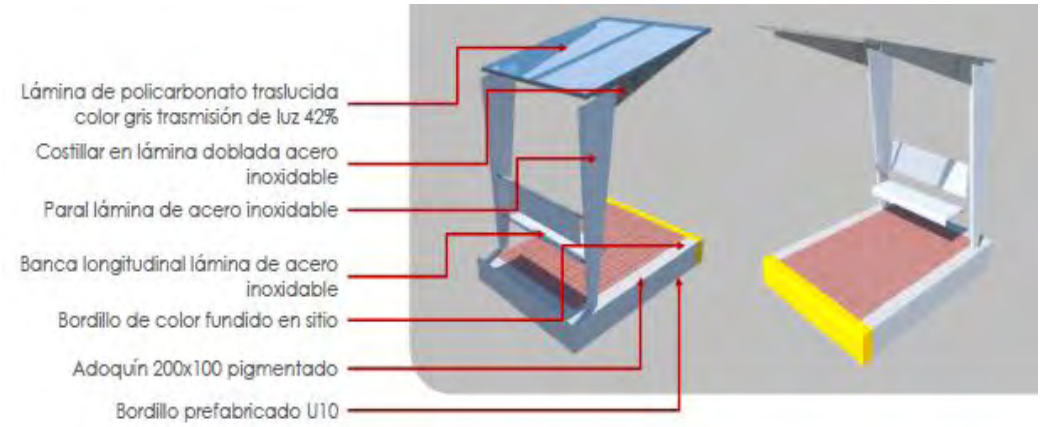
Fuente: [28].

El ancho de las puertas deberá ajustarse a los requerimientos de las normas colombianas. Para el vehículo accesible se debe manejar una puerta doble con 1100 mm de ancho libre.

El elevador debe ser al menos semiautomático, de fácil operación. [Sección Operativa AVANTE]

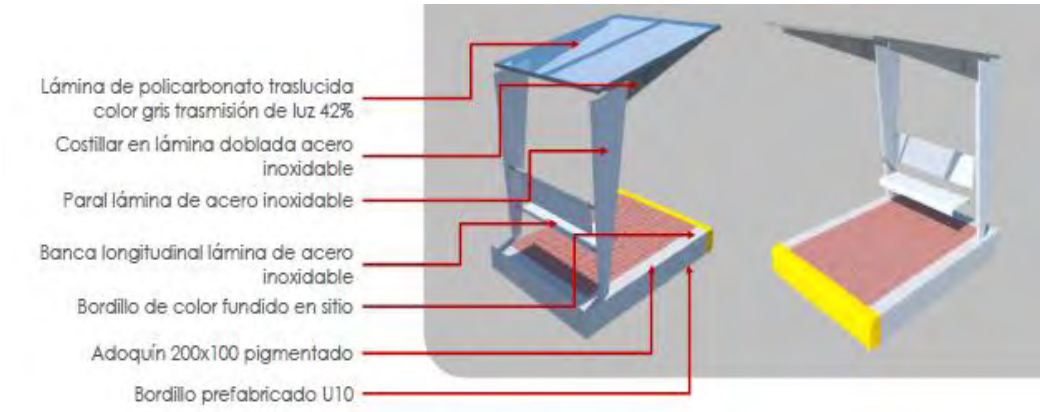
Existen 11 tipos diferentes de paraderos pero estos son el resultado de una combinación de módulos así que para lograr una mayor comprensión de los tipos de paraderos que tiene el SETP de Pasto, a continuación, se describe, a grandes rasgos, cada uno de los módulos, ver figuras 17 a 22 para conocer la descripción.

Figura 17. Módulo 1 (M1)



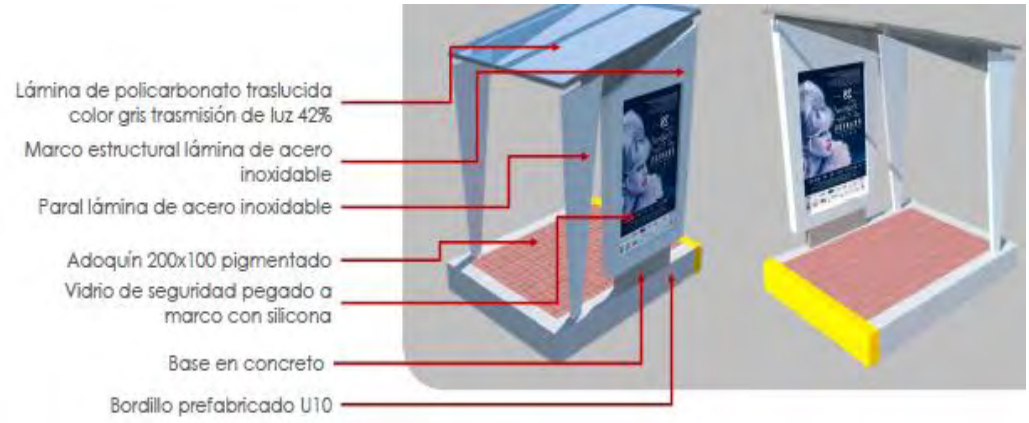
Fuente: [28].

Figura 18. Módulo 2 (M2)



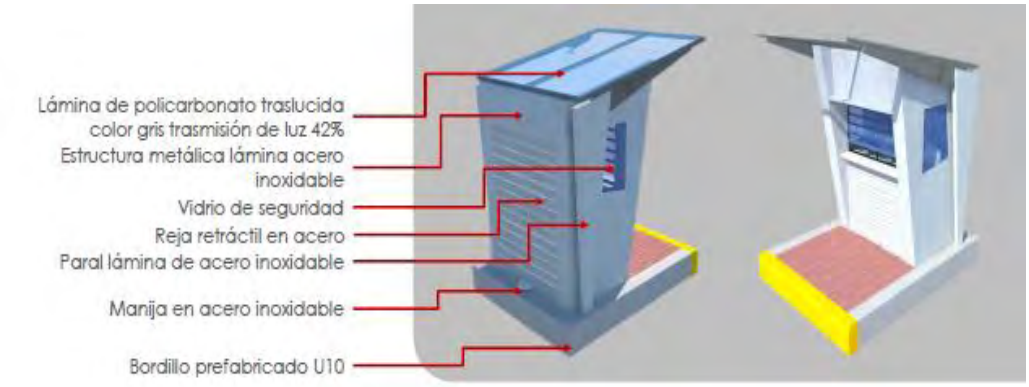
Fuente: [28].

Figura 19. Módulo 3 (M3)



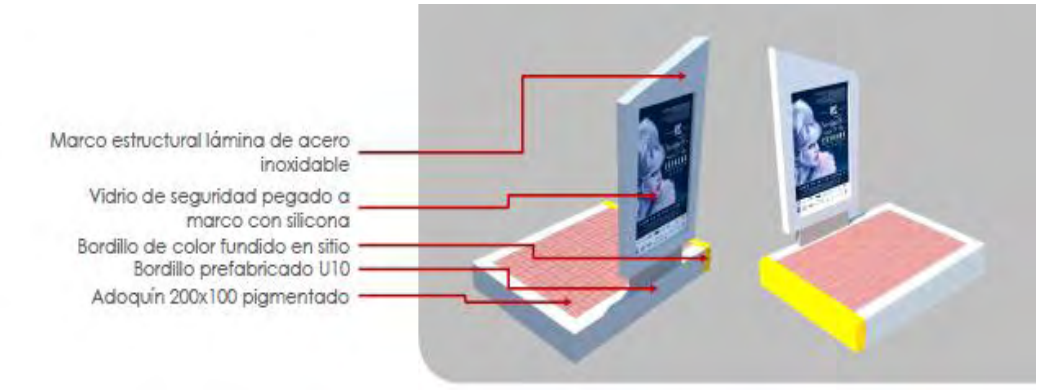
Fuente: [28].

Figura 20. Módulo 4 (M4)



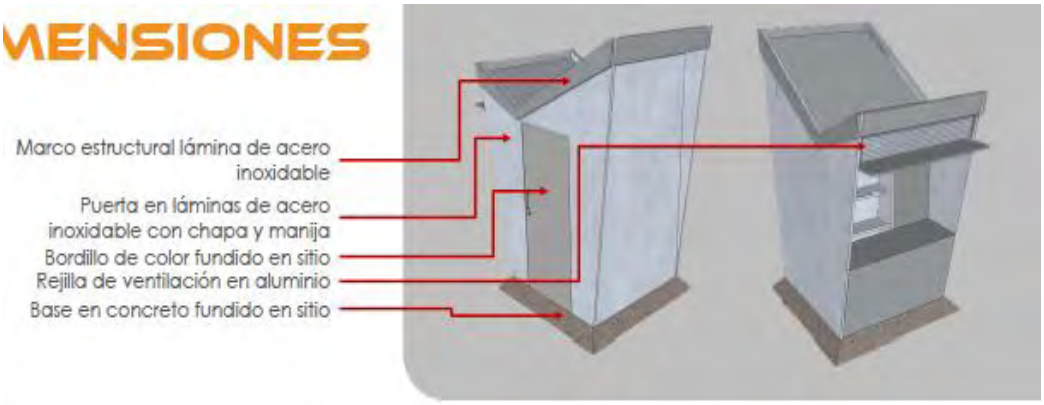
Fuente: [28].

Figura 21. Módulo 5 (M5)



Fuente: [28].

Figura 22. Módulo 6 (M6)



Fuente: [28].

Ahora bien, los paraderos resultan de las combinaciones de los módulos anteriores, por lo tanto.

- Paradero P1A compuesto por módulos (M1 + M1)
- Paradero P1B compuesto por módulos (M2 + M3)
- Paradero P2A compuesto por módulos (M1 + M2 + M3)
- Paradero P2B compuesto por módulos (M2 + M2 + M3)
- Paradero P2C compuesto por módulos (M2 + M1 + M3)
- Paradero P3A compuesto por módulos (M2 + M1 + M4 + M5)
- Paradero P3B compuesto por módulos (M2 + M2 + M4 + M5)
- Paradero P3C compuesto por módulos (M1+ M1 + M4 + M5)
- Paradero 2+3 compuesto por módulos (M1 + M5)
- Paradero S compuesto por módulos (M2 + M4)
- Paradero NA, no aplica, es decir lo único que se ubica en el sitio del paradero es un tótem.

1. METODOLOGÍA

Para lograr los objetivos de este proyecto de investigación se realizó una adaptación de la metodología de desarrollo ágil SCRUM, dado que esta metodología de desarrollo está encaminada a proyectos de construcción de software con equipos de trabajo numerosos se decidió tomar los mejores elementos y aplicarlos a cada una de las 6 fases que componen la ejecución de esta investigación, ver figura 23. A continuación, se tiene una breve descripción de cada una de ellas:

- Recolección de información geográfica.
- Diseño y construcción de la base de datos.
- Diseño y construcción de la ontología.
- Diseño y construcción de SITApp.
- Diseño y construcción del módulo administrativo de SITApp.
- Realización y evaluación de pruebas.

Recolección de información geográfica

El objetivo de esta fase fue obtener la materia prima, en cuanto a geometrías de rutas y la ubicación de los paraderos del Sistema Estratégico de Transporte Público de Pasto, además de datos referentes a sitios de interés de la ciudad de Pasto.

Diseño y construcción de la base de datos

El objetivo de esta etapa fue crear el repositorio de datos correspondiente a la información sobre rutas y paraderos del Sistema Estratégico de Transporte de San Juan de Pasto, SETP y de sitios de interés, información recolectada en la etapa anterior.

Diseño y construcción de la ontología

El objetivo de esta etapa fue diseñar y construir una ontología de sitios de interés de la ciudad de Pasto que soporte las búsquedas inteligentes que realiza SITApp. La ontología basa su modelo en la base de datos construida en la fase anterior.

Diseño y construcción de SITApp

El objetivo de esta etapa fue diseñar y desarrollar el aplicativo basado en arquitectura cliente servidor, aplicativo constituido por una aplicación móvil para dispositivos Android y un web service para el Sistema Estratégico de Transporte Público de la ciudad de Pasto.

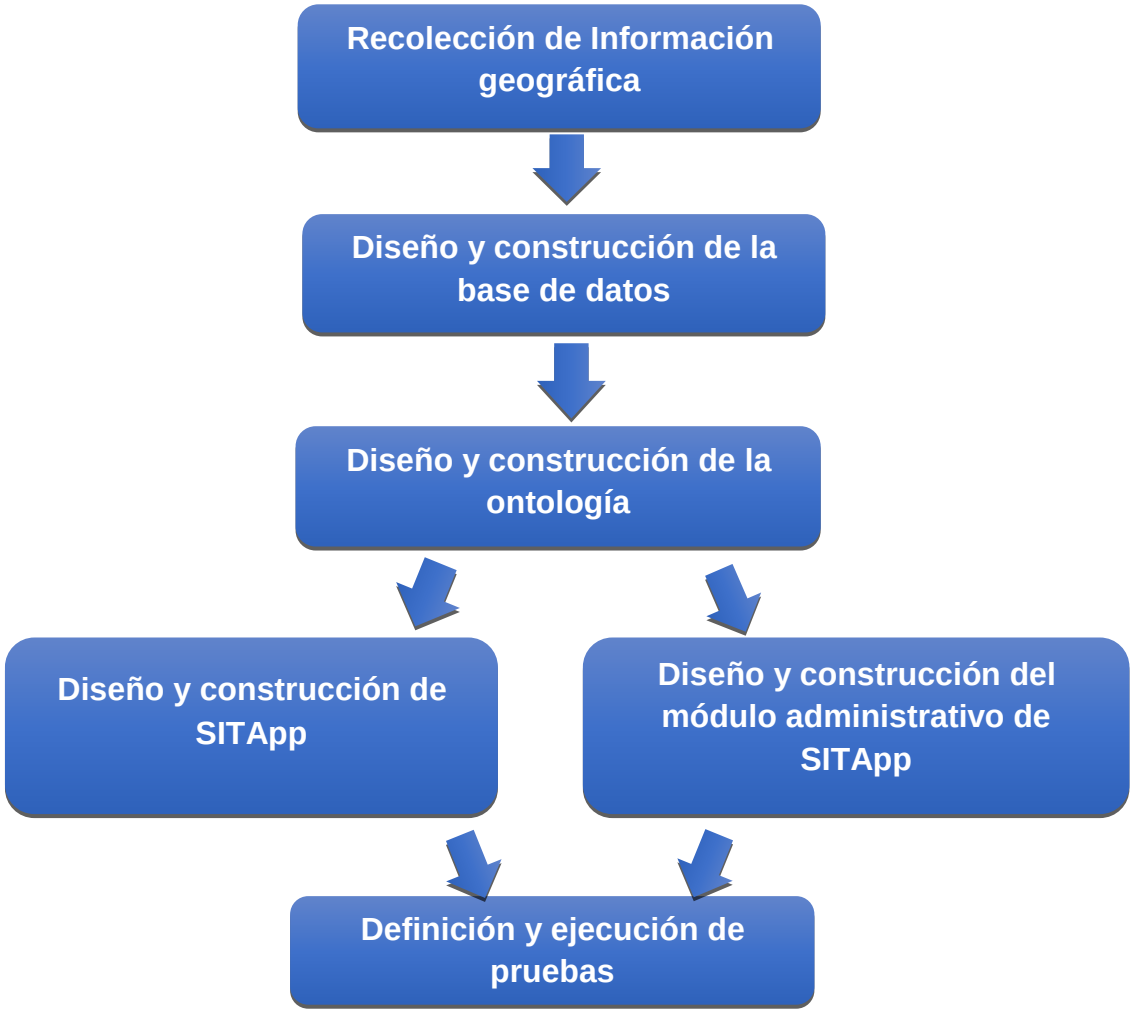
Diseño y construcción del módulo administrativo de SITApp

El objetivo de esta etapa fue diseñar y construir un módulo administrativo para el aplicativo móvil, el cual permita realizar operaciones CRUD sobre la base de datos para que esta se encuentre lo más actualizada posible.

Definición y ejecución de pruebas

El objetivo de esta fase fue diseñar y realizar pruebas de funcionamiento del sistema en su totalidad, aplicativo móvil y módulo administrativo, con el fin de lograr percibir errores, funcionalidades faltantes o incompletas.

Figura 23. Metodología de la investigación.



2. ARQUITECTURA DEL APLICATIVO

SITApp está basado en una arquitectura cliente servidor de tres capas como se muestra en la Figura 24.

Capa de presentación

En esta capa se encuentran albergados todos los elementos de presentación gráficos para el usuario haciendo que los procesos que realiza el servidor para gestionar toda la información sean transparentes y disminuyendo cargas en el procesamiento de datos del lado del cliente, los elementos que le permiten interactuar con la aplicación y sus diferentes opciones, son:

- Los gestores de mapas, que permiten visualizar toda la información georreferenciada acerca del SETP almacenada en el servidor y que se muestra de distintas maneras de acuerdo a los distintos tipos de consultas que el usuario realice en su momento.
- El buscador semántico encargado de gestionar las consultas del usuario y realizar las peticiones hacia el servidor.
- Los formularios CRUD para la gestión de información geográfica.

Lógica de negocio

En esta capa se encuentran albergados los distintos algoritmos que se encargan de gestionar las distintas peticiones recibidas por la capa de presentación, aquí se lleva a lugar la tarea de procesamiento y gestión de toda la información georreferenciada mediante:

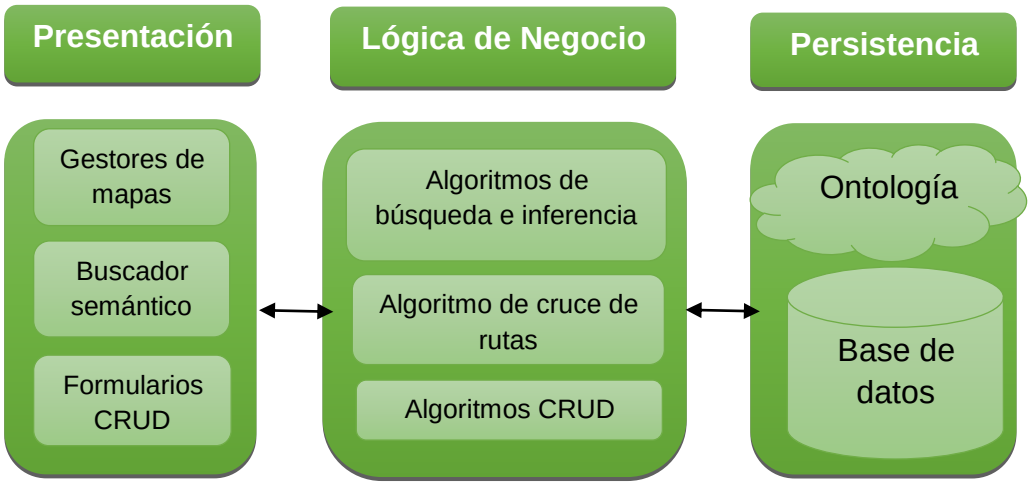
- Algoritmos de búsqueda e inferencia, encargados de gestionar consultas complejas realizando inferencias semánticamente haciendo uso de la ontología.
- Algoritmo de cruce de rutas, encargado de realizar la búsqueda de la o las rutas que llevan al usuario desde un punto geográfico “A” hasta un punto geográfico “B”.
- Algoritmos CRUD, encargados de realizar operaciones de gestión de información georreferenciada geográficamente acerca del SETP almacenada en la base de datos.

Capa de persistencia

En esta capa se encuentra la base de datos de SITApp soportada en el gestor PostgreSQL, que alberga toda la información georreferenciada acerca de las

rutas y paraderos del SETP y los sitios de interés más relevantes de la ciudad de Pasto.

Figura 24. Arquitectura de SITApp y el módulo de ontología

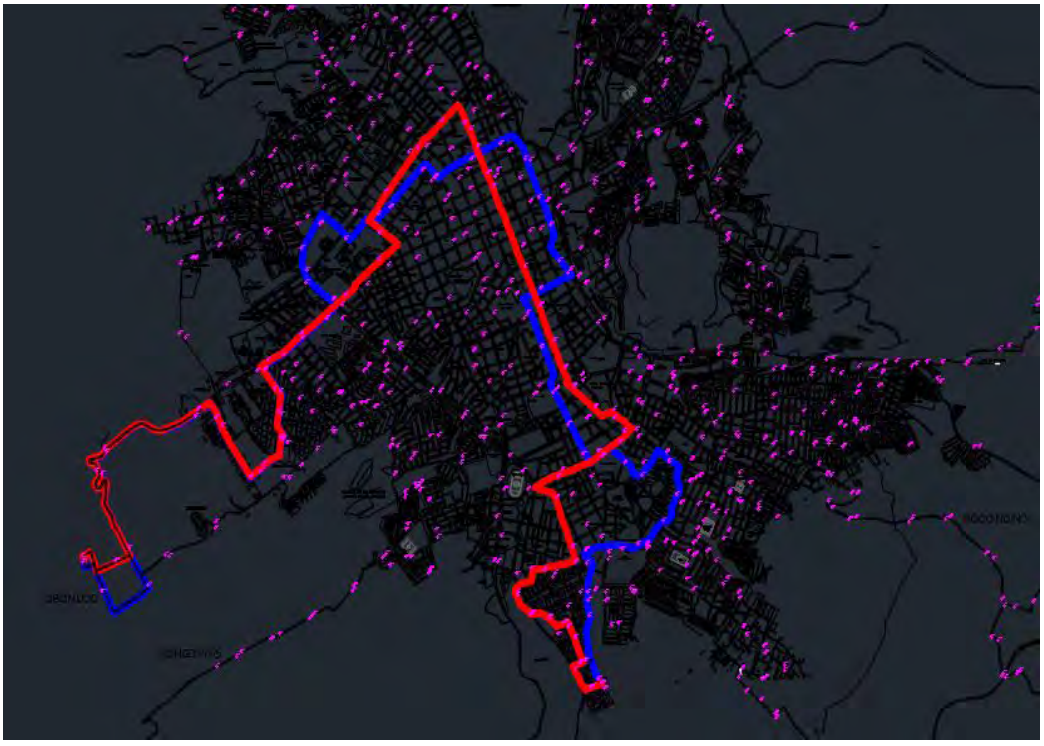


3. RESULTADOS

3.1. FASE 1: RECOLECCIÓN DE INFORMACIÓN GEOGRÁFICA

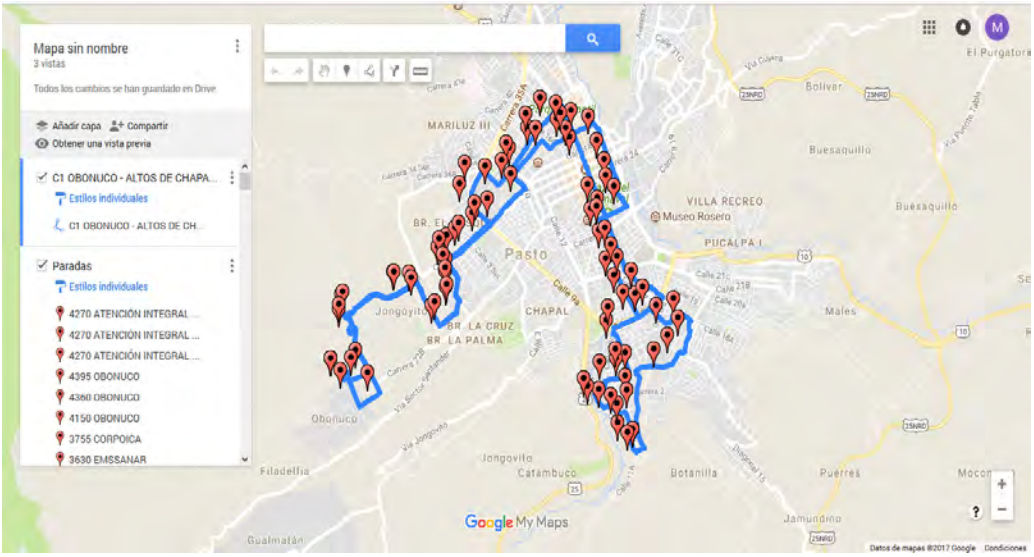
Esta fase se logró concretar con ayuda principalmente de AVANTE, entidad que proporcionó toda la información necesaria referente al Sistema Estratégico de Transporte Público de Pasto, se recibió un archivo de AutoCAD que contenía el mapa de Pasto y en diferentes capas cada uno de los recorridos de las rutas y además cada uno de los paraderos autorizados del sistema como se puede observar en la Figura 25, posteriormente se recibió una carpeta con archivos de Google Maps KML, cada archivo contenía información correspondiente a una ruta, su recorrido y cada uno de sus paraderos asociados como se puede observar en la Figura 26. Esta información fue trasladada hacia archivos OpenStreetMaps (.osm) con ayuda de la herramienta libre JOSM, en la Figura 27 se puede observar el resultado de este proceso.

Figura 25. Recorrido completo ruta C1 y todos los paraderos del SETP



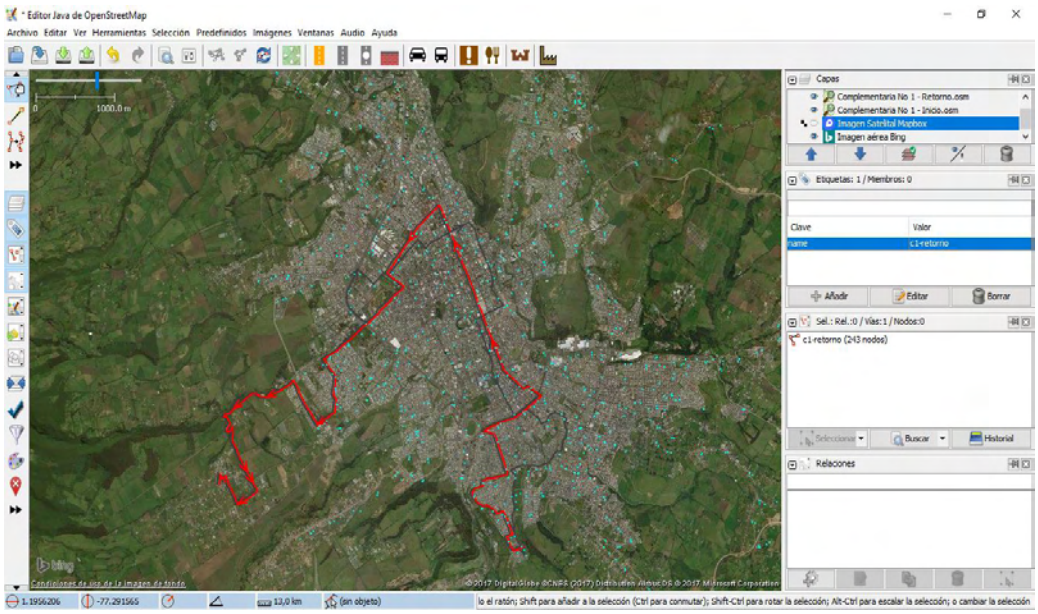
Fuente: [28]

Figura 26. Visualización archivo KML de la ruta C1, recorrido y paraderos.



Fuente: [28]

Figura 27. Visualización de información trasladada a JOSM, ruta C1 y todos los paraderos del SETP.



Como resultado de este proceso se digitalizaron 23 rutas, 7 estratégicas y 16 complementarias en 46 archivos OSM debido a que por cada ruta se obtuvo 2 geometrías, una de inicio y otra de retorno. Además, se digitalizaron 746 paraderos en un archivo OSM.

Se recibió también un archivo Excel que contiene información más detallada de cómo opera el sistema, a continuación se lista la información referente de cada una de las rutas:

- Su nombre clave.
- Horarios del primer y del último despacho.
- Frecuencia de despacho.
- Tiempos de recorridos.
- Distancia total de recorrido.

A continuación, se lista la información referente cada uno de los paraderos:

- Id del paradero.
- Nombre del paradero.
- Tipo del paradero.
- Ubicación del paradero.

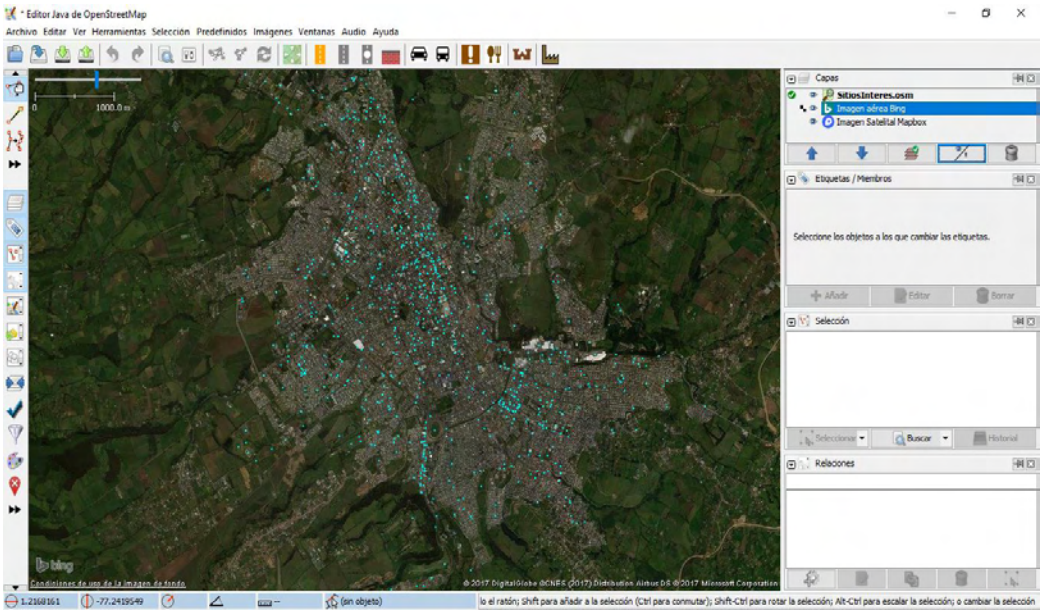
Además, el Excel contiene información sobre que paraderos pertenecen a cada una de las rutas.

Para la recolección de información referente a los sitios de interés se utilizaron mapas turísticos de San Juan de Pasto, de los cuales se decidió seleccionar solamente las siguientes categorías:

- Centros de atención imnediata CAI, estaciones de policía, estaciones de bomberos y guarniciones militares.
- Supermercados, centros comerciales y plazas de mercado.
- Gobernación, alcaldía y entidades institucionales.
- Universidades e instituciones educativas.
- Centros artesanales, concha acústica, bibliotecas, teatros y museos.
- Ancianatos, cementerios, terminal de transportes, cárcel, clubs, monumentos y fundaciones.
- Plazas y parques.
- Centros deportivos y coliseos.
- Hospitales y clínicas.
- Iglesias.
- Bancos.

Se procedió una vez seleccionadas estas categorías a digitalizar y georreferenciar cada uno de los sitios de interés con la herramienta JOSM obteniendo como resultado 502 sitios digitalizados, como se observa en la Figura 28.

Figura 28. Digitalización de sitios de interés.



Adicionalmente, se observó la necesidad de incluir barrios, corregimientos y veredas para hacer más completas las búsquedas de los usuarios. Para este objetivo se recurrió a los resultados del proyecto de investigación de la Universidad de Nariño llamado RIKUNA[29] dentro de los cuales se encuentran archivos OpenStreetMaps que contienen las geometrías de los barrios, corregimientos y veredas como resultado se obtuvo 316 barrios, ver Figura 29, 19 corregimientos, ver Figura 30 y 129 veredas digitalizados, ver Figura 31. Se decidió tomar como referencia un punto estratégico de la geometría para hacer referencia al barrio, corregimiento o vereda.

Figura 29. Digitalización barrios.

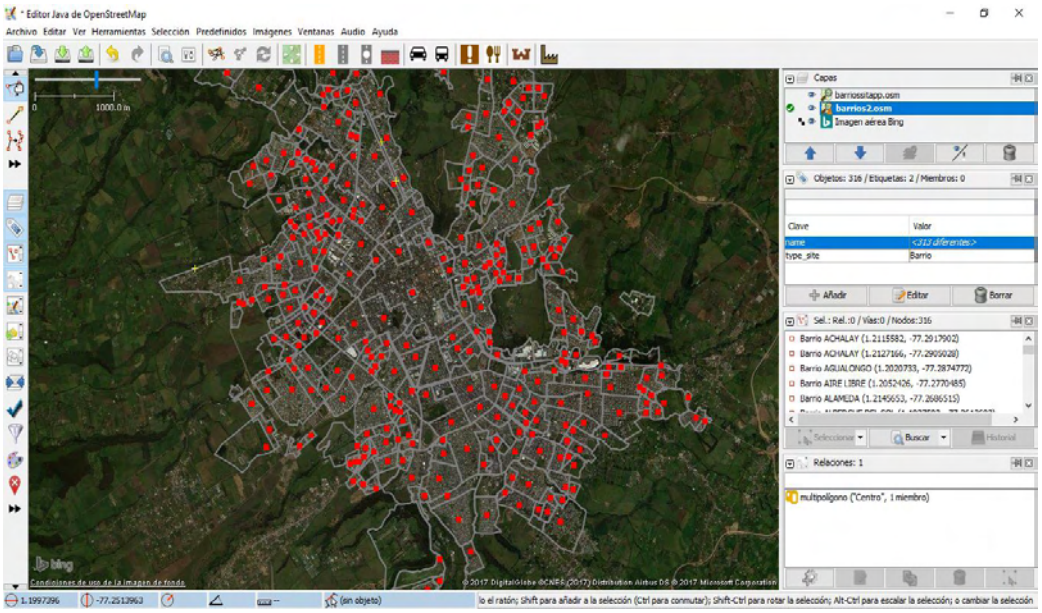


Figura 30. Digitalización de corregimientos.

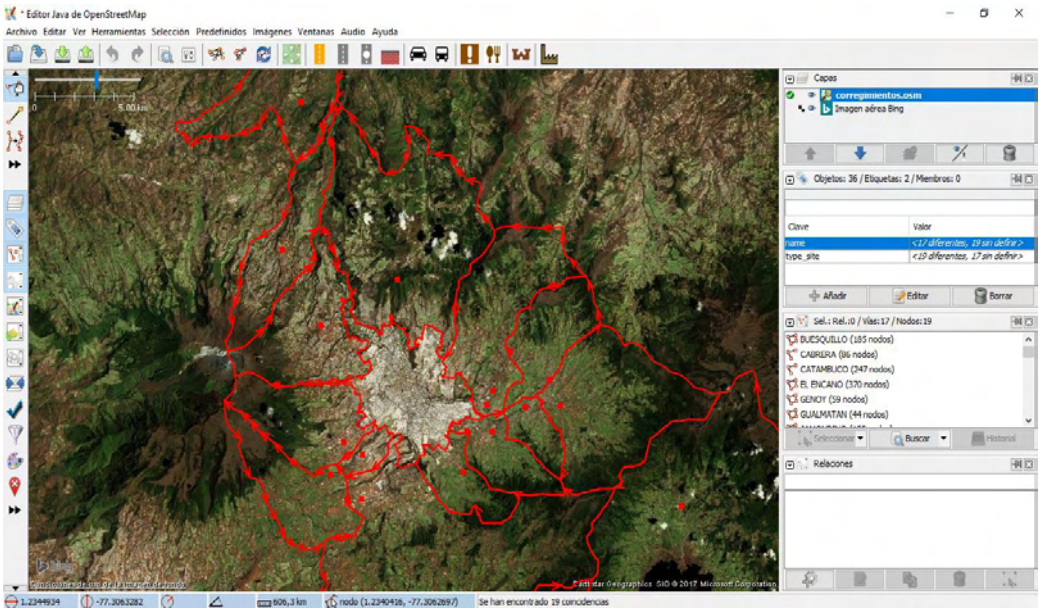
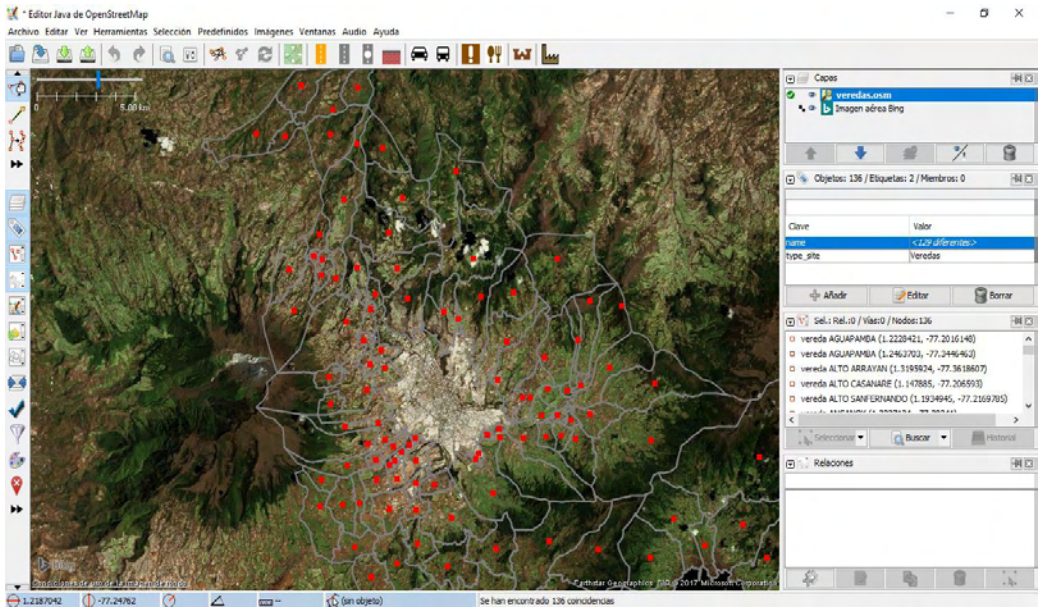


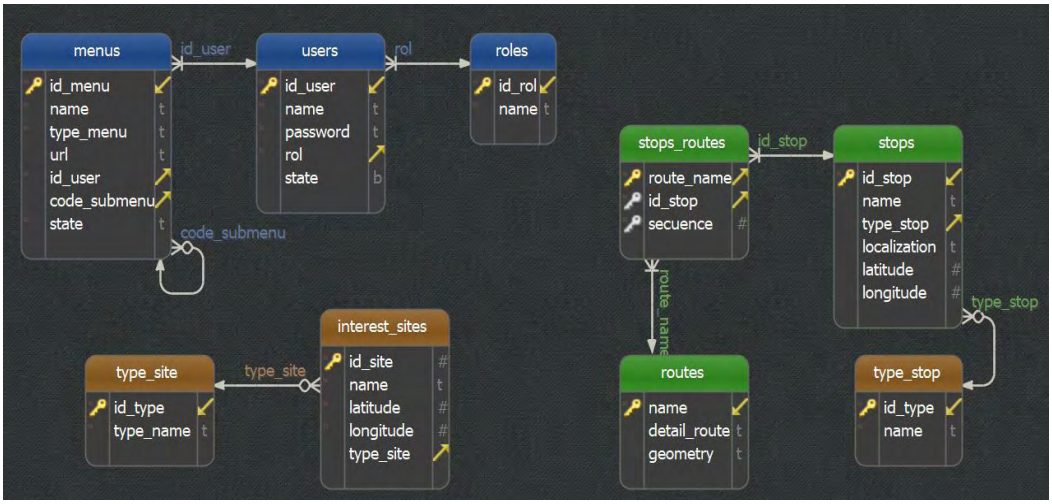
Figura 31. Digitalización de veredas.



3.2. FASE 2: DISEÑO Y CONSTRUCCIÓN DE LA BASE DE DATOS

Realizando un análisis de la información recolectada en la fase anterior se construyó el modelo de la base de datos como se muestra en la Figura 32.

Figura 32. Modelo de la base de datos.



A continuación, se describen, cada una de las entidades:

ROUTES:

Entidad que representa a las rutas del SETP cuyos atributos son los siguientes:

- Name, hace referencia al nombre clave de la ruta y es la llave primaria de la entidad.
- Detail, hace referencia al nombre detallado de la ruta.
- Geometry, hace referencia a la representación de la geometría en el tipo de dato LYNE_STRING de PostGIS.

STOPS:

Entidad que representa a los paraderos del SETP cuyos atributos son los siguientes:

- Id_stop, hace referencia al id asignado por Avante a cada uno de los paraderos y es la llave primaria de la entidad.
- Name, hace referencia al nombre del paradero.
- Type_stop, hace referencia al tipo de paradero, llave foránea que referencia la entidad TYPE_STOP.
- Localization, hace referencia a la ubicación del paradero en la ciudad, puede ser una dirección o un lugar en general de la ciudad.
- Latitude, latitud expresada en el sistema de coordenadas WGS84[30].
- Longitude, longitud expresada en el sistema de coordenadas WGS84[30].

TYPE_STOP:

- Id_type, hace referencia a un número secuencial y es la llave primaria de la entidad.
- Name, hace referencia al nombre del tipo de paradero.

STOPS_ROUTES:

Es una tabla intermedia encargada de enlazar las tablas ROUTES y STOPS y que representa los paraderos que posee cada ruta. Los atributos de esta entidad son los siguientes:

- Id_stop, hace referencia al id del paradero, llave foránea que referencia a la entidad STOPS.
- Name, hace referencia al nombre clave de la ruta, llave foránea que referencia a la entidad ROUTES.
- Secuence, hace referencia a la secuencia correspondiente a la ruta a la que pertenece.

INTEREST_SITES:

Entidad que representa los sitios de interés de la ciudad de Pasto cuyos atributos son los siguientes:

- Id_site, hace referencia a un número secuencial, es la llave primaria de la entidad.
- Type_site, hace referencia al id del tipo de sitio de interés, es llave foránea que referencia a la entidad TYPE_SITE.
- Name, hace referencia al nombre del sitio.
- Latitude, latitud expresada en el sistema de coordenadas WG S84[30].
- Longitude, longitud expresada en el sistema de coordenadas WGS 84[30].

TYPE_SITE:

Entidad que representa las categorías de sitios de interés y sus atributos son los siguientes:

- Id_type, representa el identificador numérico del tipo de sitio de interés.
- Type_name, representa el nombre de la categoría del sitio de interés.

MENUS:

Tabla que almacena la información sobre las diferentes opciones que tiene el usuario en el módulo administrativo de SITApp.

- Id_menu, hace referencia al identificador único de cada registro en la tabla.
- Name, nombre de la opción a mostrar en el módulo administrativo.
- Type, hace relación a si es un menu principal o es un submenu.
- Url, dirección solicitada cuando se presiona una opción dentro de la vista.
- Id_user, referencia hacia el usuario que tiene permisos para hacer uso de una opción determinada.
- State, estado de la opción, puede ser 1 ó 0.
- Code_submenu, si la opción es un submenu entonces se guarda la referencia hacia simismo.

USERS:

Tabla que almacena la información de los usuarios que puede hacer uso del módulo administrativo.

- Id_user, identificador único de cada usuario.
- NAme, nombre de usuario o login.
- Password, contraseña del usuario.
- Rol, representa el tipo de rol que puede tener el usuario. Puede ser administrador o un usuario sin privilegios.

ROLES:

Esta tabla almacena la información de los diferentes roles que puede tener un usuario en el sistem.

- Id_rol, identificador único de cada rol.
- Name, nombre del rol, puede ser admin o user.

Proceso de población de la base de datos

Se hizo uso de varios scripts en Python[31] del proyecto SIGEODEP-SIG[32], los cuales son software libre (licencia GNU Gegeral Public Licence v3) para realizar la migración de los datos desde los archivos OSM generados en la etapa anterior hacia la base de datos relacional con el modelo de datos generado en PostgreSQL[33] con extensión de datos PostGIS[34]. A continuación, una breve descripción de cada uno de los scripts:

- Un script para la migración de rutas del SETP que extrae de los archivos OSM[35] el nombre de la ruta y su geometría.
- Un script para la migración de paraderos autorizados del SETP que extrae del archivo OSM el nombre del paradero y su respectivo tipo.

- Un script para la migración de sitios de interés que extrae del archivo OSM de los sitios de interés migrándolo hacia una tabla temporal de sitios de interés.

Posteriormente, se realizó la normalización de la tabla temporal de sitios de interés insertando inicialmente en la tabla TYPE_SITE los diferentes tipos de sitios de interés (categorías) y luego insertando en la tabla INTEREST_SITES la totalidad de los sitios de interés migrados.

Para la migración de datos de los archivos Excel que contienen la información de los paraderos que pertenecen a cada una de las rutas y su respectiva secuencia se exportó estos datos a un archivo CSV y posteriormente se realizó la inserción de estos datos en la tabla intermedia STOPS_ROUTES.

3.3. FASE 3. DISEÑO Y CONSTRUCCIÓN DE LA ONTOLOGÍA

Para la construcción de la ontología se siguió una serie de pasos que serán descritos a continuación, tomando los conceptos básicos descritos por la base de datos y teniendo en cuenta que para la búsqueda semántica se es necesario simplemente la búsqueda de sitios de interés. La ontología fue construida haciendo uso de la metodología Methontology[36] y con la ayuda de la herramienta Protegé[37], además está basada en el lenguaje estándar OWL (Web Ontology Language).

Paso 1. Construcción del glosario de términos

Descripción de todos los términos relevantes del dominio de la ontología (conceptos, instancias, atributos, relaciones entre conceptos, etc.), sus descripciones en lenguaje natural, y sus sinónimos.

Tabla 1. Glosario de términos de la ontología

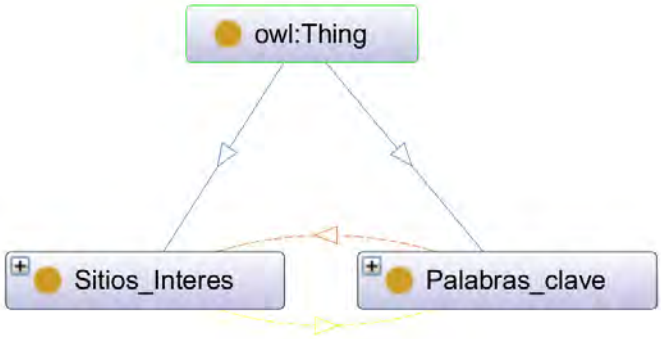
Nombre	Sinónimos	Descripción	Tipo
Sitio de interés	Lugar en la ciudad	Se refiere a un sitio estratégico o simbólico en la ciudad, por ejemplo plazas, centros comerciales, etc.	Concepto
Palabra clave	Keyword	Se refiere a las palabras claves o categorías que en general describen a los sitios de interés.	Concepto
Id	identificador	Es el identificador de cada instancia de los sitios de interés,	Atributo de instancia

		tomado de la base de datos	
Sinónimos		Son los posibles sinónimos que posea el nombre de la instancia del sitio de interés.	Atributo de instancia
Palabra	Categoría	Son los sinónimos o palabras claves de cada categoría de los sitios de interés.	Atributo de instancia
Describe (Palabra clave, Sitio de interés)	Detalla	La palabra clave o categoría describe el sitio de interés.	Relación entre conceptos
Es descrito por (Sitio de interés, Palabra Clave)	Detallado por	El sitio de interés es descrito por la palabra clave	Relación entre conceptos

Paso 2. Construcción de taxonomías de conceptos

Se seleccionó del glosario aquellos términos que son conceptos, esta taxonomía describe la jerarquía de los conceptos, ver figura 33.

Figura 33. Taxonomías de conceptos



Paso 3. Construcción de un diagrama de relaciones binarias

Mediante el siguiente diagrama, ver figura 34, se establecen las relaciones existentes entre conceptos de la ontología y sus relaciones inversas.

Figura 34. Diagrama de relaciones binarias

Paso 4. Construcción del diccionario de conceptos

Se especifican cuáles son las propiedades que describen a cada concepto de la taxonomía, las uniones del diagrama de relaciones binarias y las instancias de cada uno de los conceptos.

Tabla 2. Diccionario de conceptos de la ontología

Nombre del concepto	Instancias	Atributos de instancia	Relaciones
Sitios de interés	Plaza de Nariño, C.C. Unicentro, Iglesia de San Juan, Universidad de Nariño, etc.	id, sinónimos	Es descrito por
Palabras clave	Plaza, centro comercial, templos, supermercado, universidad, etc.	palabra	Describe

Paso 5. Descripción de las relaciones binarias

Se especifica de cada relación binaria nombre, nombres de sus conceptos origen y destino, cardinalidad y relación inversa.

Tabla 3. Relaciones binarias de la ontología

Nombre de la relación	Concepto origen	Cardinalidad máxima	Concepto destino	Relación inversa
Es descrito por	Sitio de interés	N	Palabras clave	Describe
Describe	Palabras clave	N	Sitios de interés	Es descrito por

Paso 6. Modelo final de la ontología

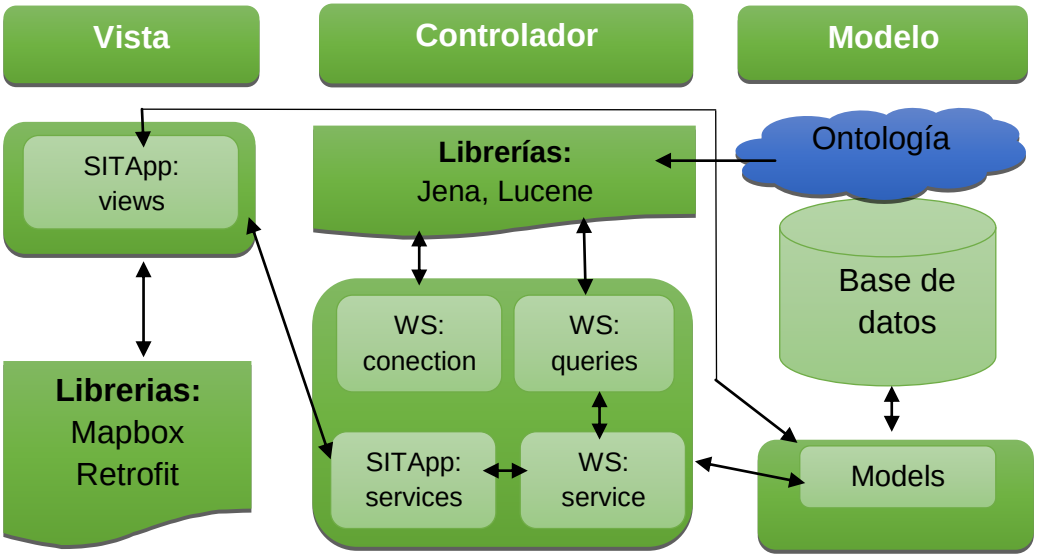
Se especifica el modelo general de la ontología con sus relaciones binarias y la jerarquía entre clases descrito, ver figura 35, construido en la herramienta de software libre Protegé[37], *La construcción detallada de la ontología en la herramienta se puede observar en el Anexo F.*

Figura 35. Modelo final de la ontología

3.4. FASE 4. DISEÑO Y CONSTRUCCIÓN DE SITAPP

Para la implementación de la arquitectura cliente servidor y construir SITApp se hizo uso del modelo vista controlador (MVC) que se describe en la Figura 36. Para el desarrollo exitoso de esta fase fue necesario realizar dos aplicaciones, SITApp para dispositivos móviles y un Web Service para la gestión de consultas.

Figura 36. Modelo Vista Controlador de SITApp por paquetes



A continuación, se explica la construcción de cada aplicativo por separado.

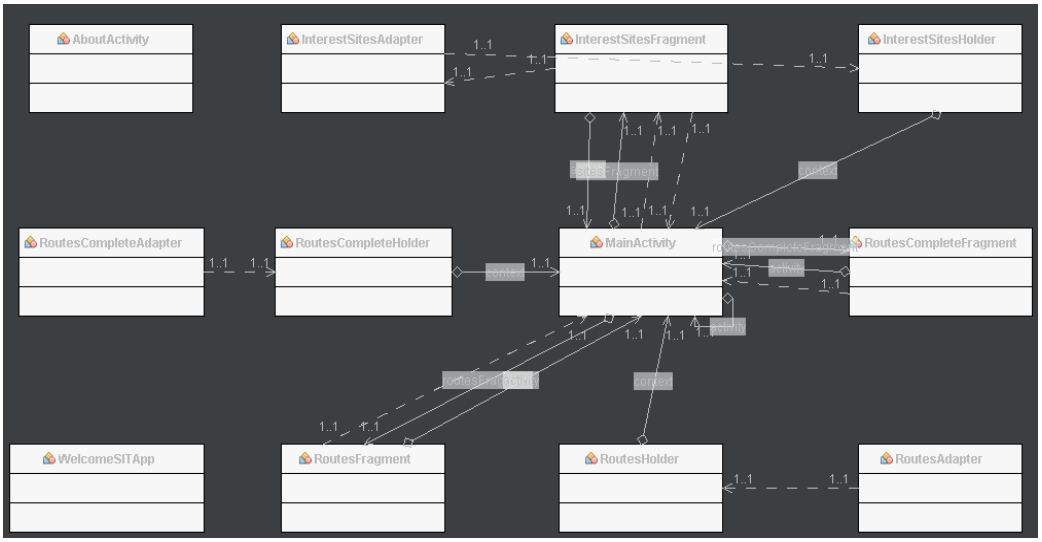
3.4.1. SITApp

SITApp es una aplicación open source para dispositivos móviles Android[38], SITApp es la aplicación que permite desplegar y visualizar la información georreferenciada del Sistema Estratégico de Transporte de Pasto (SETP), información que se encuentra almacenada en la base de datos. Fue desarrollada bajo el lenguaje de programación Java[39], lenguaje nativo de desarrollo para Android que además usa para el manejo de recursos gráficos y de texto, archivos XML. Está disponible para las versiones de Android 4.2 en adelante. SITApp utiliza la librería de Mapbox SDK para Android para la gestión de mapas y la librería Retrofit para la comunicación entre SITApp y el Web Service**.**

Implementación de la vista

La implementación de la vista del MVC de SITApp se realizó mediante el paquete views cuyo modelo general es el siguiente y que se describe a continuación, en la Figura 37 se muestra el diagrama de clases:

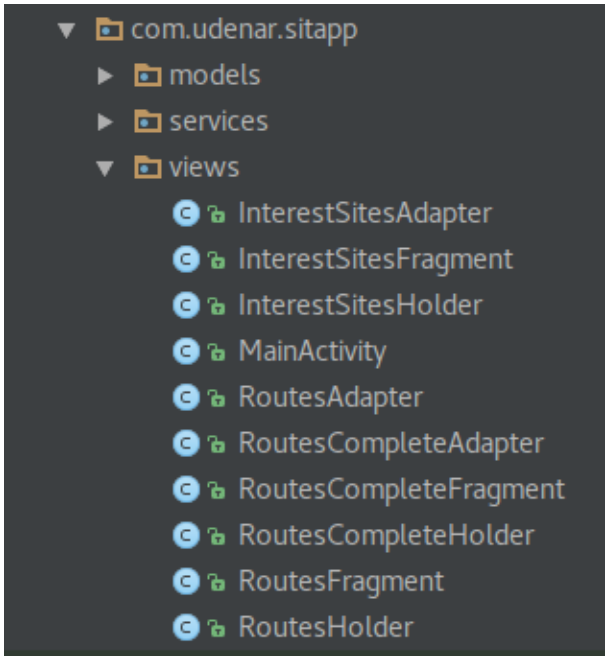
Figura 37. Diagrama de clases de la vista del SITApp



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo D Manual del programador de SITApp.

- **com.udenar.sitapp.views:** este paquete es el encargado de la representación gráfica, gestionando las distintas búsquedas e interacciones del usuario con la aplicación y desplegando los distintos resultados en el mapa o en las listas dinámicas tanto de rutas como de sitios de interés (ver figura 38).

Figura 38. Clases del paquete views de SITApp



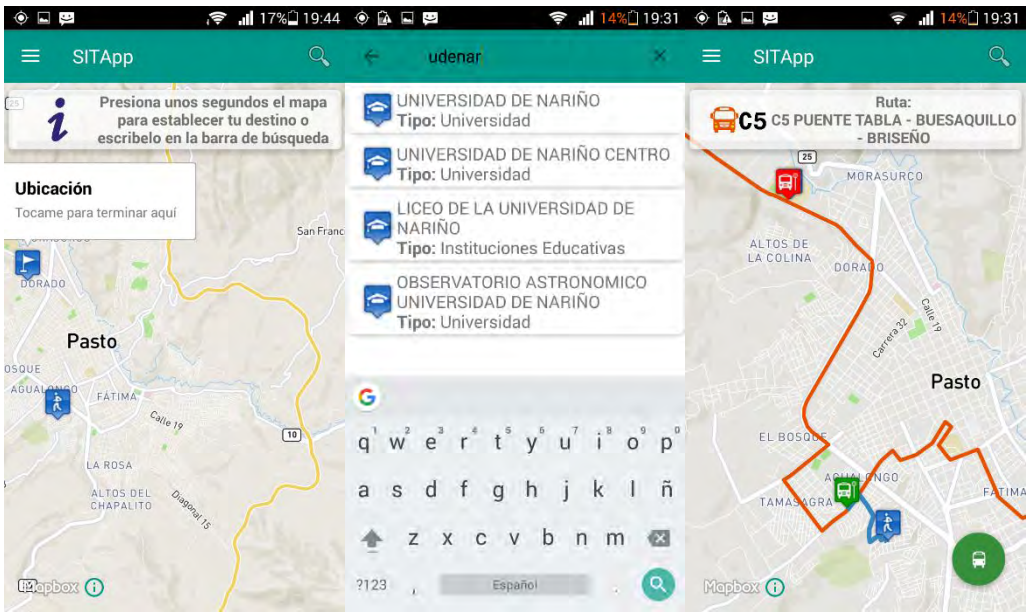
- **InterestSitesAdapter.java:** es la clase encargada de establecer comunicación entre el Fragment y el RecyclerView que contiene los sitios de interés.
- **InterestSitesFragment.java:** es la clase encargada de albergar los objetos visuales de la lista de sitios de interés e instanciar el Adapter necesario para interactuar con cada uno de los ítems de la lista.
- **InterestSitesHolder.java:** es la clase que representa como tal al objeto InterestSite dentro del RecyclerView.
- **MainActivity.java:** es el Activity principal de la aplicación, encargado de mostrar el mapa, la lista de sitios de interés y la lista de rutas. Alberga la mayoría de funciones principales como establecer comunicación con el Web Service mediante el paquete services, recibir los JSON e instanciarlos mediante el paquete models y además mostrar los diferentes Fragments dependiendo de la función requerida por el usuario.
- **RoutesAdapter.java:** es la clase encargada de establecer comunicación entre el Fragment y el RecyclerView que contiene las rutas del SETP.
- **RoutesFragment.java:** es la clase encargada de albergar los objetos visuales de la lista de rutas del SETP e instanciar el Adapter necesario para interactuar con cada uno de los ítems de la lista.
- **RoutesHolder.java:** es la clase que representa como tal al objeto Routes dentro del RecyclerView.

- **RoutesCompleteAdapter.java:** es la clase encargada de establecer comunicación entre el Fragment y el RecyclerView que contiene las rutas del SETP y todos sus paraderos.
- **RoutesCompleteFragment.java:** es la clase encargada de albergar los objetos visuales de la lista de rutas del SETP y todos sus paraderos e instanciar el Adapter necesario para interactuar con cada uno de los ítems de la lista.
- **RoutesCompleteHolder.java:** es la clase que representa como tal al objeto RoutesComplete dentro del RecyclerView.

SITApp tiene varias formas de realizar búsquedas a fin de obtener un resultado acorde a las peticiones realizadas por el usuario. A continuación se presenta una descripción de las opciones de búsqueda del aplicativo.

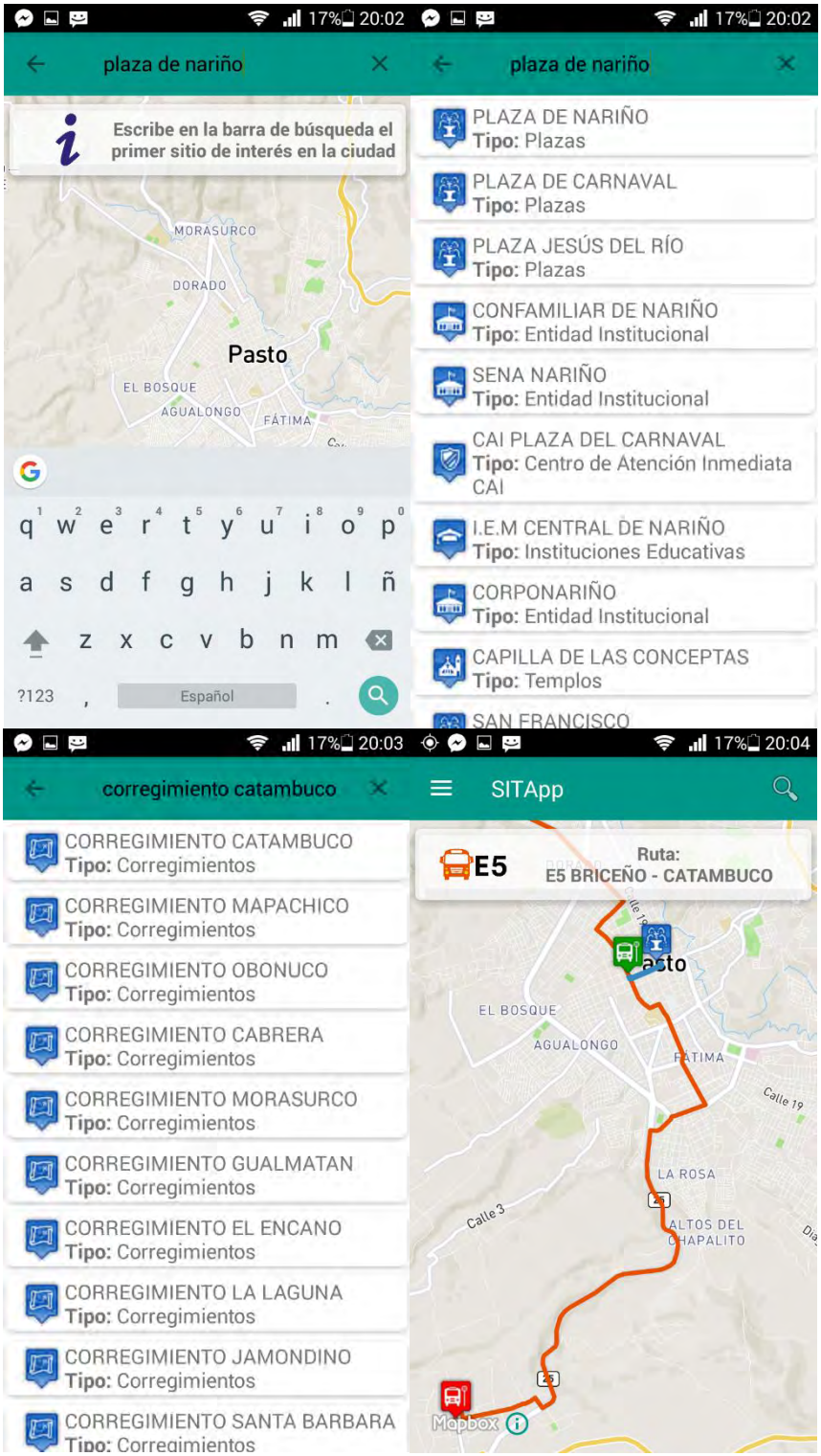
La principal opción de búsqueda de SITApp consiste en usar la posición del GPS del dispositivo y que el usuario presione el mapa para establecer su destino o usar el buscador semántico para encontrar el sitio de interés de la ciudad hacia donde desea ir y así la aplicación determina que ruta(s) van hacia ese lugar como se muestra en la Figura 39.

Figura 39. Ejemplo búsqueda mediante GPS y un sitio de interés.



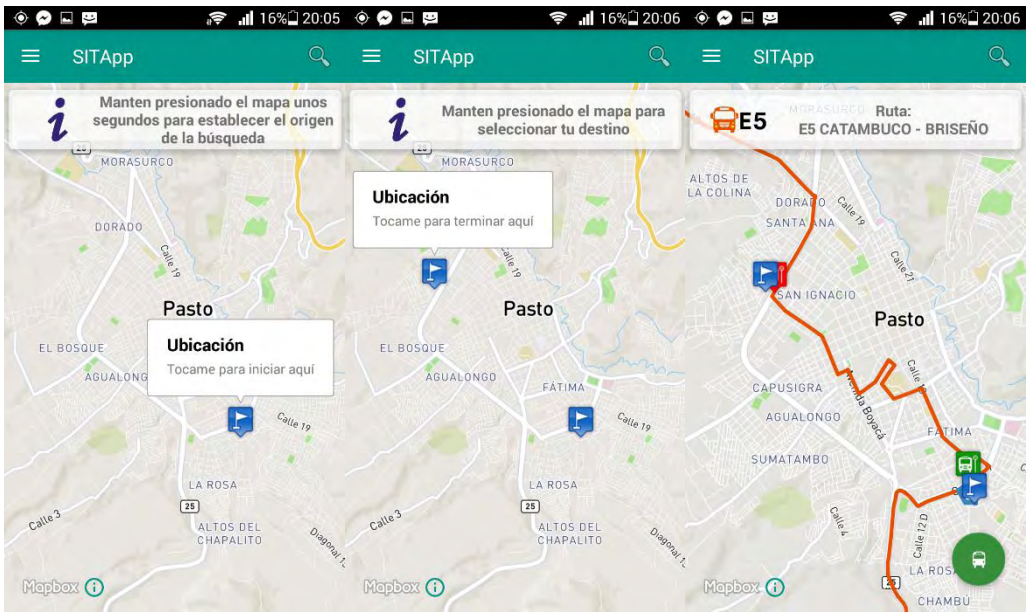
La segunda opción de búsqueda de SITApp es buscar una ruta por dos sitios de interés, el usuario mediante el buscador semántico debe encontrar su origen y su destino dentro de la ciudad y la aplicación posteriormente despliega el resultado de la búsqueda de rutas como se puede observar en la Figura 40.

Figura 40. Ejemplo búsqueda mediante dos sitios de interés



La tercera opción de búsqueda consiste en que el usuario toque prolongadamente el mapa dos veces para establecer su origen y su destino, la aplicación posteriormente realiza la búsqueda y le muestra al usuario la ruta(s) que van desde el punto A hasta el punto B como se muestra en la Figura 41.

Figura 41. Ejemplo de búsqueda tocando dos puntos en el mapa



Adicionalmente SITApp cuenta con dos opciones para mostrar el recorrido completo de las rutas complementarias y estratégicas y todos los paraderos asociados a cada una de las rutas como se observa en la Figura 42. Además SITApp tiene una opción adicional que permite aprovechar la búsqueda semántica para ubicar en el mapa de la ciudad un sitio de interés como se puede observar en la Figura 43.

Figura 42. Ejemplo función de búsqueda rutas estratégicas y complementarias

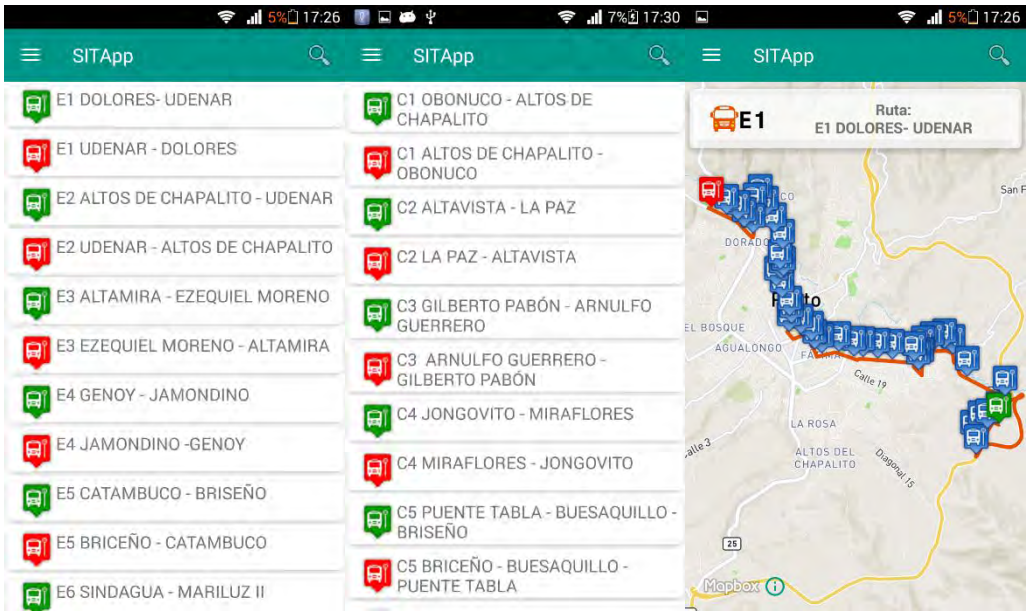
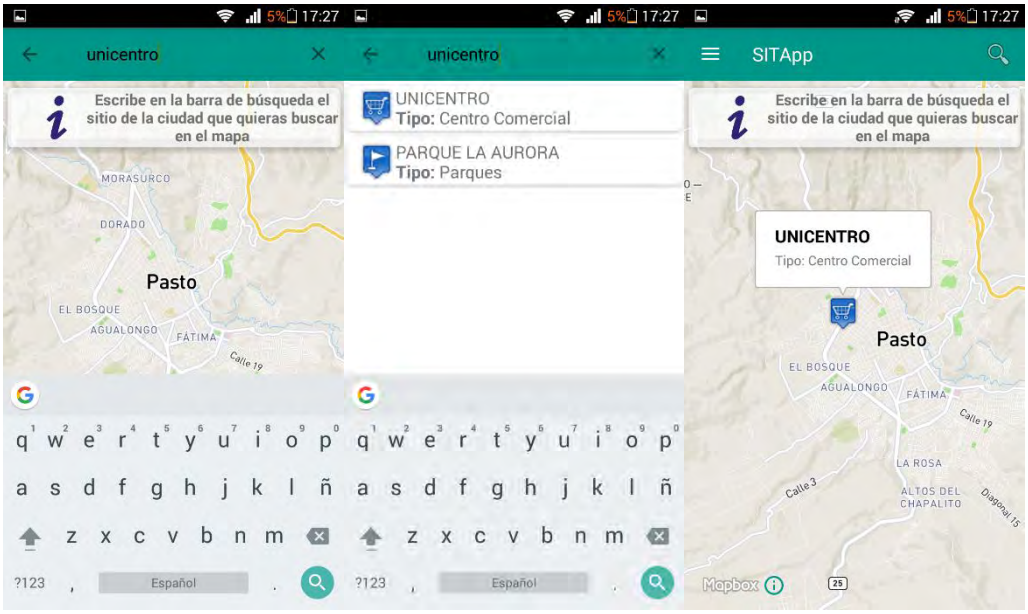


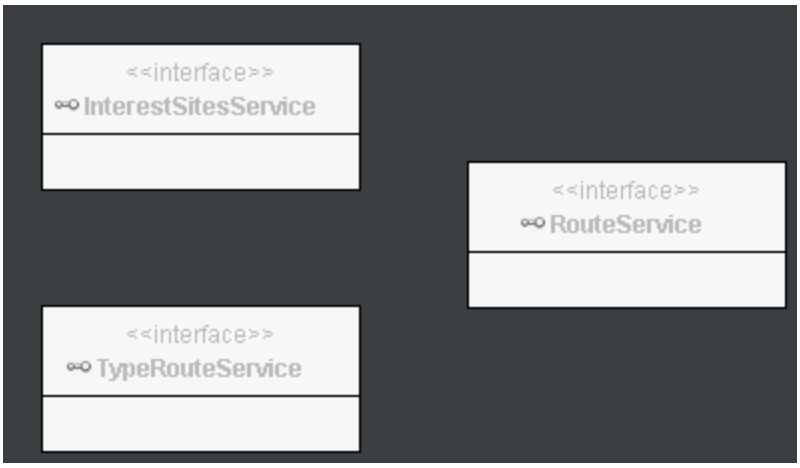
Figura 43. Ejemplo búsqueda por un sitio de interés



Implementación del controlador

La implementación del controlador del MVC de SITApp se realizó mediante el paquete services que se describe a continuación, en la Figura 44, se muestra el diagrama de clases:

Figura 44. Diagrama de clases del controlador.

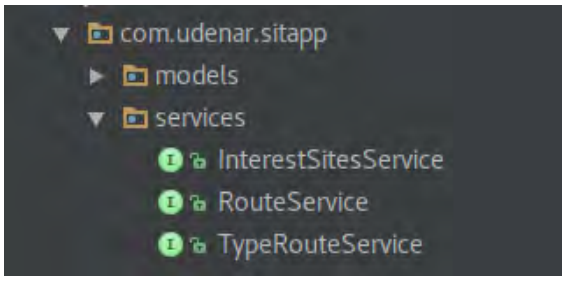


La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo D Manual del programador de SITApp.

- **com.udenar.sitapp.services:** este paquete es el encargado de la comunicación entre SITApp y el web service mediante interfaces, realiza las peticiones HTTP mediante el método GET para obtener mejores

tiempos de respuesta recibiendo del web service archivos JSON, ver figura 45 para conocer los componentes de este paquete.

Figura 45. Clases del paquete services de SITApp

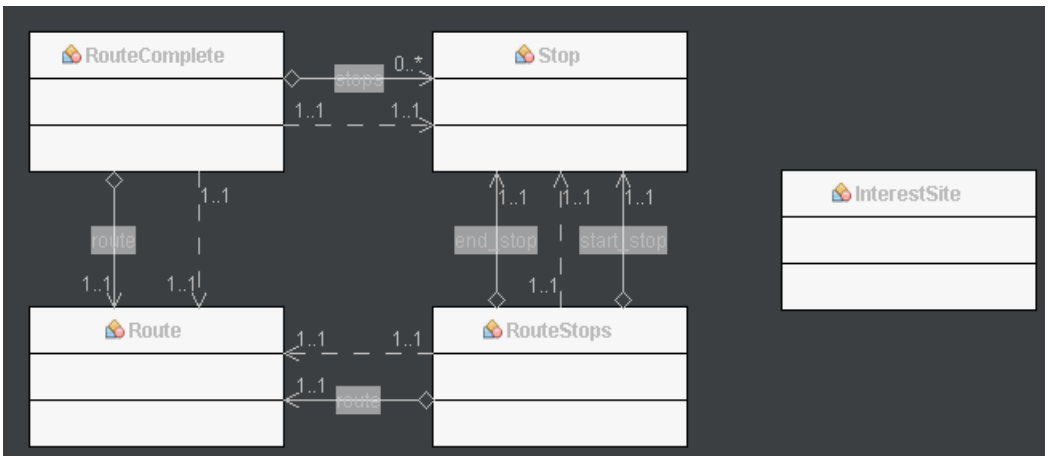


- **InterestSitesService.java:** interface encargada de enviar al Web Service las consultas del usuario y recibiendo como respuesta los sitios de interés instanciados como una lista de objetos InterestSites.
- **RouteService.java:** interface encargada de enviar al Web Service las coordenadas de los puntos A y B seleccionados por el usuario y recibiendo como respuesta las rutas que llevan al usuario hacia su destino instanciadas como una lista de objetos Routes.
- **TypeRouteService.java:** interface encargada de enviar al Web Service la petición de obtener información de las rutas complementarias o estratégicas del SETP, información instanciada en una lista de objetos RouteComplete.

Implementación del modelo

La implementación del modelo del MVC de SITApp, se realizó mediante el paquete models que se describe a continuación, en la Figura 46 se muestra el diagrama de clases:

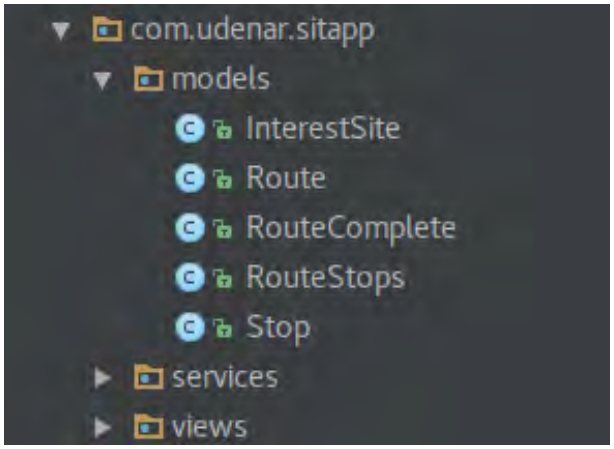
Figura 46. Diagrama de clases del modelo.



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo D Manual del programador de SITApp.

- **com.udenar.sitapp.models:** este paquete es el encargado de administrar los modelos que representan los objetos de la base de datos de SITApp, ver figura 47.

Figura 47. Clases del paquete models de SITApp



  - ❖ **InterestSite.java:** clase que representa a un sitio de interés del municipio de Pasto.
  - ❖ **Route.java:** clase que representa a una ruta del SETP de Pasto.
  - ❖ **RouteComplete.java:** clase que representa a una ruta del SETP con todos sus paraderos autorizados asociados.
  - ❖ **RouteStops.java:** clase que representa a una ruta del SETP de Pasto, el paradero de origen y el paradero de destino para el usuario.
  - ❖ **Stop.java:** clase que representa a un paradero autorizado del SETP de Pasto.

La descripción detallada de cada una de estas funcionalidades y la manera correcta de utilizar SITApp se encuentra registrado en el manual de usuario, el cual está en el Anexo B

3.4.2. SITApp web service

Web Service SITApp es una aplicación desarrollada bajo el lenguaje de programación Java Enterprise Edition 7. Alberga el algoritmo de búsquedas semánticas de sitios de interés el cual se comunica con la ontología y la base de datos de SITApp para seleccionar sitios de interés según la correlación entre la búsqueda del usuario y los sitios almacenados en la base de datos con

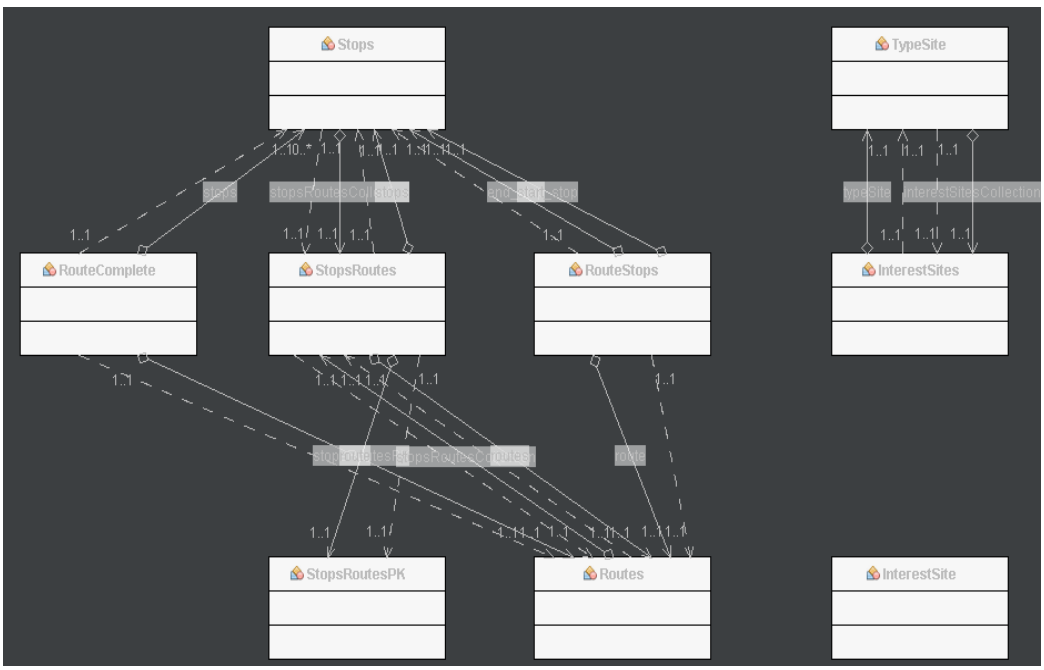
criterios tales como palabras clave, categorías o silimitud de palabras con el fin de eliminar errores de ortografía, estos resultados son enviados hacia SITApp Android. Adicionalmente, este aplicativo alberga también el algoritmo de cruce de rutas que mediante una posición de origen y una posición de destino, cada una de ellas debidamente referenciadas espacialmente, se evalúa que paraderos se encuentran a un radio máximo de 500 métros con respecto a las dos posiciones dadas por el usuario, con este listado de paraderos se cruza la información con los paraderos asociados a cáda una de las rutas y de esta manera se identifica que rutas le sirven al usuario y finalmente son ordenadas dándole prioridad a la ruta en la cual el usuario debe caminar menos. Para la comunicación con la ontología el web service utiliza la librería Jena, adicionalmente para mejorar las búsquedas utiliza la librería Lucene con el fin de suprimir palabras muertas. Está instalado en el servidor de aplicaciones Glassfish 4.1 bajo el sistema operativo Linux con la distribución Ubuntu Server 14.04.

Este aplicativo hace parte solamente del modelo y el controlador en el modelo vista controlador (MVC) de SITApp.

Implementación del modelo

La implementación del modelo del MVC de SITApp para el web service se realizó mediante el paquete model que se describe a continuación, en la figura 48 se muestra el diagrama de clases del modelo:

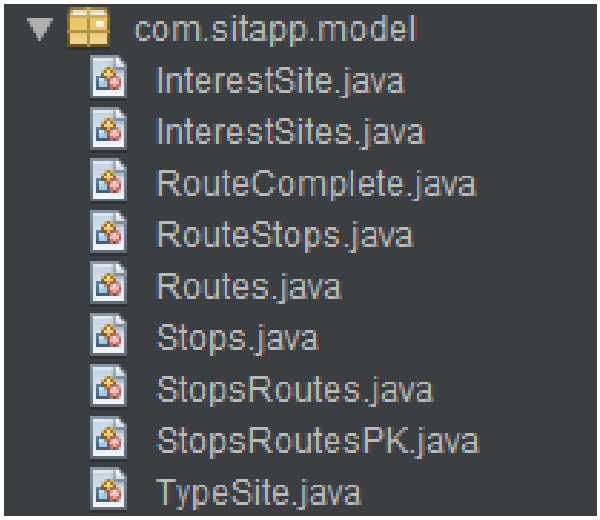
Figura 48. Diagrama de clases del paquete Model de WSSitapp



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo D Manual del programador de SITApp.

- **com.sitapp.model:** paquete encargado de la representación del modelo de la base de datos en clases Java. En la figura 49, se muestran las clases que componen este paquete.

Figura 49. Clases del paquete model del SITApp Web Service

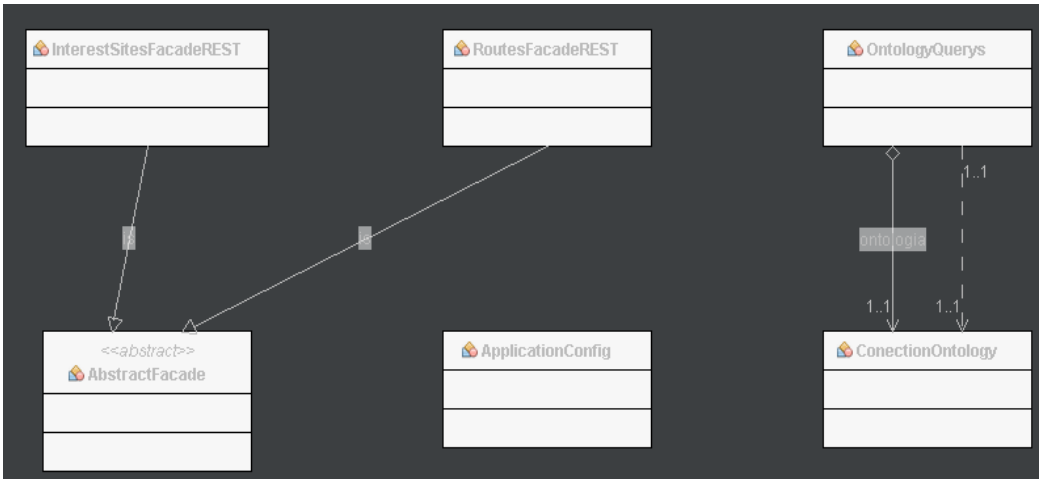


- ❖ **InterestSites.java:** clase que representa la tabla InterestSites, referencia a los sitios de interés del municipio de Pasto.
- ❖ **RouteComplete.java:** clase que representa a una ruta del SETP con todos sus paraderos autorizados asociados, personalizada con el fin de enviar solamente la información necesaria a SITApp Android.
- ❖ **RouteStops.java:** clase que representa a una ruta del SETP de Pasto, el paradero de origen y el paradero de destino para el usuario, personalizada con el fin de enviar solamente la información necesaria a SITApp Android.
- ❖ **Routes.java:** clase que representa la tabla Routes de la base de datos, referencia a las rutas del SETP de Pasto.
- ❖ **Stop.java:** clase que representa la tabla Stops de la base de datos, referencia a los paraderos autorizados del SETP de Pasto.
- ❖ **StopsRoutes.java:** clase que representa la tabla intermedia StopsRoutes de la base de datos.
- ❖ **StopsRoutesPK.java:** clase que representa la llave primaria de la tabla intermedia StopsRoutes de la base de datos.

Implementación del controlador

La implementación del controlador del MVC de SITApp para el web service se realizó mediante los paquetes ontology.conection, ontology.queries y services que se describen a continuación, en la figura 50, se muestra el diagrama de clases del controlador:

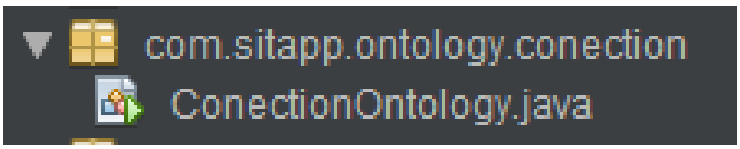
Figura 50. Diagrama de clases de los paquetes conection, quieries y service de SITApp Web Service.



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo D, Manual del programador de SITApp.

- **com.sitapp.ontology.conection:** paquete que alberga la clase encargada de realizar la conexión con la ontología (ver figura 51).

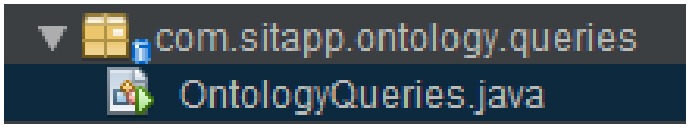
Figura 51. Clases del paquete ontology.conection del SITApp Web Service



  - ❖ **ConectionOntology.java:** clase que realiza la conexión con la ontología ubicada en el servidor usando la librería Jena.

- **com.sitapp.ontology.queries:** paquete encargado de realizar las consultas a la ontología, ver figura 52.

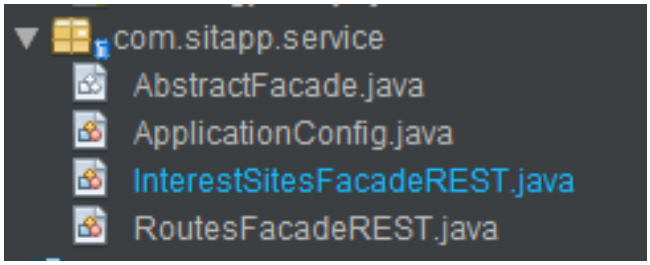
Figura 52. Clases del paquete ontolgy.queries de SITApp Web Service



- ❖ **OntologyQueries.java:** clase que realiza las consultas sobre la ontología para soportar la búsqueda semántica de sitios de interés de la ciudad de Pasto.

- **com.sitapp.service:** paquete encargado de soportar los servicios web requeridos por SITApp Android, ver figura 53.

Figura 53. Clases del paquete service de SITApp Web Service



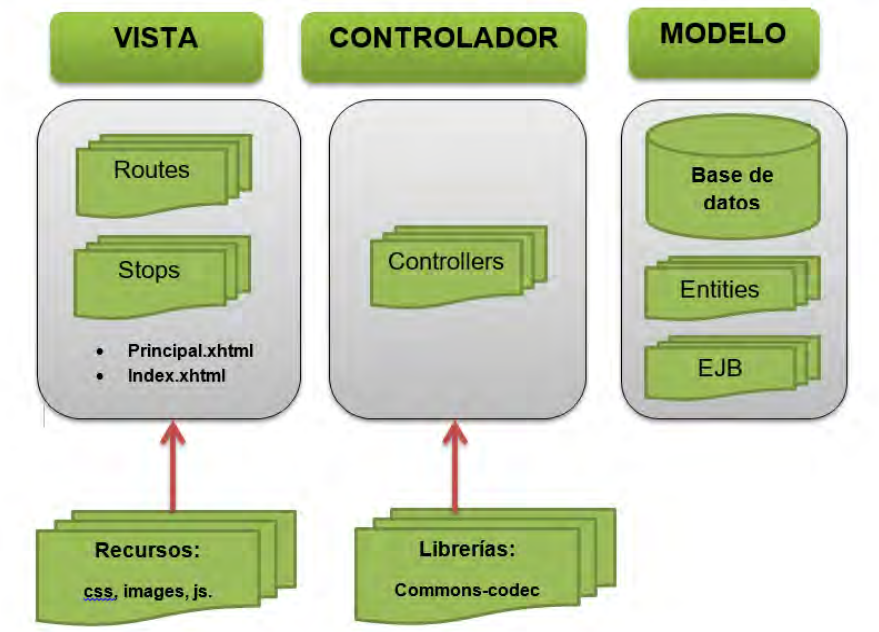
- ❖ **AbstractFacade.java:** esta clase es un tipo de patrón estructural que representa en general a los servicios web soportados en este paquete**.**
- ❖ **ApplicationConfig.java:** esta clase alberga las configuraciones generales de SITApp Web Service, indicando que clases del paquete service representan servicios web funcionales.
- ❖ **InterestSitesFacadeREST.java:** en esta clase se encuentra el algoritmo de búsqueda semántica de sitios de interés que se realiza mediante la ontología y posteriormente con la base de datos, esta clase utiliza al paquete ontology.conection y ontology.queries, además este service utiliza la librería Lucene para eliminar palabras muertas tales como conectores o conjunciones y mejorar el resultado de la búsqueda.
- ❖ **RouteFacadeREST.java:** en esta clase se encuentra el algoritmo de cruce de rutas necesario para encontrar la(s) ruta(s) del SETP que le sirven al usuario para ir desde un punto A hasta un punto B, además se encuentra el servicio que soporta la búsqueda mediante tipo de ruta ya sean rutas estratégicas o complementarias.

3.5. DISEÑO Y CONSTRUCCIÓN DEL MÓDULO ADMINISTRATIVO DE SITAPP

La construcción del módulo administrativo fue realizada en su totalidad bajo el sistema operativo Linux Mint 18.1 KDE Edition. El lenguaje de programación utilizado fue Java, en específico se hizo uso del framework Java Server faces (jsf), primefaces, javascript, css y omnifaces esta última es una librería para convertir datos primitivos (cadenas de texto, números) a objetos, el entorno de desarrollo empleado fue Netbeans versión 8.1. El sistema está montado sobre el servidor de aplicaciones Glassfish versión 4.1.

Este módulo está desarrollado bajo el modelo MVC (Model View Controller o Modelo Vista Controlador). En la Figura 54, se puede observar la arquitectura de este módulo.

Figura 54. Arquitectura módulo administrativo SITApp.

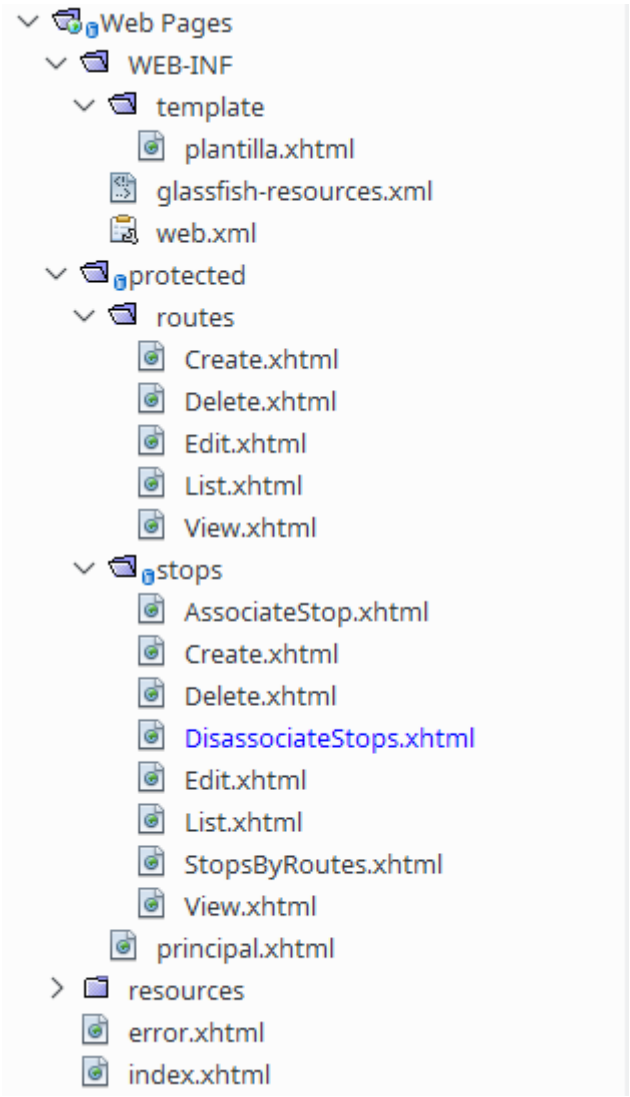


Implementación de la vista.

Permite administrar el sistema mediante una interfaz gráfica, el usuario administrador puede realizar operaciones como crear rutas o paraderos, asociar o desasociar paraderos, editar rutas y paraderos.

Cada uno de los archivos que se encuentran en este paquete tiene extensión .xhtml. En la Figura 55, se puede observar los elementos que conforman el paquete Vista.

Figura 55. Componentes del paquete Vista.



Como se puede observar en la figura anterior este paquete está formado por varias carpetas y archivos con extensión .xhtml o xml. A continuación se hace una pequeña descripción de los elementos más sobresalientes dentro de este paquete.

- **template**: dentro de esta carpeta se encuentra un archivo, plantilla.xhtml, el cual permite gestionar las peticiones del usuario de manera dinámica.
- **routes**: dentro de esta carpeta se encuentran los archivos, con extensión .xhtml, que permiten crear, modificar, ver, listar y eliminar rutas del SETP.
- **stops**: en esta carpeta están alojados los archivos, con extensión .xhtml, que permiten crear, modificar, ver, listar, eliminar, asociar y

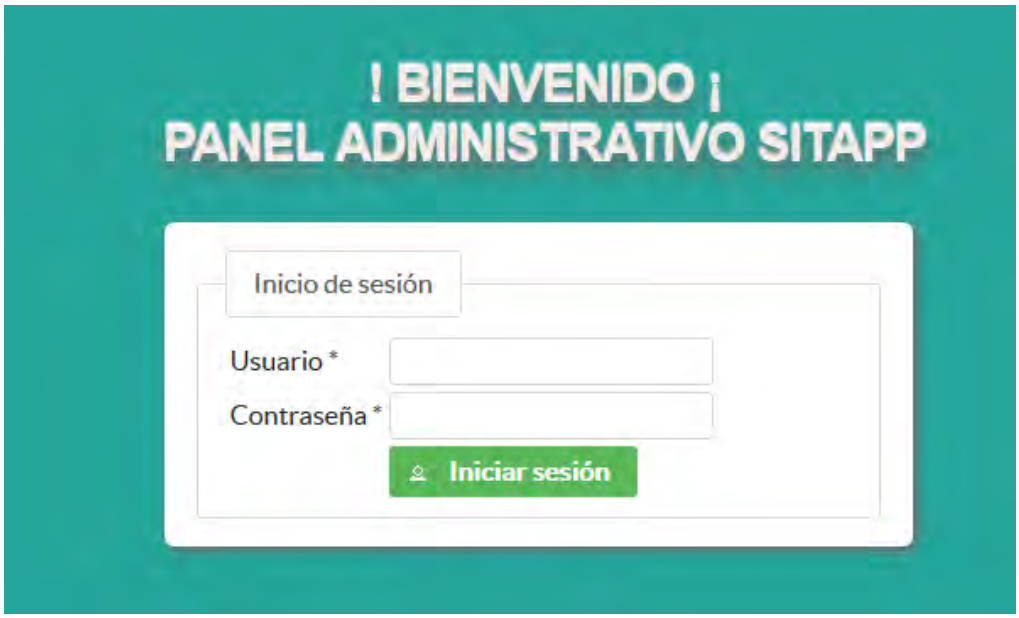
desasociar paraderos así como también ver los paraderos asociados a una determinada ruta.

- **principal.xhtml**: muestra la interfaz principal del módulo administrativo luego de que el usuario ha iniciado sesión.
- **error.xhtml**: el contenido de este archivo es desplegado en pantalla cuando el usuario intenta acceder al sistema sin haber iniciado sesión.
- **index.xhtml**: muestra la pantalla de inicio de sesión.

Además, este paquete hace uso de algunos recursos como son imágenes, hojas de estilo y archivos con extensión .js, elementos necesarios para una adecuada apariencia del aplicativo web.

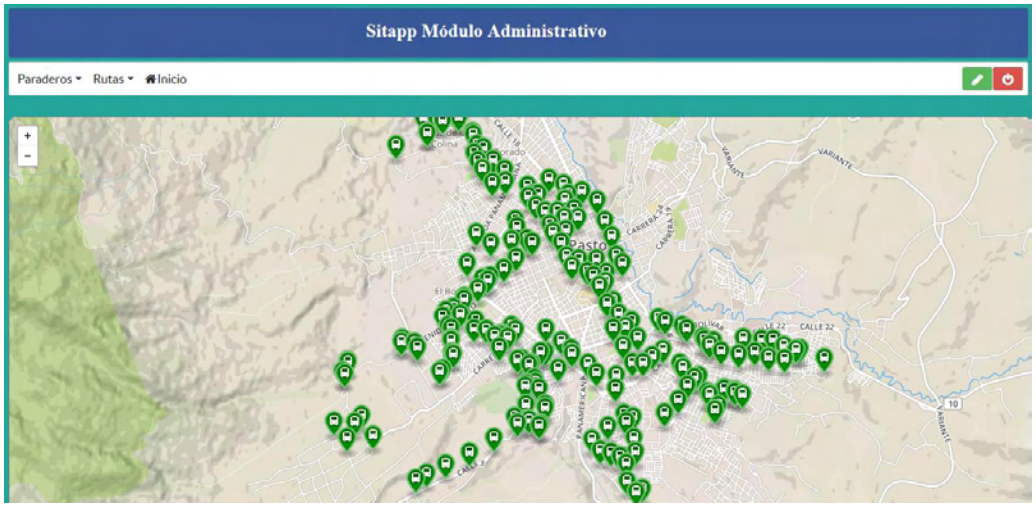
Al desplegar la aplicación, el usuario debe iniciar sesión (ver Figura 56) para poder realizar cambios sobre la base de datos, estos cambios una vez realizados son reflejados de manera inmediata en el aplicativo móvil.

Figura 56. Inicio de sesión módulo administrativo del aplicativo móvil.



Posterior al inicio de sesión se muestra la interfaz principal (ver Figura 57) del aplicativo web, desde esta interfaz el usuario administrador puede realizar las labores de actualización de la base de datos del aplicativo móvil.

Figura 57. Interfaz principal del módulo administrativo del aplicativo móvil.



Esta interfaz contiene una barra de menú con tres opciones Paraderos, Rutas e Inicio, además de dos botones en la parte derecha, los cuales permite cerrar sesión y actualizar la contraseña del usuario administrador.

La opción Paraderos permite realizar las siguientes operaciones.

- Eliminar Paradero.
- Crear Paradero.
- Ver y Editar.
- Asociar paraderos.
- Des-asociar paraderos.

La opción Rutas permite realizar las siguientes operaciones.

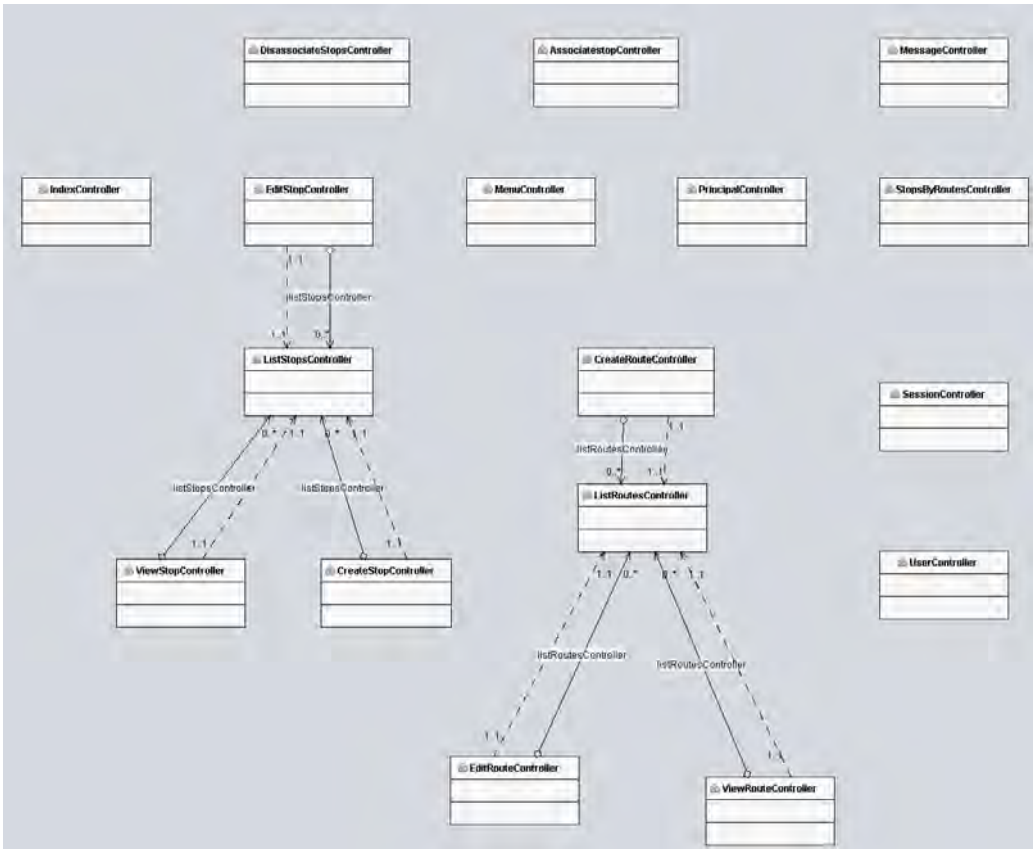
- Eliminar ruta.
- Crear ruta.
- Ver y Editar.
- Paraderos asociados.

La opción Inicio, permite regresar a la interfaz principal.

Implementación del controlador

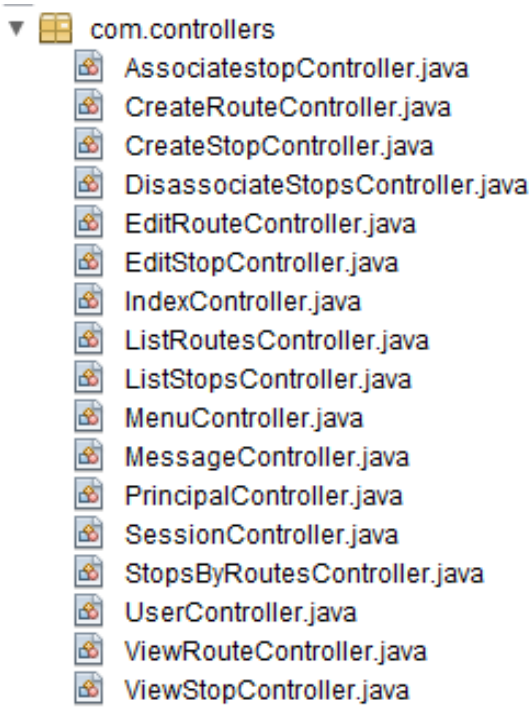
El controlador tiene como función servir de enlace entre la vista y el modelo, ver figura 58 (diagrama de clases paquete controlador). El controlador se conecta directamente con el paquete com.entities.ejb para así dar respuesta a las peticiones hechas, por el administrador, desde la vista. En la Figura 59 se puede observar las clases que componen este paquete.

Figura 58 Diagrama de clases paquete controlador.



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo E, Manual del programador Módulo Administrativo de SITApp.

Figura 59. Clases del paquete Controlador del módulo administrativo de SITApp.



Para el óptimo funcionamiento y división de archivos de acuerdo a sus responsabilidades, por cada archivo con extensión .xhtml, los cuales forman el paquete vista, se creó un archivo con extensión .java para que dé respuesta a las peticiones de la vista.

- **AssociateStopsController.java**: clase que permite asociar paraderos a una determinada ruta del SETP.
- **CreateRouteController.java**: clase que permite crear nuevas rutas en la base de datos del SETP.
- **CreateStopController.java**: clase encargada de crear nuevos paraderos en el repositorio de datos de SETP.
- **DisassociateStopsController.java**: clase que permite desasociar paraderos de una ruta determinada del SETP.
- **EditRouteController.java**: esta clase permite editar la información de una ruta del SETP.
- **EditStopController.java**: clase que permite modificar la información de un paradero del SETP.
- **IndexController.java**: clase encargada de gestionar el inicio de sesión del usuario administrador de SITApp.
- **ListRoutesController.java**: clase encargada de retornar un listado de todas las rutas del SETP, así como también de eliminar una ruta indicada.

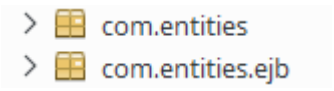
- **ListStopsController.java**: clase encargada de mostrar un listado de todos los paraderos y, si el administrador de SITApp lo desea puede eliminar una ruta del SETP.
- **PlantillaController.java**: clase encargada de gestionar el menú de opciones y mostrar dinámicamente el resultado de cada petición.
- **SessionController.java**: clase encargada de verificar que el usuario haya iniciado sesión para poder ingresar a las opciones del sistema.
- **StopsByRoutesController.java**: clase que muestra un listado de los paraderos asociados a una ruta en particular.
- **ViewRouteController.java**: clase encargada de mostrar información de una ruta en particular.
- **ViewStopController.java**: clase encargada de mostrar información detallada de un paradero en particular.

El controlador hace uso de una librería llamada Commons-codec la cual permite encriptar datos, la encriptación es utilizada el momento de iniciar sesión.

Implementación del modelo.

Dentro del modelo se encuentran las clases e interfaces, codificadas en el lenguaje de programación Java, que permiten realizar operaciones de inserción, lectura, actualización y eliminación de registros en la base de datos. Ver Figura 60.

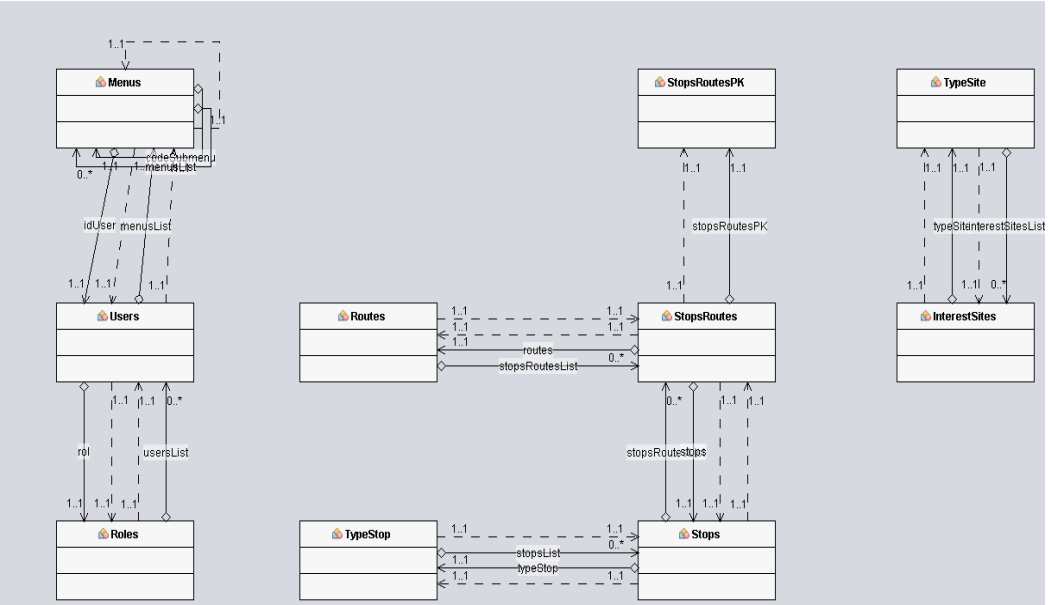
Figura 60. Paquete Modelo módulo administrativo SITApp.



Como se puede observar en la figura anterior, el modelo está formado por dos paquetes.

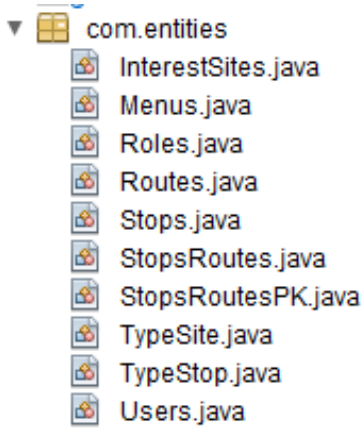
- El paquete **com.entities**, contiene el modelo de la base de datos representado en clases escritas en lenguaje de programación Java, ver figura 61 (diagrama de clases del paquete com.entities), las cuales emulan cada una de las tablas de la base de datos. El mapeo de las tablas, con todos sus atributos y restricciones, lo realiza el framework JPA (Java Persistence Api). Las clases que componen este paquete se muestran en la Figura 62.

Figura 61: Diagrama de clases paquete com.entities.



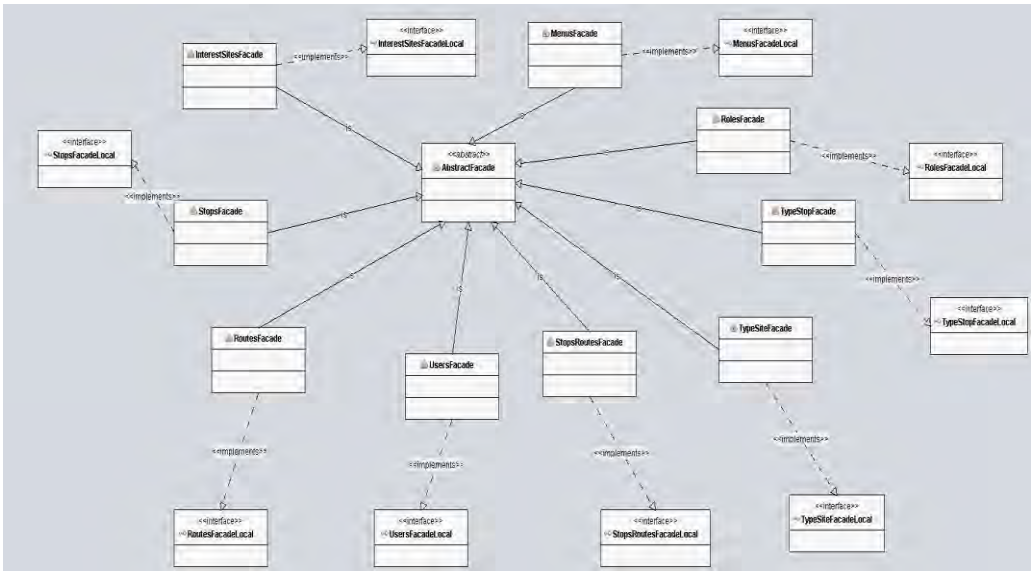
La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo E, Manual del programador Módulo Administrativo de SITApp.

Figura 62. Clases del paquete com.entities.



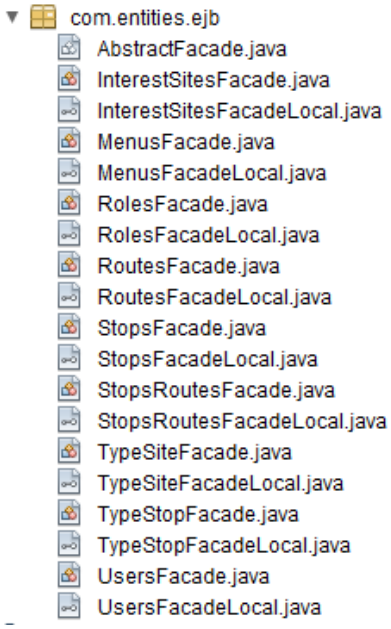
- El paquete **com.entities.ejb**, contiene un conjunto de clases que se encargan de realizar el acceso a los registros contenidos en la base de datos (ver figura 63, diagrama de clases del paquete com.entities.ejb), al momento de intentar realizar cualquier operación sobre la base de datos, insertar, leer, actualizar o eliminar registros, se debe hacer uso de este paquete, el cual a su vez hace uso del paquete com.entities. En la Figura 64, se puede ver las clases que forman este paquete.

Figura 63. Diagrama de clases paquete com.entities.ejb.



La especificación detallada de cada una de las clases con sus atributos y métodos se puede observar en el Anexo E, Manual del programador Módulo Administrativo de SITApp.

Figura 64. Clases del paquete com.entities.ejb



- **AbstractFacade.java**: clase padre, abstracta, que herada todos sus métodos a las demás clases, dentro de ella se encuentran los métodos que permiten realizar operaciones CRUD sobre cada una de las tablas de la base de datos.

- **InterestingSitesFacade.java**: es la clase que se encarga de permitir operaciones CRUD sobre la tabla interesting_sites.
- **MenusFacade.java**: clase encargada de mostrar el menú de opciones al usuario una vez que este ha iniciado sesión.
- **RoutesFacade.java**: clase encargada de realizar operaciones CRUD sobre la tabla routes.
- **StopsFacade.java**: esta clase tiene como objetivo realizar operaciones CRUD sobre la tabla stops.
- **StopsRoutesFacade.java**: esta clase, emula el enlace entre la clase StopsFacade.java y RoutesFacade.java, y es la encargada de las operaciones CRUD sobre la tabla stops_routes.
- **UserFacade.java**: clase encargada de gestionar la sesión del usuario administrador de SITApp.

Para evitar SQL Injection se hiso uso de consultas SQL preparadas, el Framework JPA (Java Persistence Api) permite crear consultas SQL a las cuales se les puede asignar parámetros pero el parámetro enviado debe ser válido, es decir, debe ser un número o una cadena de texto que cumpla con la restricción de longitud del campo al cual está siendo asignado el parámetro.

La descripción detallada de cada una de estas funcionalidades y la manera correcta de utilizar el módulo administrativo se encuentra registrado en el manual de usuario, el cual está en el Anexo E, Manual del Programador Módulo Administrativo de SITApp.

3.6. FASE 6. DEFINICIÓN Y EJECUCIÓN DE PRUEBAS

En esta fase se describen las pruebas aplicadas a SITApp y al módulo administrativo con el objetivo de verificar la calidad de los resultados de las búsquedas semánticas y la calidad de los datos recolectados. Se definieron 3 casos de prueba, clasificados como correcto o incorrecto, según el resultado obtenido. Además, se realizaron pruebas unitarias con el objetivo de verificar el correcto funcionamiento de los aplicativos desarrollados.

6.7.1. Casos de prueba

- **Caso de prueba 1 – Búsqueda de un sitio de interés:**

La búsqueda se realizó mediante SITApp Android, el objetivo de esta prueba es verificar el funcionamiento del buscador semántico implementado en SITApp. Se ingresaron búsquedas con errores ortográficos para verificar la calidad de los resultados de las búsquedas. Los resultados de esta prueba se presentan en la tabla 4.

Tabla 4. Caso de prueba 1, búsqueda de un sitio de interés

No.	Búsqueda prueba	Resultado	Calificación
1.	udenar	Universidad de Nariño, Universidad de Nariño Centro, Liceo de la Universidad de Nariño.	Correcto
2.	unimar	Universidad Mariana	Correcto
3.	ucc	Universidad Cooperativa de Colombia	Correcto
4.	unico	Centro Comercial Unico	Correcto.
5.	San adrecito	San Andresito, San Andres, San Francisco.	Correcto.
6.	Santigo	Santiago Apostol, Parque de Santiago.	Correcto.
7.	Otel cuelar	Hotel Cuellar's, Hotel Grecia, Hotel Palermo.	Correcto.
8.	Pisina	Piscina Semi-olímpica Aranda, Barrio pinos del norte, Pista de patinaje Obonuco.	Correcto.
9.	seani	Cehani	Correcto.
10.	La mserd	La Merced, Hotel Mardoll, Paso Libertadores.	Correcto.

- **Caso de prueba 2 – Búsqueda por dos sitios de interés**

La búsqueda se realizó mediante SITApp Android, el objetivo de esta prueba es verificar el funcionamiento del buscador semántico implementado en SITApp. Se ingresaron búsquedas con errores ortográficos para verificar la calidad de los resultados de las búsquedas. Los resultados de esta prueba se presentan en la tabla 5.

Tabla 5. Caso de prueba 2, búsqueda por dos sitios de interés

No.	Búsqueda prueba	Resultado	Calificación
1	tegar	Barrio el Tejar	Correcto.
2	briseno	Barrio Briceño	Correcto.
3	reminton	Remington	Correcto.
4	bonvona	Mercado Artesanal Bombona	Correcto.
5	Parqe narni	Plaza de Nariño, Parque Infantil Confamiliar	Correcto.
6	Palasa carnabal	Plaza de Carnaval	Correcto.
7	Muceo del horo	Museo del Oro	Correcto.
	Vario ajualongo	Barrio Agualongo, Barrio	

8		Gualcaloma, Barrio Salomon	Correcto.
9	merkado	Merka rapido, Mercado del Lorenzo, Mercado del potrerillo entrada sur.	Correcto.
10	unisento	Unicentro, Universidad Antonio Nariño, Luiseg	Correcto.

- **Caso de prueba 3 – Tocar dos puntos del mapa**

El objetivo de esta prueba es verificar que el app asocie a una ruta del SETP dos puntos del mapa teniendo como parámetro coordenadas geográficas. Los resultados de esta prueba se pueden apreciar en la tabla 6.

Tabla 6. Caso de prueba 3, tocar dos puntos en el mapa

No.	Coordenadas ingresadas	Ruta asociada.	Calificación
1	• 1.20537087, -77.2997644 • 1.20688608, -77.2924470	**C1**, OBONUCO – ALTOS DE CHAPALITO	Correcto.
2	• 1.19268762, -77.2730191 • 1.21840043, -77.2631403	**C12**, BRICEÑO - CAROLINA - ALCALDIA - BRICEÑO	Correcto.
3	• 1.21464846, -77.2782468	**C8**, CUJACAL - SAN MARTIN	Correcto.
4	• 1.23118792, -77.2938912	**E2**, UDENAR – ALTOS DE CHAPALITO	Correcto.
5	• 1.23372069, -77.2982203	**E5**, BRICEÑO - CATAMBUCO	Correcto.

6.7.2. Pruebas unitarias

Se plantearon una serie de pruebas para comprobar el correcto funcionamiento del app y el módulo administrativo. Las pruebas del web services se ven reflejadas en el app, puesto que todos los datos ingresados en el app se envían al servidor a través del web services y es el web services el que gestiona las peticiones y da respuesta a las mismas. Se realizaron una serie de preguntas con respuestas de SI o NO para verificar el correcto funcionamiento de las aplicaciones desarrolladas. Los resultados de estas pruebas se pueden ver en las tablas 7 a 13.

Búsqueda por un sitio de interés:

Tabla 7. Pruebas unitarias, búsqueda por un sitio de interés

Pruebas de aceptación	Descripción de la prueba	Resultado
Validación de datos	¿Se valida el ingreso de datos nulos?	SI
Búsqueda por un sitio de interés.	¿Se valida que el GPS este activo?	SI
	¿Se muestra mensaje de error cuando el GPS no está activo?	SI
	¿El app toma la posición actual del usuario?	SI
	¿Se muestra un icono que identifique la posición actual del usuario?	SI
	¿Se muestra en el mapa el sitio de interés ingresado por el usuario?	SI
	¿Se muestra información del tipo de sitio al hacer clic sobre el icono?	SI
	¿El sistema asocia la ubicación del usuario y el sitio de interés a una ruta del SETP?	SI
	¿El sistema muestra más de una opción de ruta, cuando existe?	SI
	¿El app muestra un mensaje cuando no existe una ruta?	SI
	¿El app permite distinguir los paraderos de inicio y fin de viaje?	SI

Búsqueda origen y destino:

Tabla 8. Pruebas unitarias, búsqueda origen y destino

Pruebas de aceptación	Descripción de la prueba	Resultado
Validación de datos	¿Se valida el ingreso de datos nulos?	SI.
	¿Se valida que el GPS este activo?	SI
	¿Se muestra mensaje de error cuando el GPS no está activo?	SI
	¿El app muestra un mensaje de ayuda para el usuario?	SI
	¿El app solicita el primer sitio de interés?	SI

Búsqueda por dos sitios de interés.	¿El mensaje de ayuda se actualiza luego de que el usuario ha ingresado el primer sito de interés?	SI
	¿El app ubica en el mapa los dos sitios de interés ingresados por el usuario?	SI
	¿El app asocia los dos sitios de interés a una ruta del SETP?	SI
	¿Cuándo no hay rutas que pasen cerca de los dos sitios de interés, el app muestra un mensaje?	SI
	¿Cuándo existe más de una ruta que pase cerca de los dos sitios de interés, el app las muestra?	SI
	¿El app permite distinguir los paraderos de inicio y fin de viaje?	SI
	¿El app permite cambiar el destino del viaje?	SI
	¿El app permite distinguir los paraderos de inicio y fin de viaje?	SI

Tocar dos puntos en el mapa:

Tabla 9. Pruebas unitarias, tocar dos puntos en el mapa

Pruebas de aceptación	**Descripción de la prueba**	**Resultado**
Validación de datos	¿Se valida el ingreso de datos nulos?	SI
Tocar dos puntos en el mapa.	¿El app muestra un mensaje de ayuda para el usuario?	SI
	¿El app muestra una ventana emergente para indicar al usuario donde inicia o termina el viaje?	
	¿El app ubica un marcador en el lugar donde inicia el viaje?	SI
	¿El app ubica un marcador en el lugar donde termina el viaje?	SI
	¿El app asocia los dos puntos a una ruta del SETP?	SI
	¿Cuándo existe más de una ruta, el app	

	muestra las diferentes opciones?	SI
	¿En el caso de no existir una ruta que pase cerca a los dos puntos, el app muestra un mensaje?	SI
	¿El app permite cambiar el inicio del viaje?	SI
	¿El app permite cambiar el destino del viaje?	SI
	¿El app permite distinguir los paraderos de inicio y fin de viaje?	SI

Inicio de sesión:

Tabla 10. Pruebas unitarias, inicio de sesión

Pruebas de aceptación	**Descripción de la prueba**	**Resultado**
Validación de datos	¿Se valida el ingreso de datos nulos?	SI
Validación inicio de sesión	¿Se muestra un mensaje de usuario o contraseña no válidos?	SI
	¿Se valida que no se muestra la contraseña cuando se está escribiendo?	SI
	¿Si los datos del usuario son correcto se re-direcciona a la página principal?	SI

Cierre de sesión:

Tabla 11. Pruebas unitarias, cierre de sesión

Pruebas de aceptación	**Descripción de la prueba**	**Resultado**
Cierre de sesión	¿Se muestra claramente la opción de cierre de sesión?	SI
	¿Se envía a la vista de inicio de sesión una vez se ha cerrado la sesión?	SI
	¿Se deniega el acceso a las opciones del panel administrativo si no se ha iniciado sesión?	SI
	¿Se muestra la vista de permisos no válidos para los usuarios que quieren acceder al sistema sin haber iniciado sesión?	SI

Paraderos:

Tabla 12. Pruebas unitarias, paraderos

Pruebas de aceptación	Descripción de la prueba	Resultado
Eliminación de Paraderos	¿Se muestra un listado de todos los paraderos disponibles?	SI
	¿Se muestra información detallada del paradero que se va a eliminar?	SI
	¿Se muestra un mensaje de paradero eliminado exitosamente?	SI
Creación de Paraderos	¿Se valida que todos los campos estén diligenciados?	SI
	¿Se valida el tipo de dato para cada campo?	SI
	¿Se valida que el usuario ubique el paradero en el mapa?	SI
	¿Se muestra una ventana emergente con los datos del nuevo paradero?	SI
	¿Se muestra un mensaje de paradero creado exitosamente?	SI
Ver o editar paradero	¿Se muestra un listado de todos los paraderos disponibles?	SI
	¿Se muestra información detallada del paradero seleccionado?	SI
	¿Se muestra la ubicación del paradero en una cartografía de la ciudad de Pasto?	SI
	¿El usuario puede arrastrar el marcador para actualizar la ubicación del paradero?	SI
	¿El usuario puede editar toda la información del paradero?	SI
	¿Se muestra mensaje de paradero actualizado exitosamente?	SI
Asociar paraderos	¿Se muestra una lista de las rutas disponibles?	SI
	¿Se muestra un listado de todos los paraderos que no están asociados a la ruta seleccionada?	SI
	¿Se ubican los paraderos de la ruta seleccionada en una cartografía de la ciudad de Pasto?	SI
	¿Se ubica el paradero, candidato a ser asociado, en una cartografía de la ciudad de Pasto?	SI
	¿Se valida que el paradero, candidato a ser asociado, no este a una distancia superior	SI

	a 22 metros?	
	¿Se valida que el paradero tenga un orden lógico?	SI
	¿Se valida el ingreso de datos nulos?	SI
	¿Se muestra mensaje de paradero asociado con éxito?	SI
Desasociar paraderos	¿Se muestra un listado de todas las rutas?	SI
	¿Se muestran, en una cartografía de la ciudad, todos los paraderos de la ruta seleccionada?	SI
	¿Se resalta el paradero seleccionado para ser desasociado?	SI
	¿Se muestra un mensaje de paradero desasociado con éxito?	SI

Rutas:

Tabla 13. Pruebas unitarias, rutas

Pruebas de aceptación	**Descripción de la prueba**	**Resultado**
Eliminación de rutas	¿Se muestra un listado de todas las rutas disponibles?	SI
	¿Se muestra una ventana emergente con información de la ruta seleccionada para ser eliminada?	SI
	¿Se puede eliminar varias rutas al tiempo?	NO
	¿Se muestra un mensaje de ruta eliminada exitosamente?	SI
Creación de rutas	¿Se valida el ingreso de datos nulos?	SI
	¿Se valida que el usuario trace la ruta en el mapa?	SI
	¿Se muestra un mensaje de ruta creada exitosamente?	SI
Ver o editar ruta	¿Se muestra un listado de todas las rutas disponibles?	SI
	¿Se muestra información detallada de la ruta seleccionada?	SI
	¿Se muestra el recorrido de la ruta en una cartografía de la ciudad de Pasto?	SI
	¿El usuario puede editar toda la información, incluido el recorrido, de la ruta seleccionada?	SI

	¿Se muestra mensaje de ruta editada exitosamente?	SI
Paraderos asociados	¿Se muestra un listado de todas las rutas disponibles?	SI
	¿Se muestra el recorrido de la ruta y sus paraderos en una cartografía de la ciudad de Pasto?	SI
	¿Se muestra información del paradero al hacer clic sobre el marcador?	SI
	¿Se muestra un listado de todos los paraderos asociados a la ruta seleccionada?	SI

4. CONCLUSIONES

- ✓ Con la finalización de este proyecto de investigación se obtiene SITApp: una aplicación inteligente para el sistema estratégico de transporte público de la ciudad de Pasto. La ciudadanía de Pasto y los visitantes cuentan con esta aplicación gratuita como un medio y guía muy importante para agilizar su movilidad en la ciudad a través del transporte público debido a las distintas utilidades y funciones que brinda SITApp tales como realizar búsquedas inteligentes de sitios de interés de la ciudad y asociarlos a una o más rutas del SETP o indicar la ruta que debe seguir el usuario hasta llegar al punto en el cual debe esperar el bus.
- ✓ Para poder realizar las búsquedas inteligentes fue necesario la construcción de un WebService dado que la información existente en la base de datos debe ser tratada para lograr mostrar los resultados de las consultas realizadas por los usuarios. El procesamiento de información es bastante complejo y difícil de procesar por los equipos móviles, de ahí la necesidad de crear un mecanismo de comunicación entre los datos georreferenciados y el aplicativo móvil.
- ✓ El procesamiento de la información georreferenciada presente en la base de datos se logró realizar gracias al uso de un complemento para PostgreSQL llamado Postgis, el cual permite realizar operaciones sobre datos geográficos. Gracias al uso de este complemento fue posible dar respuesta a las consultas espaciales de la ciudad y poder asociar los resultados de la búsqueda a una o más rutas del SETP.
- ✓ Para la realización de búsquedas inteligentes fue necesaria la implementación de una ontología, la cual permite brindar a los usuarios resultados más acordes a las consultas realizadas mediante el uso de semántica asociada a las palabras claves de la búsqueda que las relaciona con la información almacenada en la base de datos espacial de SITApp.
- ✓ Una aplicación desarrollada bajo un modelo de arquitectura cliente servidor requiere de un módulo administrativo que permita mantener la información actualizada y brinde un control sobre la misma, haciendo los procesos de actualización de datos transparentes para el usuario del aplicativo móvil que consume los servicios alojados en el servidor, garantizando la sostenibilidad del proyecto.
- ✓ El uso de tecnologías libres reduce significativamente los costos de desarrollo de un proyecto software, ya que en la mayoría de los casos las librerías y recursos son totalmente gratuitos y además poseen buena documentación soportada por una gran comunidad, en algunos casos estos proyectos poseen licencias que permiten que empresas que no deseen liberar los códigos de sus proyectos los cuales son derivados de proyectos de código abierto, lo puedan hacer.

- ✓ La apropiación y aplicación de conocimientos adquiridos durante el periodo de formación académica dentro de la Universidad de Nariño y los procesos de formación como investigadores en el grupo de investigación GRIAS permitió que los autores del presente documento tuvieran un acercamiento al proceso investigativo y el descubrimiento de nuevo conocimiento
- ✓ El aplicativo móvil actualmente cuenta con las características propias de una aplicación de calidad. Con la intención de lograr una mayor utilidad para la aplicación móvil se plantea como trabajo futuro la construcción de un sistema de control de flota, con la finalidad de brindar mejores y más informativas respuestas a las consultas realizadas por el usuario.
- ✓ SITApp es la primera aplicación, para dispositivos móviles Android, de código abierto en la ciudad de Pasto. De esta manera el proyecto puede recibir aportes de la comunidad de desarrolladores. Para facilitar esta tarea y la mantenibilidad de los aplicativos fue necesaria la creación de un manual de programador por cada uno de ellos.

5. RECOMENDACIONES

- ✓ Trabajar mancomunadamente con Avante SETP para la continua actualización de la información espacial presente en la base de datos de SITApp haciendo que el proyecto sea sostenible a lo largo del tiempo brindando información precisa a los usuarios del sistema de transporte público.
- ✓ Difundir la existencia de esta aplicación en las redes sociales y en los medios escritos, radiales y televisivos de la Universidad de Nariño con el fin de fomentar el uso de esta aplicación por parte de la ciudadania del municipio de Pasto y de los estudiantes de la Universidad de Nariño.
- ✓ Articular a la Universidad de Nariño con los entes gubernamentales y municipales para el apoyo de procesos investigativos más acordes a las necesidades de la ciudadanía y de esta manera fomentar e incentivar la investigación y así fortalecer con innovación el desarrollo tecnológico y productivo de la ciudad de Pasto.
- ✓ Contratar un servidor dedicado con alta disponibilidad y rendimiento para que la aplicación no tenga interrupciones en el servicio prestado a la comunidad.
- ✓ Continuar con el proceso investigativo de manera tal que se desarrolle un aplicativo para hacer posible la actualización de la ontología haciendo uso de una interfaz gráfica de usuario acoplada con el módulo administrativo de SITApp.
- ✓ Hacer uso de la información georreferenciada que envían los usuarios hacia SITApp para realizar procesos de minería de datos y así identificar ciertos patrones de comportamiento de quienes usan el sistema de transporte público.
- ✓ Fomentar la inclusión de estudiantes investigadores y acoplarlos a los proyectos de investigación vigentes o ya terminados para su consecución haciéndolos partícipes del proceso de creación de conocimiento, tecnologías e innovación que se realizan en el grupo de investigación GRIAS.
- ✓ Realizar publicidad para informar que el código fuente del aplicativo SITApp fue liberado con el fin de obtener contribuciones por parte de la comunidad de desarrolladores.

BIBLIOGRAFÍA

[1] Google, «Transit – Google Maps». [En línea]. Disponible en: http://maps.google.com/landing/transit/index.html. [Accedido: 24-sep-2017].

[2] «Moovit». [En línea]. Disponible en: https://moovitapp.zendesk.com/hc/es/categories/201382629-Sobre-Moovit. [Accedido: 24-sep-2017].

[3] F. Parra, «moviliXa - Red de aplicaciones móviles para el transporte público», 2016. [En línea]. Disponible en: http://movilixa.com/. [Accedido: 24-sep-2017].

[4] C. Pang Lo y A. K.W. Yeung, *Concepts and Techniques of Geographic Information Systems*. 2002.

[5] Mine Action Information Center – James Madison University, «What is a GIS?» [En línea]. Disponible en: https://www.jmu.edu/cisr/research/sic/gis/main.htm. [Accedido: 24-sep-2017].

[6] V. Olaya, *Sistemas de Información Geográfica*. 2014.

[7] N. R. Council, *Learning to Think Spatially*. Washington, D.C.: National Academies Press, 2006.

[8] A. P. Navarro, *Introducción a los sistemas de información geográfica y geotelemática*. 2011.

[9] P. Snaider, «PROYECCIONES CARTOGRÁFICAS y SISTEMAS DE REFERENCIA», *Rev. Geográfica Digit. IGUNNE. Fac. Humanidades. UNNE. Año 7 N^{o} 13 Enero – Junio*, vol. 7, n.^{o} 24, 2010.

[10] «Proyecciones Cartográficas». [En línea]. Disponible en: http://profrnoe.blogspot.es/categoria/bloque-1/. [Accedido: 15-nov-2017].

[11] «Georreferenciación y sistemas de coordenadas | ArcGIS Resource Center». [En línea]. Disponible en: http://resources.arcgis.com/es/help/getting-started/articles/026n0000000s000000.htm. [Accedido: 15-nov-2017].

[12] Instituto internacional Español de Marketing Digital, «QUE ES GEOLOCALIZACION - Definición y características - IIEMD». [En línea]. Disponible en: https://iiemd.com/geolocalizacion/que-es-geolocalizacion. [Accedido: 24-sep-2017].

[13] T. Giménez Rodríguez y M. E. Ros Bernabeu, *SISTEMA DE POSICIONAMIENTO GLOBAL (GPS)*. 2010.

[14] «Triangulacion - Imagen». [En línea]. Disponible en: http://gpstotalcr.com/wp-content/uploads/2014/08/triangulacion-1024x650.jpg. [Accedido: 15-nov-2017].

[15] V. Bembibre, «Base de datos», 2009. [En línea]. Disponible en: https://www.definicionabc.com/tecnologia/base-de-datos.php. [Accedido: 24-sep-2017].

[16] C. M. Bender, C. Deco, J. S. González, M. Hallo, y J. C. P. Gallegos, *Tópicos avanzados de Bases de datos*. 2014.

[17] «EDUCACIÓN GEOGRÁFICA DAYANA». [En línea]. Disponible en: http://dayanaheros19.blogspot.com.co/2015/03/. [Accedido: 15-nov-2017].

[18] Telecentro Regional en Tecnologías Geoespaciales, «Fundamentos SIG». [En línea]. Disponible en: http://geoservice.igac.gov.co/contenidos_telecentro/fundamentos_sig/cursos/sem_2/uni2/index.php?id=2. [Accedido: 24-sep-2017].

[19] M. Instituto Nacional de Estadística y geografía, «BASE DE DATOS GEOGRÁFICOS, MODELO DE DATOS VECTORIALES». [En línea]. Disponible en: http://www.inegi.org.mx/geo/contenidos/topografia/doc/mod_datos_vec.pdf. [Accedido: 24-sep-2017].

[20] ArcGIS, «¿Qué son los datos ráster?» [En línea]. Disponible en: http://desktop.arcgis.com/es/arcmap/10.3/manage-data/raster-and-images/what-is-raster-data.htm. [Accedido: 24-sep-2017].

[21] «La definición de la geocodificación». [En línea]. Disponible en: http://www.wminformatica.com/g4ggMXO4w/. [Accedido: 15-nov-2017].

[22] P. Castells, «La web semántica».

[23] M. Camacho Rodriguez, «Incorporación de un buscador semántico en la plataforma LdShake para la selección de patrones educativos», ESCUELA SUPERIOR POLITÉCNICA UPF, 2013.

[24] R. Rodríguez García y M. L. Zayas De Diego, «Buscadores Web y Redes Semánticas».

[25] A. Lozano Tello, «Ontología en la web semántica», *Cuad. Investig. en Ing. Informática*, vol. 5, 2001.

[26] D. Lázaro, «Introducción a los Web Services». [En línea]. Disponible en: https://diego.com.es/introduccion-a-los-web-services. [Accedido: 24-sep-2017].

[27] «Origen de las aplicaciones, Aplicaciones móviles». [En línea]. Disponible en: https://appsmovilescavucm.wordpress.com/origen/. [Accedido: 24-sep-2017].

[28] «Avante SETP, Sistema Estratégico de Transporte Público de Pasto». [En línea]. Disponible en: https://www.avante.gov.co/. [Accedido: 21-ene-2018].

[29] D. R. Gallo Eraso, W. A. Botina Escobar, y Y. E. Estrada Bastidas, «RIKHUNA: VISOR CARTOGRÁFICO INTELIGENTE DE DIRECCIONES URBANAS Y SITIOS DE INTERÉS DEL MUNICIPIO DE PASTO BASADO EN POSTGIS», 2017.

[30] «WGS 84 - Spatial Reference». [En línea]. Disponible en: http://spatialreference.org/ref/epsg/wgs-84/. [Accedido: 24-sep-2017].

[31] «Welcome to Python.org». [En línea]. Disponible en: https://www.python.org/. [Accedido: 24-sep-2017].

[32] R. T. Pereira, G. Hernández Garzón, y N. Q. Taimbud, «GEOCODIFICADOR DE EVENTOS DELICTIVOS GEORREFERENCIADOS A NIVEL DE DIRECCIONES URBANAS EN EL MUNICIPIO DE PASTO».

[33] «PostgreSQL: The world’s most advanced open source database». [En línea]. Disponible en: https://www.postgresql.org/. [Accedido: 24-sep-2017].

[34] «PostGIS — Spatial and Geographic Objects for PostgreSQL». [En línea]. Disponible en: http://postgis.net/. [Accedido: 24-sep-2017].

[35] «Open Street Maps». [En línea]. Disponible en: https://wiki.openstreetmap.org/wiki/ES:Página_principal. [Accedido: 31-ene-2018].

[36] O. Corcho, M. Fernández-López, A. Gómez-Pérez, y A. López-Cima, «Construcción de ontologías legales con la metodología METHONTOLOGY y la herramienta WebODE».

[37] «protégé». [En línea]. Disponible en: https://protege.stanford.edu/. [Accedido: 20-ene-2018].

[38] «Android». [En línea]. Disponible en: https://www.android.com/. [Accedido: 24-sep-2017].

[39] «Java». [En línea]. Disponible en: https://www.java.com/es/. [Accedido: 24-sep-2017].

ANEXOS

ANEXO A. Manual de instalación de SITApp

Para poder instalar SITApp Android, se requiere un Smartphone que posea una versión de sistema operativo Android igual o superior a 4.1.

Desde el dispositivo se accede a la Google Play Store y se digita en la barra de búsqueda el nombre de la aplicación "**SITApp**" como se muestra en la Figura 1. Se accede a la ficha de la aplicación y se procede a instalar como se muestra en las Figuras 2 y 3.

Figura 1. Búsqueda de SITApp en la Google Play Store.

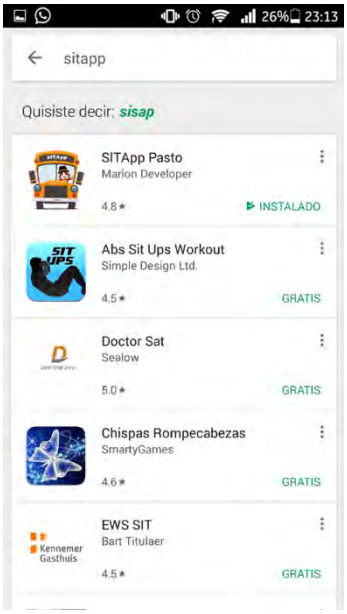


Figura 2. Ficha de SITApp en Google Play Store.

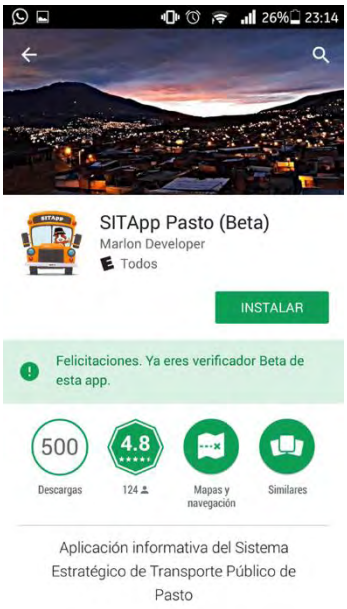
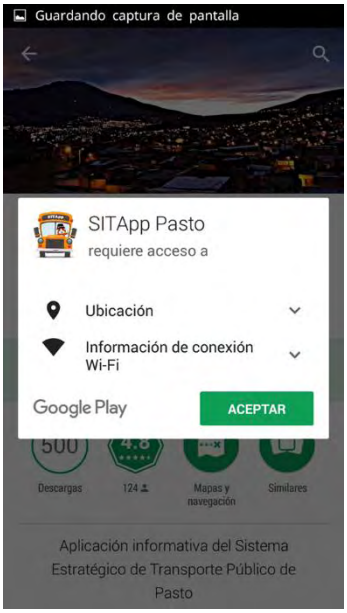


Figura 3. Confirmación de instalación de SITApp.



Para poder instalar SITApp Web Service y el Módulo administrativo de SITApp es necesario instalar previamente PostgreSQL 9.4, las extensión para PostgreSQL llamadas pg_similarity y PostGIS, el servidor de aplicaciones

Glassfish 4.1. Las siguientes instrucciones están diseñadas para ejecutarse bajo sistemas operativos GNU/Linux.

1. Instalación PostgreSQL 9.4

Para la instalación de PostgreSQL 9.4 y PostGIS en distribuciones basadas en Debian/Linux, en la terminal se ejecuta lo siguiente:

```
# apt-get install postgresql postgresql-client pgadmin3 postgresql-server-dev-9.4 postgresql-contrib postgresql-9.4-postgis-2.2
```

Ahora se borra la contraseña para la cuenta de admisitrador de “postgres”, para ello se ejecuta lo siguiente en la consola:

```
# su postgres –c psql template1
template1=# ALTER USER postgres WITH PASSWORD ‘postgres1’;
template1=# \q
```

Esto altera la contraseña dentro de la base de datos, ahora se debe realizar lo mismo pero para el usuario ‘postgres’ del sistema operativo y usar la misma contraseña:

```
# passwd –d postgres
# su postgres –c passwd
```

Ahora para instalar pg_similarity primero se descarga desde Github desde la siguiente dirección:

```
$ git clone https://github.com/eulerto/pg_similarity.git
```

Luego compilamos la extensión y la instalamos dentro de la base de datos de SITApp, adicionalmente también añadiremos la extensión PostGIS:

```
# cd pg_similarity
# USE_PGXS=1 make
# USE_PGXS=1 make install
# su postgres
$ createdb sitapp
$ psql -f pg_similarity.sql sitapp
$ psql sitapp
sitapp=# LOAD 'pg_similarity';
sitapp=# CREATE EXTENSION postgis;
sitapp=# \q
```

Luego se restaura la copia de la base de datos sitapp.sql:

```
$ psql –f /<ruta_archivo>/sitapp.sql sitapp
```

2. Instalación Glassfis 4.1

Primero se debe descargar la versión 4.1 para GNU/Linux desde el siguiente enlace: https://javaee.github.io/glassfish/download, este archivo se debe descomprimir en la carpeta donde se desea instalar el servidor, generalmente en **/opt/**, una vez descomprimido se inicia el servidor:

```
# cd /opt/glassfish4/glassfish/bin
# ./asadmin start-domain domain1
```

Para acceder al servidor utilizamos la ruta http://localhost:8080 y para acceder a la consola de administración utilizamos la ruta https://localhost:4848.

2.1. Instalación de SITApp Web Service

Para la instalación de SITApp Web Service primero se debe crear el pool de conexiones a la base de datos en el servidor de aplicaciones Glassfish en el menú “Resources/JDBC/JDBC Conections Pools” con los datos que se muestran en las Figuras 4 y 5:

Figura 4. Creación del pool de conexión, paso 1.

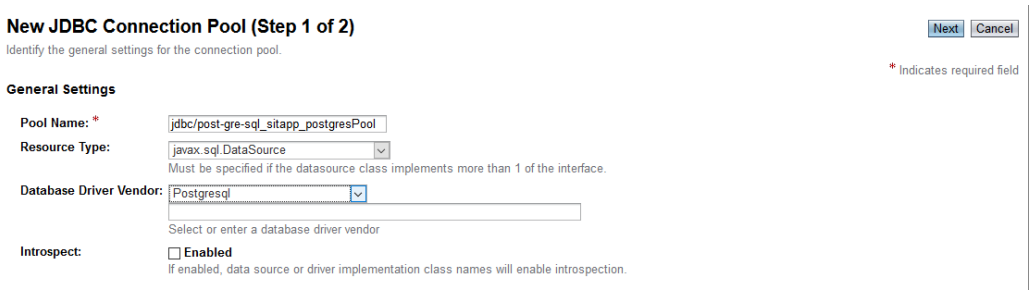
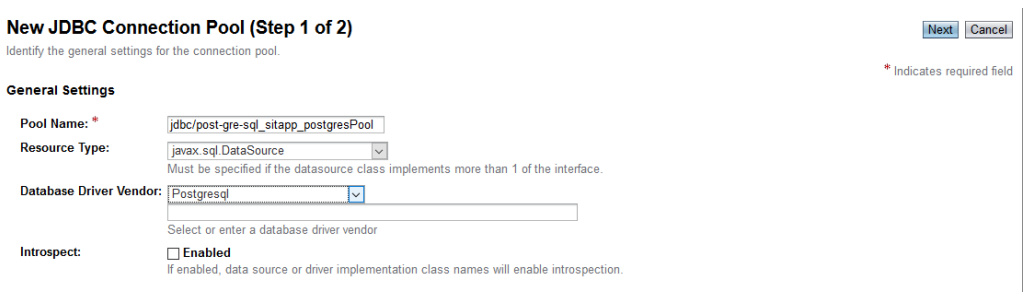
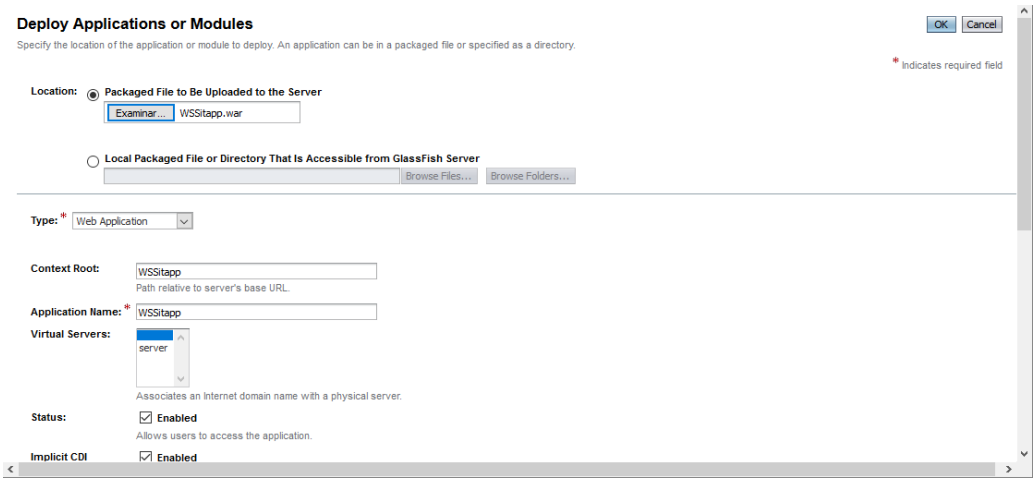


Figura 5. Creación del pool de conexión, paso 2.



Ahora procedeos a cargar el compilado de SITapp Web Service (.war) como se muestra en la Figura 6:

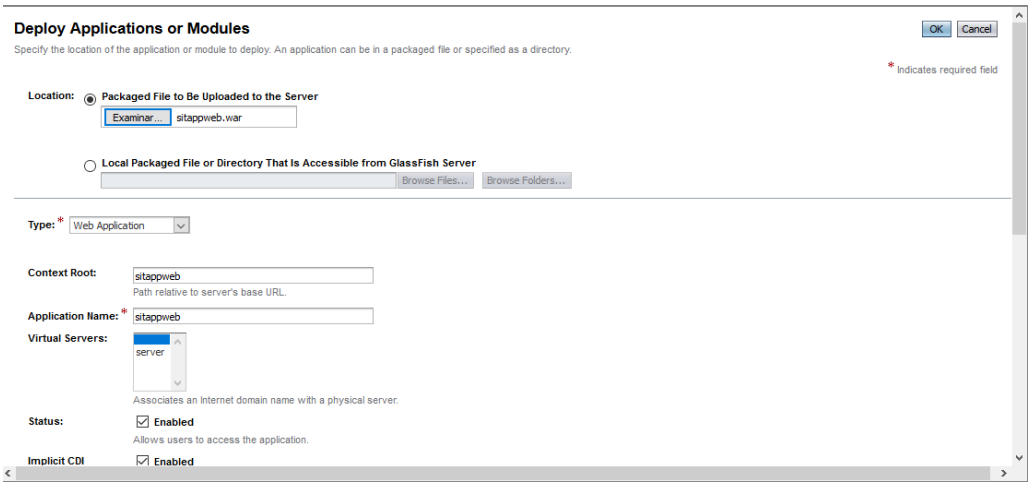
Figura 6. Despliegue de SITApp.



2.2. **Instalación del Módulo Administrativo de SITApp**

Para la instalación del módulo administrativo de SITApp solamente necesitamos cargar el compilado del módulo en el servidor de aplicaciones Glassfish como se muestra en la Figura 7:

Figura 7. Despliegue del Módulo Administrativo de SITApp



ANEXO B. Manual de usuario final SITApp

Este documento pretende guiar a los usuarios finales en el uso e interacción de la aplicación móvil denominada "**SITApp**: una aplicación inteligente para dispositivos móviles del sistema estratégico de transporte público del municipio de pasto". Esta app tiene como objetivo asociar sitios de interés de la ciudad de Pasto a una determinada ruta del Sistema Estratégico de Transporte Público del municipio de Pasto (SETP).

3. Interfaz principal

En la Figura 1, se puede observar la interfaz principal del aplicativo móvil. Esta consta de varios elementos. Un menú desplegable (número 1), una barra de búsqueda (número 2) y una sección de mensajes (número 3). El sistema solicita activar el GPS y dar permisos de acceso a la ubicación del equipo puesto que la función principal del app hace uso del GPS para ubicar una ruta desde la ubicación del usuario hasta el sitio digitado sobre la barra de búsqueda.

El app hace uso de sitios de interés de la ciudad; un sitio de interés puede ser clínicas, hospitales, hoteles, centros comerciales, museos, bancos, parques, etc. Cabe destacar que se encuentran almacenados los sitios más relevantes de la ciudad.

Consideraciones, el sistema hace uso de paraderos autorizados por el SETP del municipio de Pasto, así que el app muestra el camino que el usuario debe hacer para llegar al paradero de inicio y además muestra el camino que el usuario debe recorrer para llegar a su destino desde el paradero de fin de viaje. En ocasiones la ruta seleccionada pasa justo por el sitio destino pero el sistema sugiere abandonar el bus antes o después del sitio destino esto es debido a que en el sitio destino no hay un paradero autorizado.

Figura 1. Interfaz principal SITApp.

Aplicación en ejecución.

A continuación se describen cada uno de los elementos de la interfaz principal.

4. Sección mensajes

Esta parte de la interfaz es utilizada para brindar al usuario final una pequeña ayuda en cada una de las funciones que tiene el aplicativo. Cada vez que el usuario selecciona una de las opciones disponibles en el menú desplegable la sección mensajes se actualiza y muestra al usuario una ayuda referente a la opción seleccionada. Además esta sección es utilizada para mostrar información acerca de la ruta seleccionada para realizar el viaje en bus, ver Figura 2.

Figura 2: sección mensajes del aplicativo móvil.

Aplicación en ejecución.

5. **Barra de búsqueda**

Este elemento es utilizado para escribir el sitio de interés de la ciudad de Pasto. Esta sección hace que el aplicativo sea inteligente puesto que la barra de búsqueda, ver figura 3, hace uso de semántica para realizar las búsquedas en la base de datos, es decir, aun cuando existan errores de ortografía, el sistema intentará dar respuesta a las diferentes consultas realizadas por el usuario.

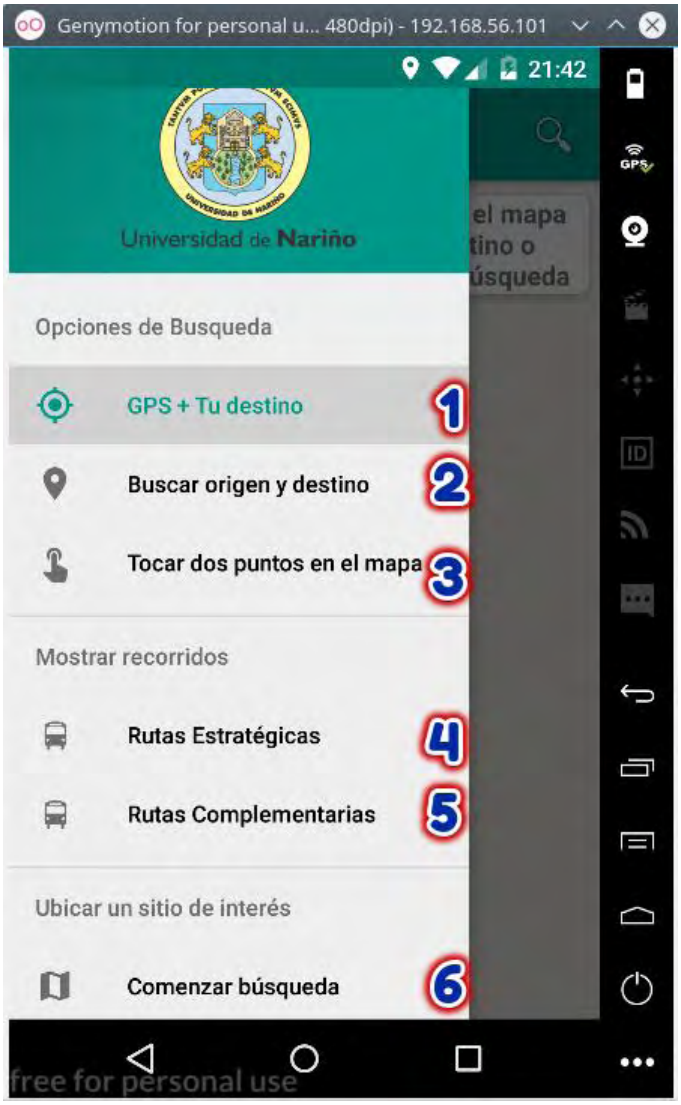
Figura 3: Barra de búsqueda del aplicativo SITApp.

Aplicación en ejecución.

6. Menú desplegable

Este elemento despliega una sección desde la cual el usuario puede hacer uso de las funciones con las cuales cuenta SITApp, ver figura 2.

Figura 4: Menú desplegable SITApp.



Aplicación en ejecución.

A continuación se describen cada una de las funcionalidades que tiene SITApp.

6.1. **GPS + Tu destino**

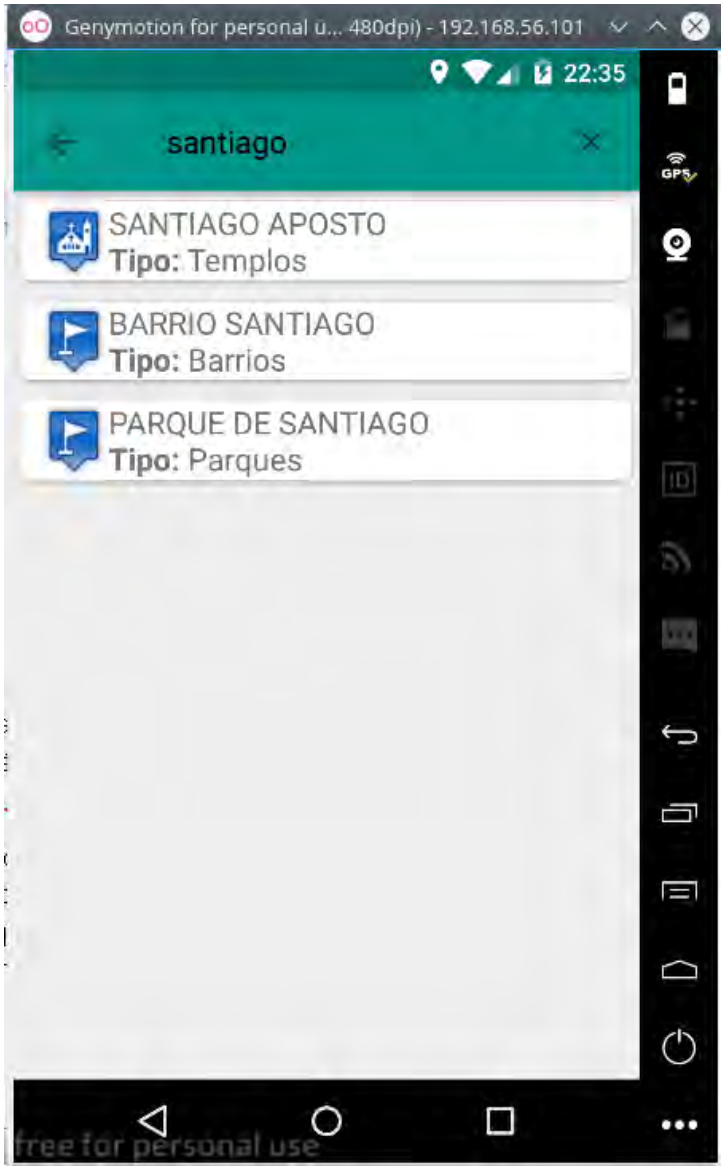
Función principal del app y además es la función que está activa por defecto, es decir, cada vez que se abre el app, la función GPS + TU DESTINO está activa. Esta función hace uso del GPS del dispositivo desde el cual se está haciendo uso del aplicativo. El app utiliza el GPS y toma como punto de partida la ubicación del teléfono, el usuario debe escribir, en la barra de búsqueda, el sitio al cual quiere llegar o puede tocar el mapa unos segundos y el sistema realiza la búsqueda de rutas que pasen cerca del sitio donde se encuentra el usuario y que además pasen cerca del sitio destino, digitado o pulsado por el usuario, ver figura 5.

Figura 5: Uso de opción GPS + TU DESTINO.

Aplicación en ejecución.

En la figura anterior, el icono que se puede observar en el mapa, hace referencia a la ubicación actual del usuario.

Figura 6: Uso barra de búsqueda para localizar destino.



Aplicación en ejecución.

En la figura anterior, al hacer uso de la barra de búsqueda, el usuario debe seleccionar una de las opciones presentadas por el app y esta selección será tomada como destino y se procederá a realizar el cruce de rutas.

Figura 7: Resultado de búsqueda.

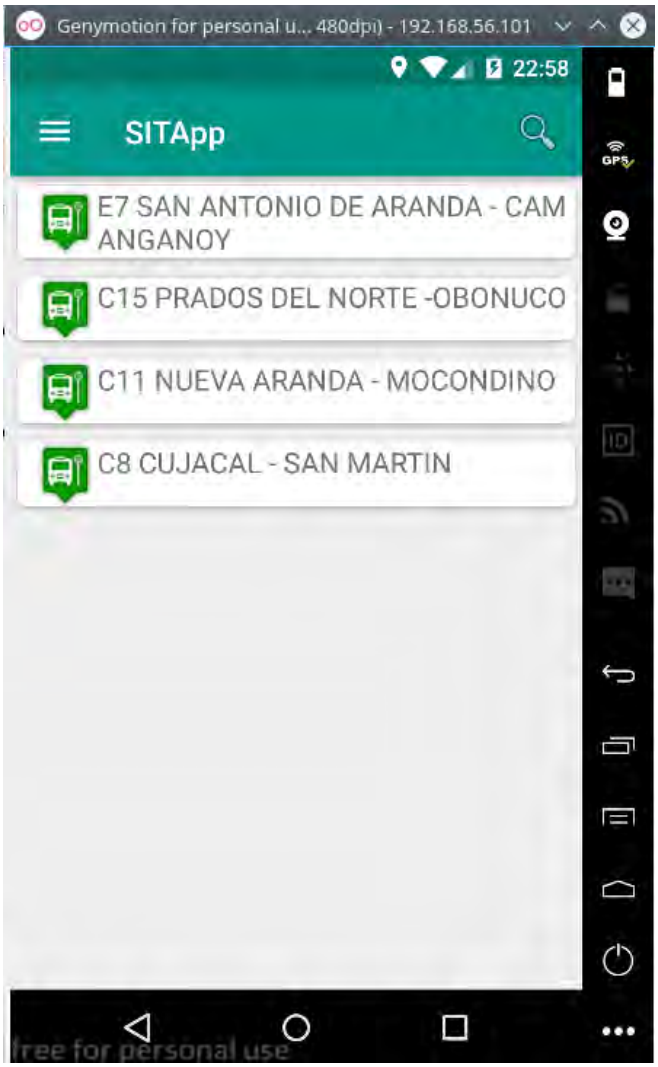
Aplicación en ejecución.

Consideraciones para la figura anterior.

- El aplicativo, en la sección mensajes, muestra información de la ruta seleccionada para realizar el viaje para este ejemplo la ruta seleccionada es la E7.
- Los iconos se pintan de acuerdo al tipo de sitio, es decir, si es una iglesia entonces el icono corresponderá con una iglesia o si es un parque el icono se pintara con una imagen que haga alusión a este tipo de sitio.
- La línea de color naranja representa el recorrido que realiza el bus.

- El icono de color verde hace referencia al paradero al cual se debe dirigir el usuario para tomar el bus.
- El icono de color rojo hace referencia al paradero en el cual el usuario debe bajarse.
- Las líneas de color azul, dibujan el camino que el usuario debe seguir para llegar a su sitio destino, en caso de que el bus no pase justo sobre el sitio destino.
- En caso de existir más de una ruta que pase cerca a la ubicación del usuario y que además pase cerca al sitio destino, entonces, el app muestra un botón de color verde, el cual está ubicado en la parte inferior derecha, al presionar este botón se despliega una lista de rutas que el usuario puede seleccionar para observar cual es el recorrido seguido y así tomar la ruta que él considere mejor, ver figura 8.

Figura 8: Lista de posibles rutas para realizar el viaje en bus.

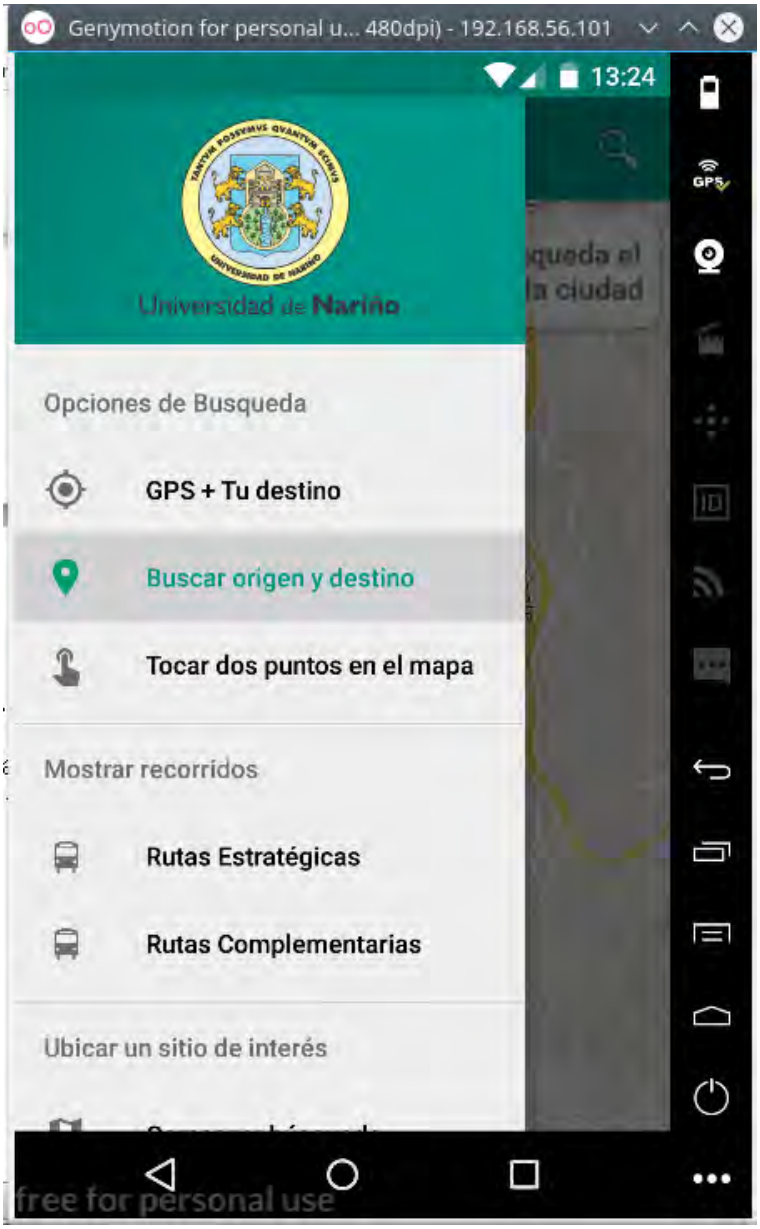


Aplicación en ejecución.

6.2. **BUSCAR ORIGEN Y DESTINO.**

Esta función permite que el usuario, haciendo uso de la barra de búsqueda, digite el sitio de origen y el punto destino, ver figura 9.

Figura 9: Función BUSCAR ORIGEN Y DESTINO.



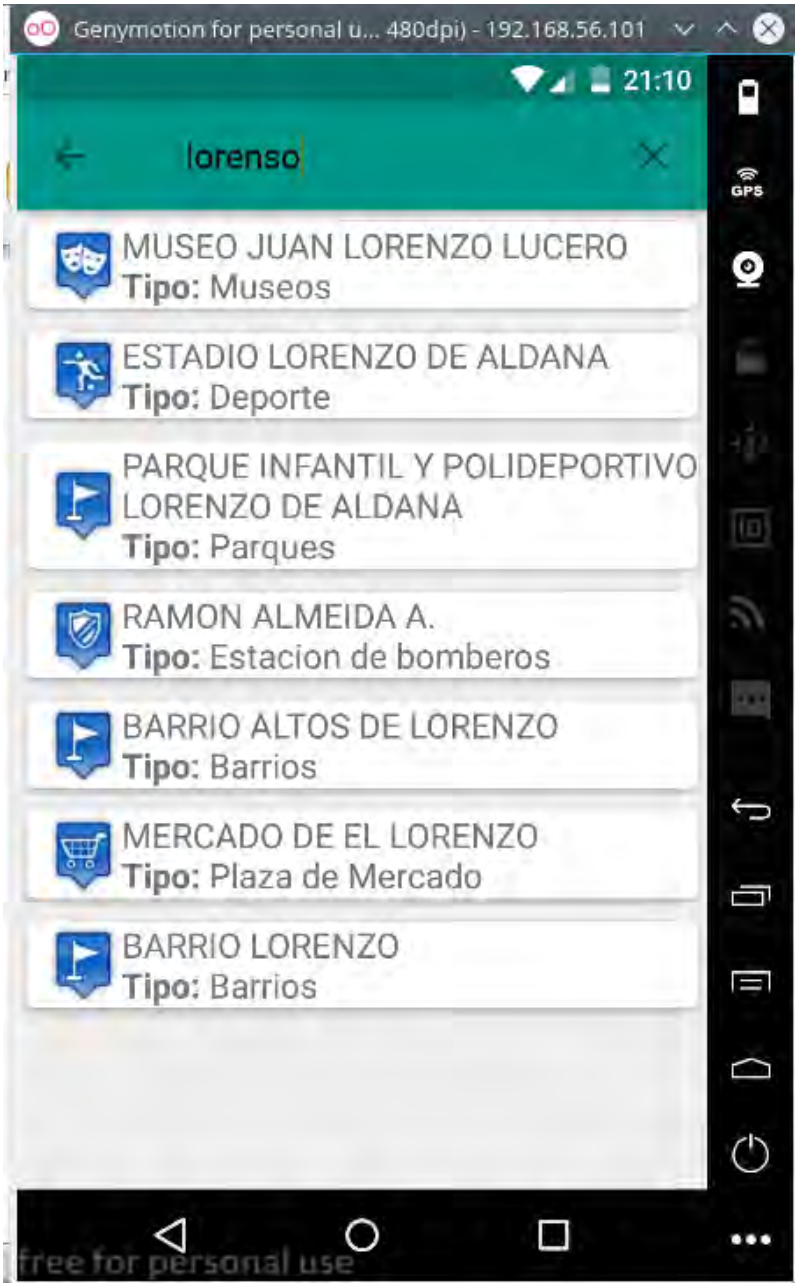
Aplicación en ejecución.

En este caso el usuario debe digitar los dos sitios, origen y destino, haciendo uso de la barra de búsqueda.

Figura 10: Buscar origen de viaje.

Aplicación en ejecución.

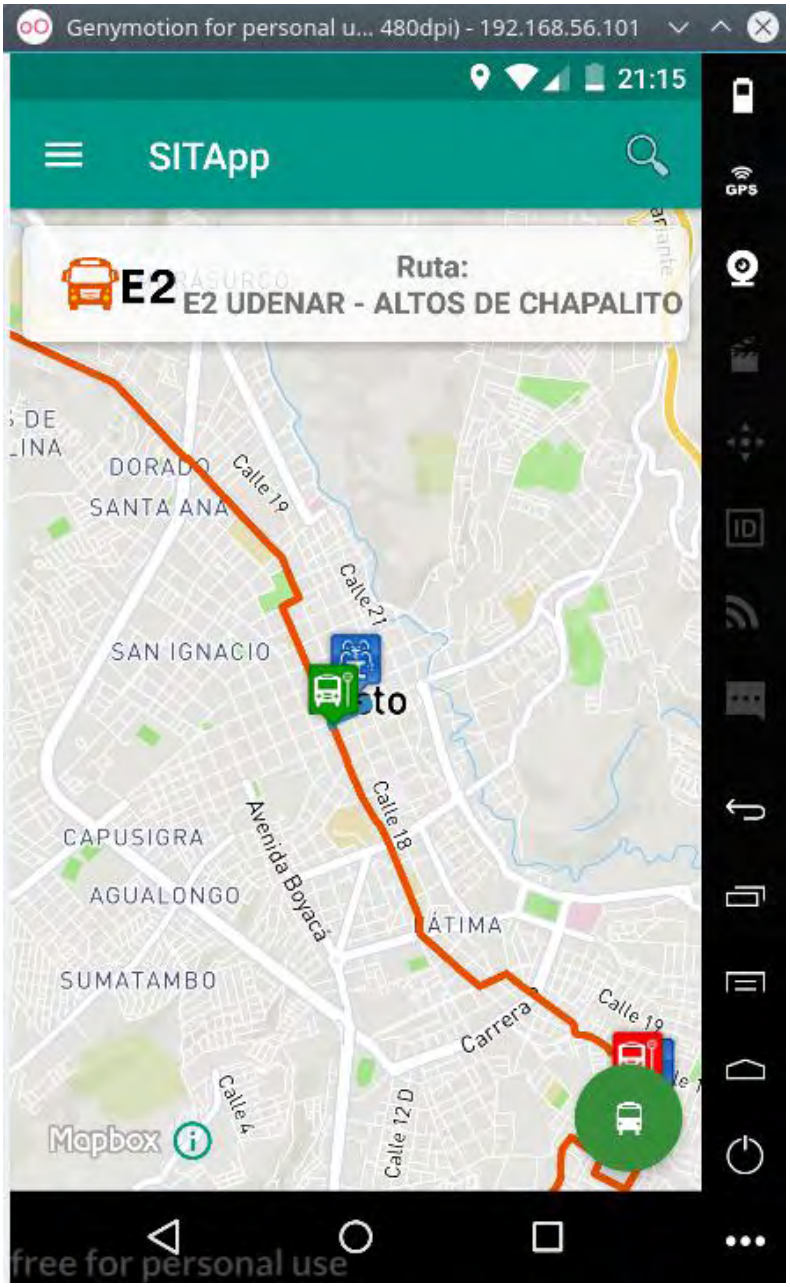
Figura 11: Buscar destino de viaje.



Aplicación en ejecución.

Como se puede observar en las dos figuras anteriores, a pesar de existir errores de ortografía el sistema brinda una seria de posibles respuestas.

Figura 12: resultado opción BUSCAR ORIGEN Y DESTINO.



Aplicación en ejecución.

El ejemplo anterior es un viaje desde Plaza de Nariño hasta el barrio Lorenzo. Al igual que en la función GPS +TU DESTINO, en el caso de existir más de una ruta que cumpla con las condiciones de viaje el sistema muestra un botón de color verde el cual permite conocer más de una opción al momento de viajar en bus.

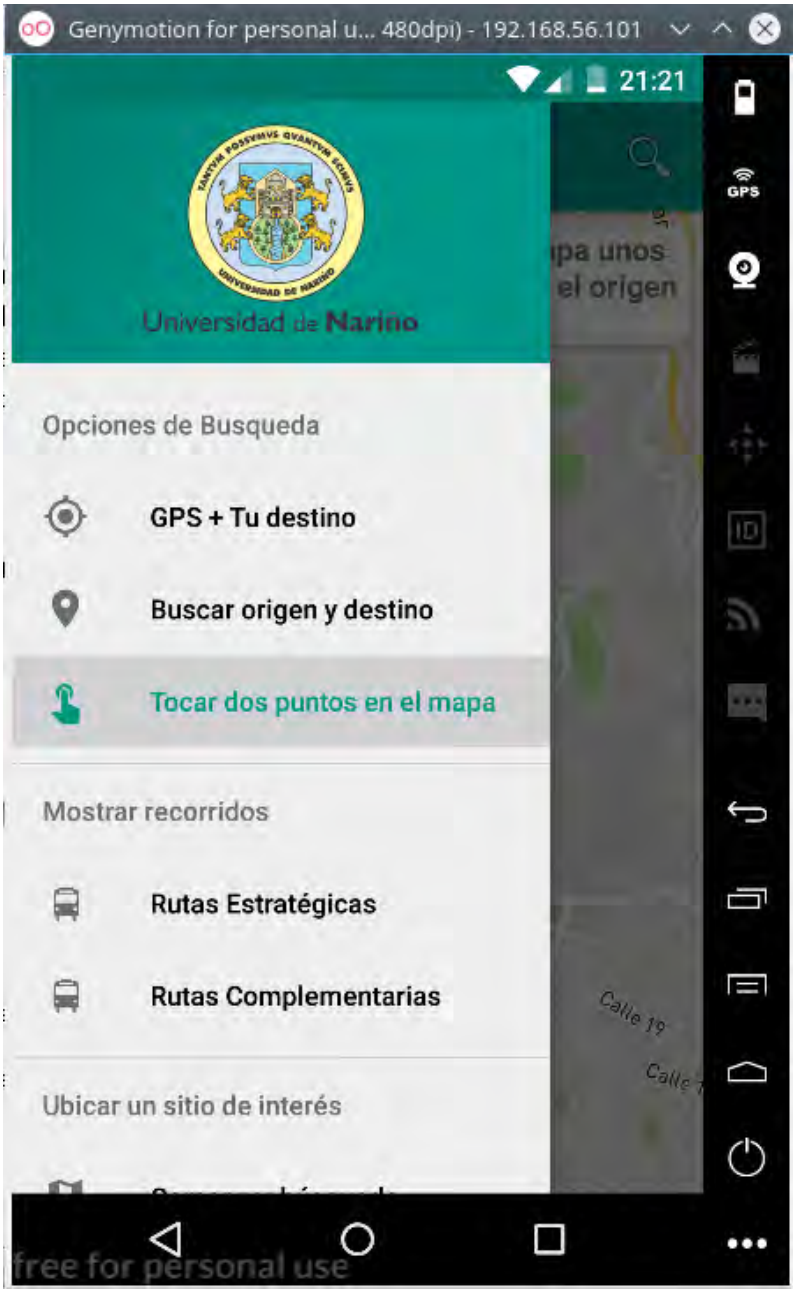
La sección mensajes ha sido actualizada y ahora muestra información de la ruta seleccionada para realizar el viaje en bus.

6.3. **TOCAR DOS PUNTOS EN EL MAPA.**

Esta función permite que el usuario pueda hacer clic sobre el mapa y el punto que el usuario seleccione será tomado como parámetro en la búsqueda de rutas del SETP del municipio de Pasto.

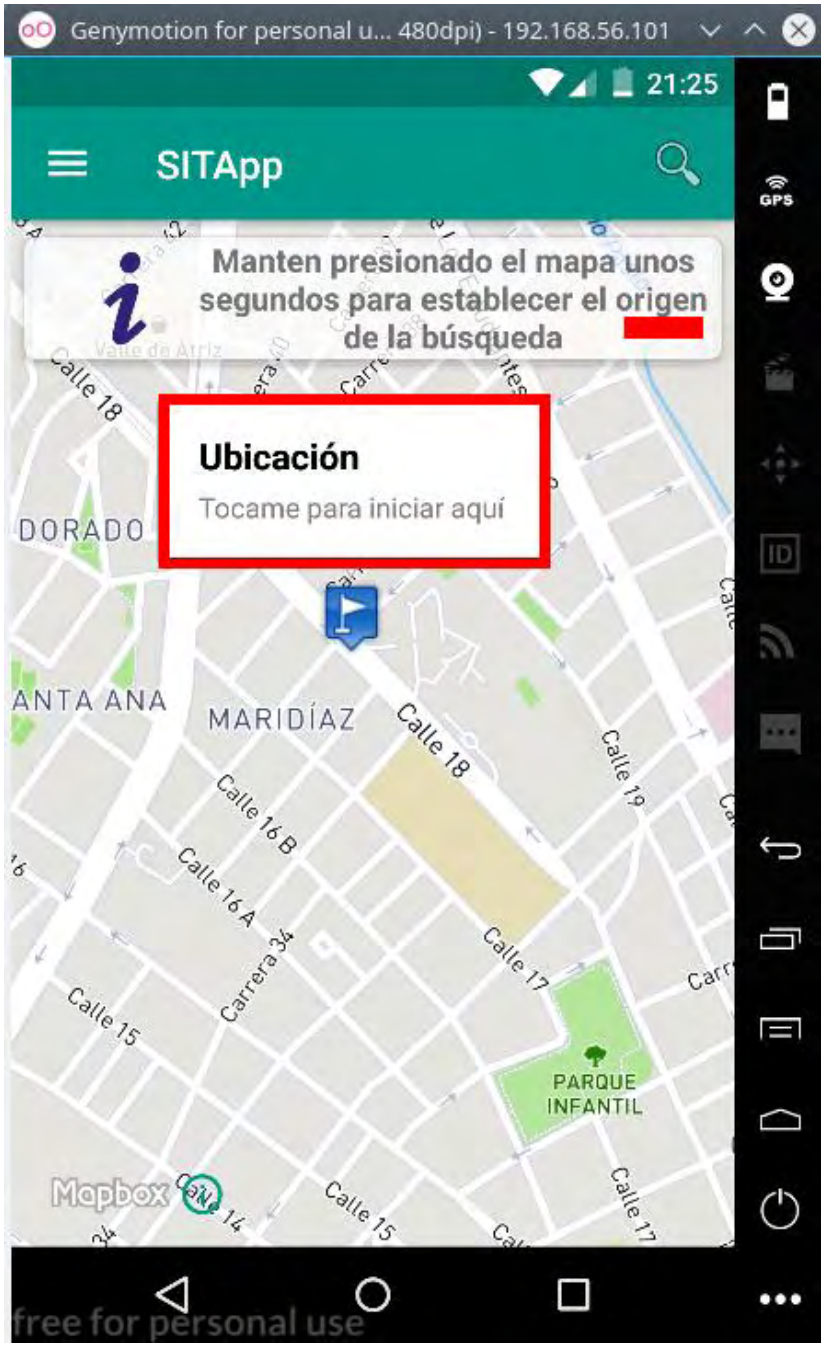
En este caso el usuario debe presionar unos segundos el mapa y al momento de que se dibuje el marcador hacer clic sobre la ventana emergente, ver figura 13.

Figura 13: Función TOCAR DOS PUNTOS EN EL MAPA.



Aplicación en ejecución.

Figura 13: Tocar mapa para seleccionar origen de viaje.



Aplicación en ejecución.

En la figura anterior, se puede apreciar que el sistema muestra un mensaje indicando al usuario que debe hacer, además se muestra un icono de color azul, el cual tiene una ventana sobre él, indicando que se debe presionar este elemento para establecer el origen del viaje.

Figura 14: Tocar puntos para establecer destino de viaje.

Aplicación en ejecución.

En la figura anterior se puede ver que la sección mensajes fue actualizada para indicar al usuario que debe ubicar su destino, además se dejó ubicado el origen del viaje para que el usuario pueda observar cual su origen y cuál será su destino. Al igual que en el paso anterior, el usuario debe presionar el elemento ubicado sobre el marcador para establecer el destino de viaje.

Figura 15: Resultado TOCAR DOS PUNTOS EN EL MAPA.

Aplicación en ejecución.

Al igual que en las dos funciones anteriores el sistema muestra paradero de inicio de viaje y paradero de fin de viaje además del botón que permite conocer más opciones de rutas que cumplan con las condiciones de viaje.

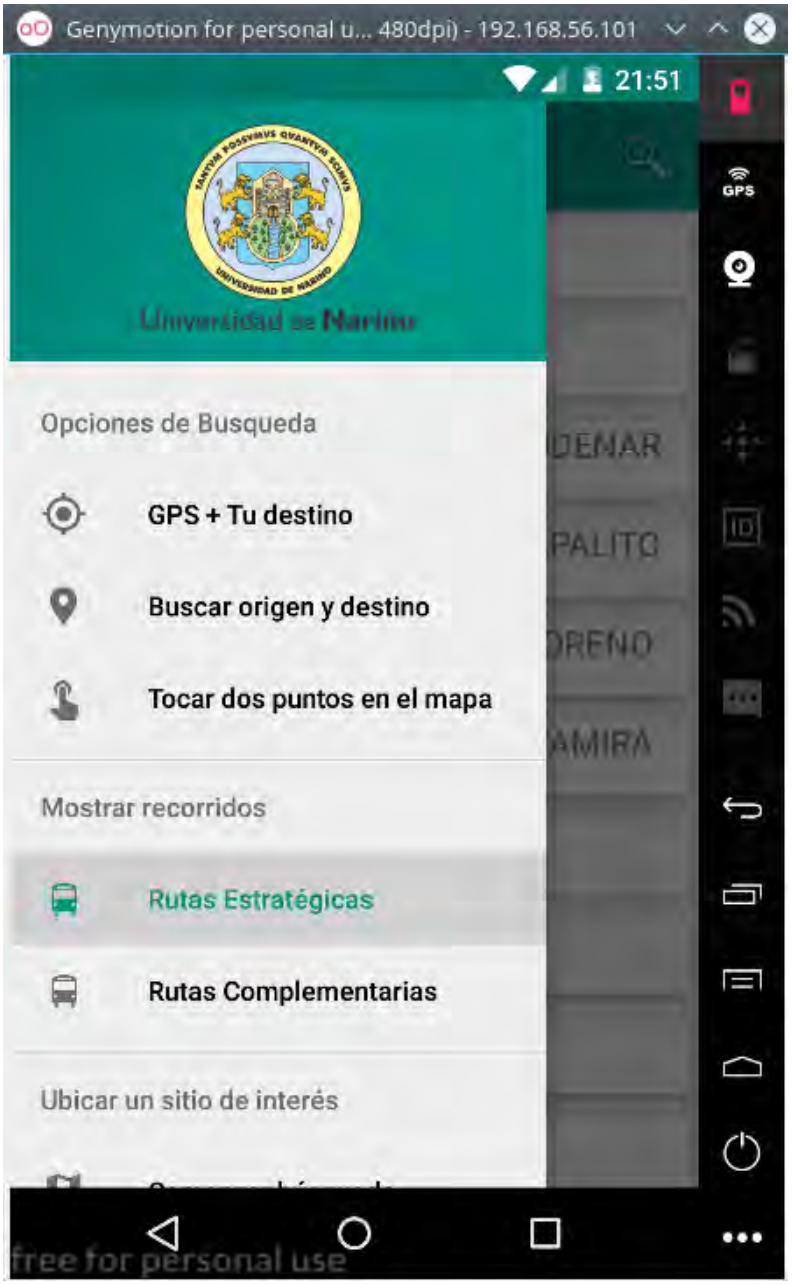
El espacio para mensajes ha sido actualizado y ahora muestra información de la ruta seleccionada para realizar el viaje en bus.

6.4. **MOSTRAR RECORRIDOS DE RUTAS.**

Esta opción permite que el usuario pueda ver el recorrido de las rutas y las diferentes paradas que hace el bus.

En esta parte el usuario únicamente debe seleccionar la ruta, de la cual desea ver el recorrido, y el sistema mostrará sobre el mapa el recorrido y las paradas que hace la ruta seleccionada, ver figura 16.

Figura 16: Opción RECORRIDOS DE RUTAS.



Aplicación en ejecución.

Figura 17: Ruta C9, recorrido y paradas.

Aplicación en ejecución.

Consideraciones.

- La sección mensajes muestra detalles de la ruta seleccionada.
- Sobre el mapa se pinta recorrido de ruta, color naranja, paradas, color azul.

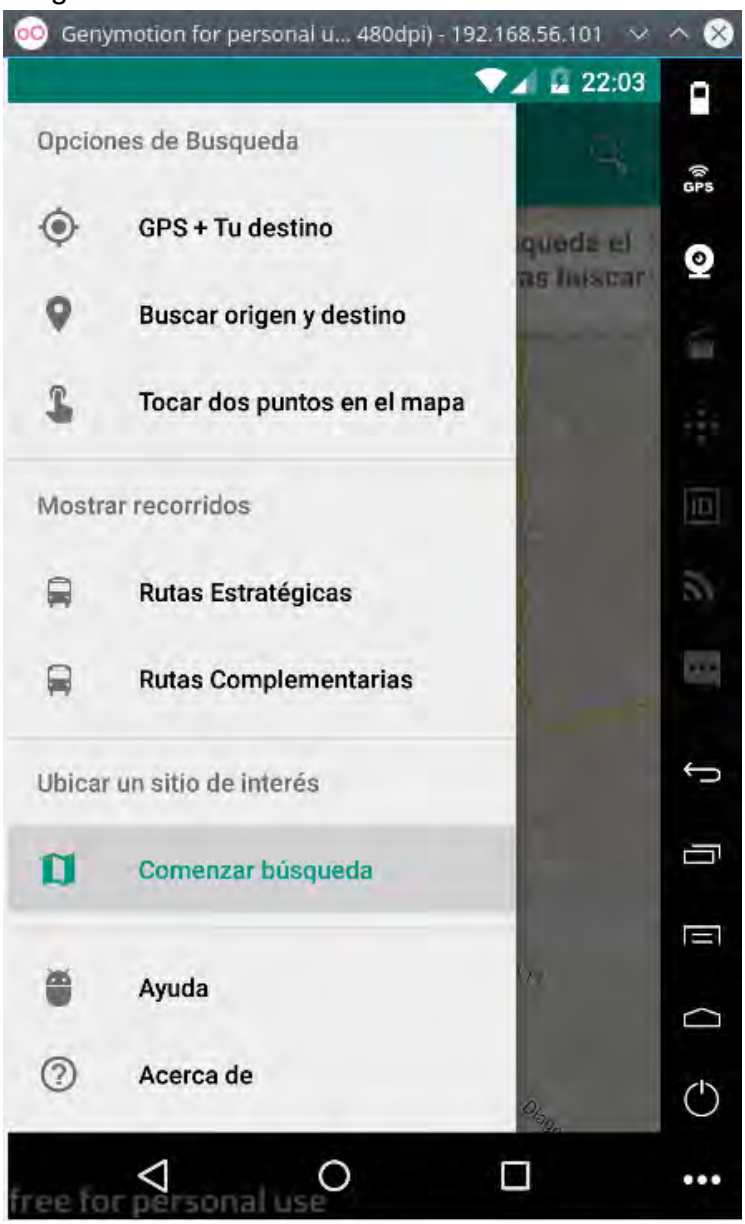
- Iconos de color verde y rojo indican, respectivamente, inicio y fin de recorrido.
- El usuario puede hacer clic sobre una de las paradas para conocer algunos detalles de la parada seleccionada.

6.5. **UBICAR SITIO DE INTERÉS.**

Esta opción permite encontrar sitios de interés de la ciudad de Pasto.

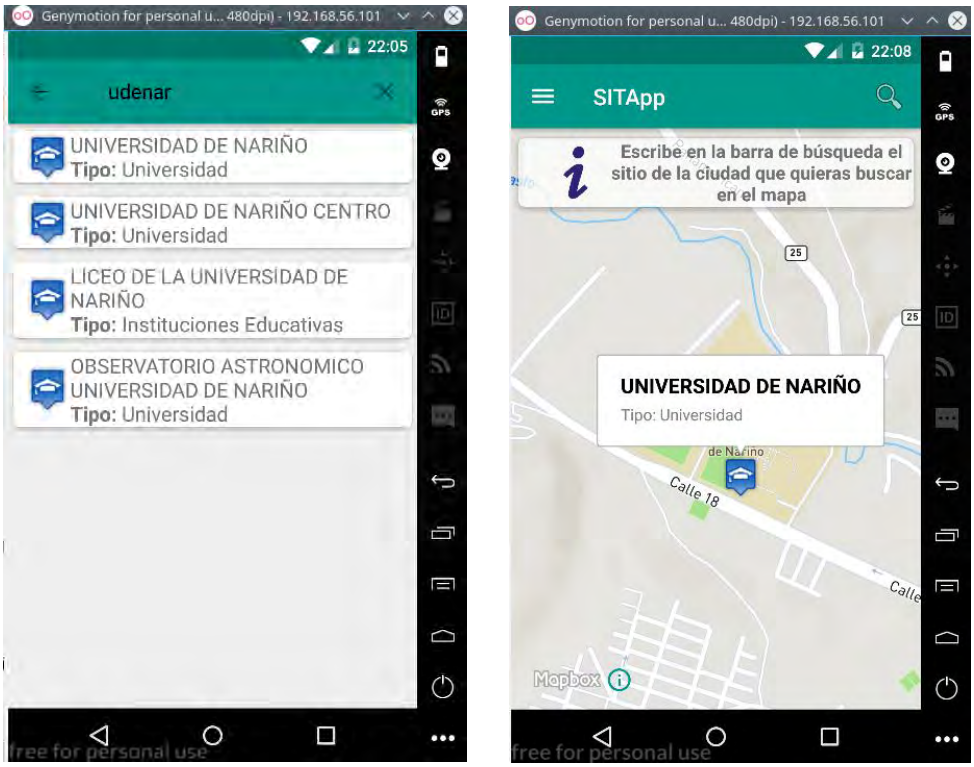
Al igual que en la función ORIGEN Y DESTINO, el usuario debe hacer uso de la barra de búsqueda para encontrar el sitio deseado, ver figura 18.

Figura 18: Función UBICAR SITIO DE INTERÉS.



Aplicación en ejecución.

Figura 19: Ubicar sitio de interés de la ciudad de Pasto.



, aplicación en ejecución.

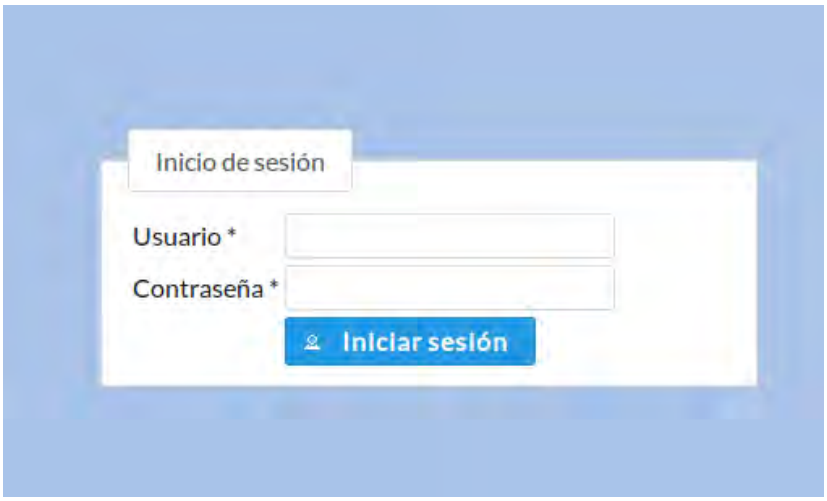
ANEXO C. Manual de usuario módulo administrativo SITApp.

La siguiente es la documentación que pretende guiar a los usuarios finales en el uso del aplicativo desarrollado. Cabe destacar que solo existe un rol asociado al sistema, administrador.

1. Inicio de sesión

La siguiente imagen, figura 1, es la presentación inicial del módulo administrativo de SITApp. El usuario, administrador, debe iniciar sesión para realizar cambios sobre la bases de datos del aplicativo móvil.

Figura 1: Interfaz inicial del módulo administrativo.



Aplicación en ejecución.

2. Interfaz secundaria de la aplicación

Una vez el usuario ha iniciado sesión, se muestra la interfaz, figura 2, mediante la cual puede realizar labores de actualización de rutas y paraderos del Sistema Estratégico de Transporte del municipio de Pasto (SETP).

Figura 2: Interfaz secundaria del módulo administrativo.

Aplicación en ejecución.

Esta interfaz contiene una barra de menú con 3 opciones, paraderos, rutas e inicio; estas opciones permiten gestionar la información de paraderos y rutas del SETP de la ciudad de Pasto. Además la barra contiene dos botones de color rojo y verde desde los cuales el usuario puede cerrar sesión o cambiar la contraseña del usuario administrador. Los puntos, que se pueden ver sobre el mapa, son algunos paraderos del SETP del municipio de Pasto.

Las opciones de paraderos y rutas se describen a continuación.

3. Opción paraderos

El usuario puede realizar:

- Eliminar paradero.
- Crear paradero.
- Ver y Editar.
- Asociar paraderos.
- Des-asociar paraderos.

3.1. Eliminar paradero

En esta opción el sistema muestra un listado de todos los paraderos que existen en la base de datos. El usuario puede buscar y seleccionar el paradero que desea eliminar, ver Figura 3.

Figura 3: Listado de paraderos con la opción de eliminar.

Aplicación en ejecución.

Para localizar el paradero a eliminar, el usuario tiene dos maneras para lograrlo. Escribir el código del paradero (número 1) o desplazarse haciendo uso de los números ubicados sobre la tabla de paraderos (número 2), una vez localizado el paradero el usuario debe presionar el botón eliminar (número 3). Cuando el usuario presiona el botón eliminar el sistema muestra una ventana emergente con los datos del paradero a eliminar y las opciones Aceptar o Cancelar, ver figura 4.

Figura 4: Ventana emergente con información de paradero seleccionado para ser eliminado.

, aplicación en ejecución.

NOTA: Esta opción borra el paradero de la base de datos, si una o más rutas tienen asociado este paradero al presionar el botón aceptar la(s) ruta(s) quedara(n) sin este paradero dado que dejara de existir.

3.2. **Crear paradero**

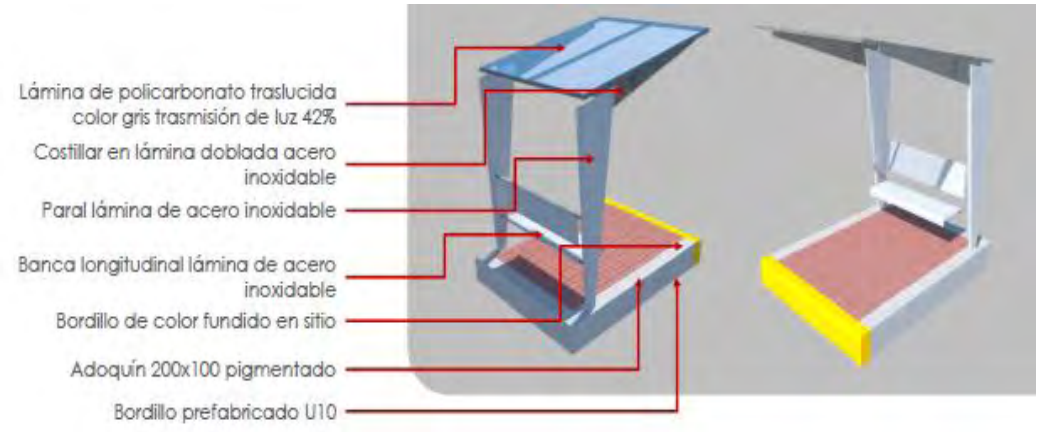
Esta opción permite al usuario administrador crear nuevos paraderos en el SETP. El usuario debe llenar datos referentes al paradero como son código, nombre, tipo, localización y además debe ubicar el paradero sobre un mapa, ver figura 5.

Figura 5: Crear paradero en el SETP.

Aplicación en ejecución.

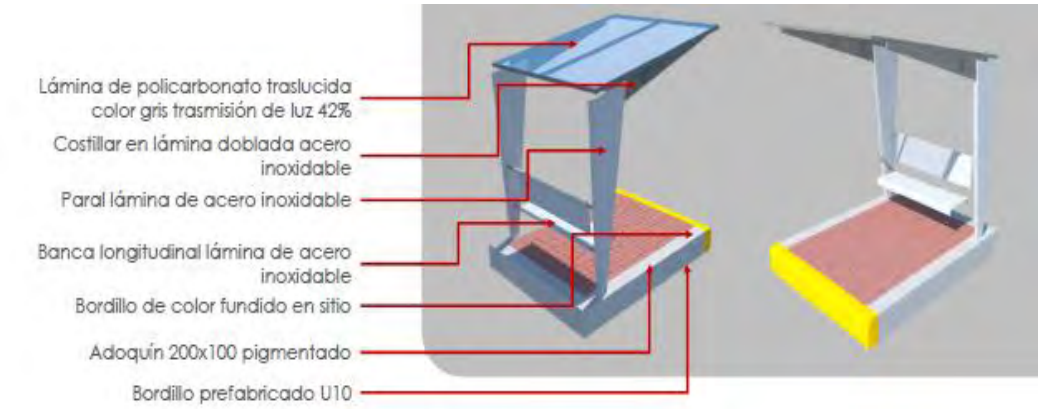
Existen 11 tipos diferentes de paraderos pero estos son el resultado de una combinación de módulos así que para lograr una mayor comprensión de los tipos de paraderos que tiene el SETP de Pasto empezaremos por describir, a grandes rasgos, cada uno de los módulos.

Figura 6. Módulo 1 (M1)



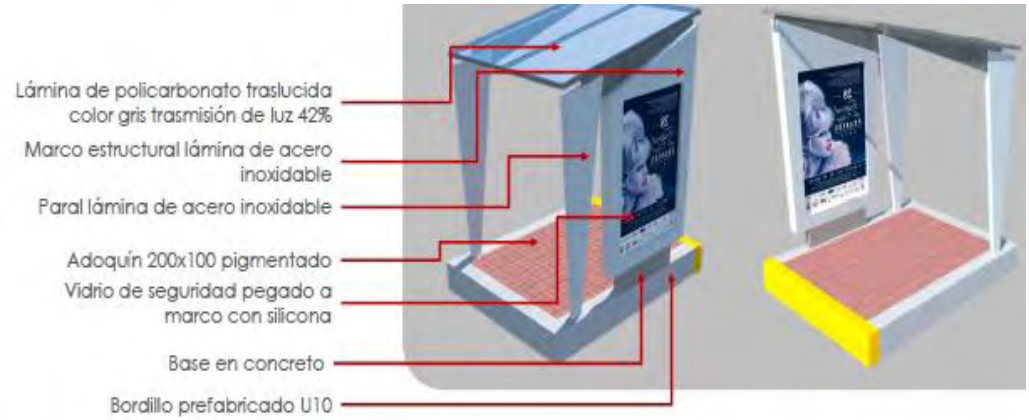
Fuente: [28].

Figura 7. Módulo 2 (M2)



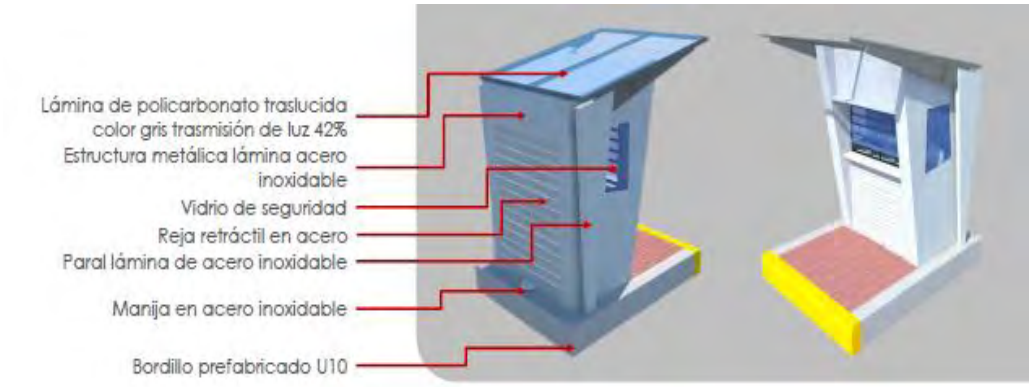
Fuente: [28].

Figura 8. Módulo 3 (M3)



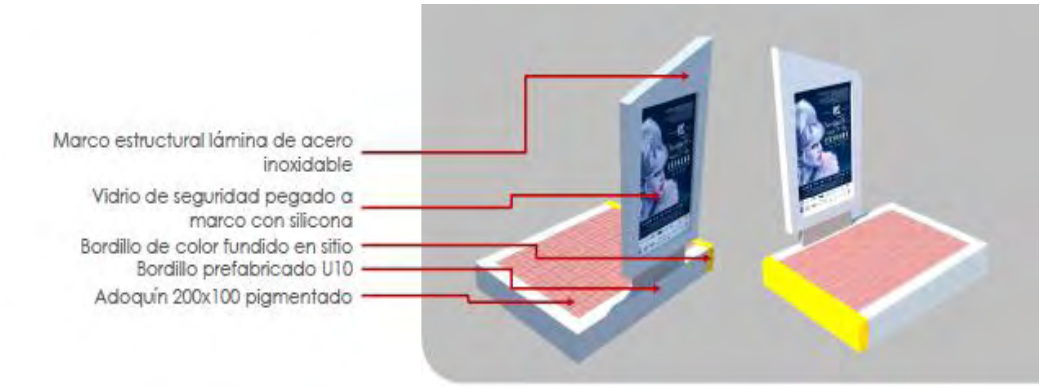
Fuente: [28].

Figura 9. Módulo 4 (M4)



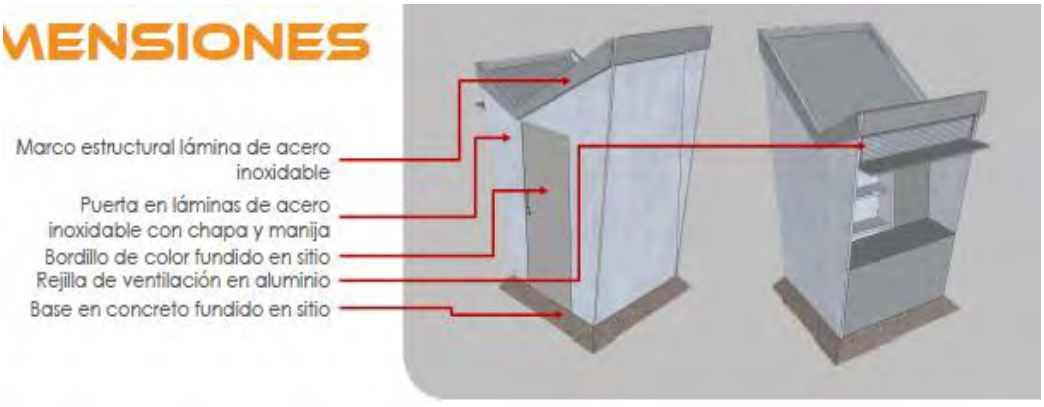
Fuente: [28].

Figura 10. Módulo 5 (M5)



Fuente: [28].

Figura 11. Módulo 6 (M6)



Fuente: [28].

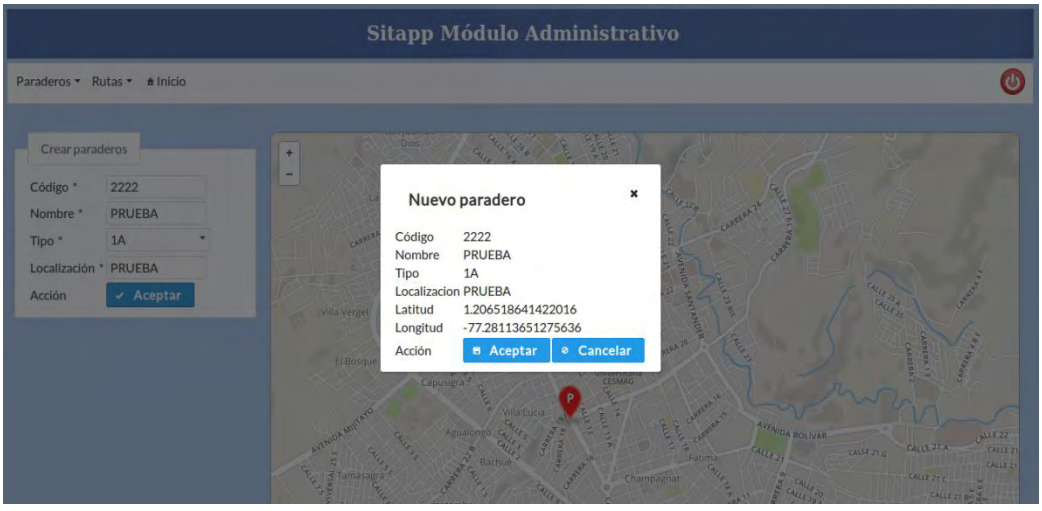
Ahora bien, los paraderos resultan de las combinaciones de los módulos anteriores, por lo tanto.

- Paradero P1A compuesto por módulos (M1 + M1)
- Paradero P1B compuesto por módulos (M2 + M3)
- Paradero P2A compuesto por módulos (M1 + M2 + M3)
- Paradero P2B compuesto por módulos (M2 + M2 + M3)
- Paradero P2C compuesto por módulos (M2 + M1 + M3)
- Paradero P3A compuesto por módulos (M2 + M1 + M4 + M5)
- Paradero P3B compuesto por módulos (M2 + M2 + M4 + M5)
- Paradero P3C compuesto por módulos (M1+ M1 + M4 + M5)
- Paradero 2+3 compuesto por módulos (M1 + M5)
- Paradero S compuesto por módulos (M2 + M4)
- Paradero NA, no aplica, es decir lo único que se ubica en el sitio del paradero es un tótem.

NOTA: El usuario debe hacer doble clic sobre el mapa para que se dibuje el marcador y en el caso de que la posición no sea correcta puede arrastrar el marcador hasta la ubicación deseada. **Se recomienda que el paradero este lo más cerca posible a la malla vial** dado que al momento de asociar un paradero a una ruta debe existir una distancia mínima entre el paradero y la vía, así que si el paradero está demasiado lejos no será posible asociarlo a la ruta deseada.

Al presionar el botón aceptar se muestra una ventana emergente con la información del nuevo paradero y las opciones Aceptar o Cancelar, ver figura 6.

Figura 12: Ventana emergente con los datos del nuevo paradero.



Aplicación en ejecución.

3.3. **Ver y editar**

Esta opción presenta un listado de paraderos y dos botones con la opción Ver o Editar, ver figura 7.

Figura 13: Listado de paraderos con las opciones Ver y Editar.

Aplicación en ejecución.

Al igual que en la opción de Eliminar, el usuario tiene varias formas para ubicar el paradero deseado. El usuario puede escribir el código del paradero (número 1), nombre del paradero (número 2) o utilizar los números ubicados en la parte superior del listado de paraderos (número 3). Las opciones Ver y Editar están en los botones con forma de lupa (número 4) y tintero (número 5) respectivamente.

3.3.1. Ver paradero

La opción Ver, re-direcciona a una página la cual muestra información y ubicación en el mapa del paradero seleccionado, ver figura 8.

Figura 14: Página con información del paradero seleccionado.

Aplicación en ejecución.

3.3.2. Editar paradero

La opción Editar, muestra la información y ubicación del paradero seleccionado pero permite editarla, ver figura 9.

Figura 15: Información y ubicación de paradero seleccionado.

, aplicación en ejecución.

NOTA: Para actualizar la ubicación del paradero sencillamente se arrastra el marcador hacia la posición deseada y una vez realizados los cambios se presiona el botón Aceptar.

3.4. Asociar paraderos

Esta opción permite asociar nuevos paraderos a las rutas del SETP, ver figura 10. El usuario debe seleccionar la ruta, el sistema busca automáticamente todos los paraderos que no están asociados a esa ruta y los muestra en la sección paraderos donde se debe seleccionar el paradero que se desea asociar a la ruta.

Consideraciones, El paradero no puede estar a más de 2 metros de la ruta. La secuencia, hace referencia al orden de los paraderos, puede estar entre el primer paradero y el último + 1, para los demás casos generará un error.

NOTA: El aplicativo móvil trabajo con los paraderos y estos son los que dan el sentido de la ruta, si va o vuelve, esto debido a que no existe una conexión con los GPS de los buses, es por eso que se debe tener cuidado al momento de asociar un paradero, **CORROBORAR QUE EL ORDEN SEA EL CORRECTO**. Para corroborar el orden se puede hacer clic sobre los marcadores para ver información relacionada.

Figura 16: Asociar paraderos a una determinada ruta del SETP.

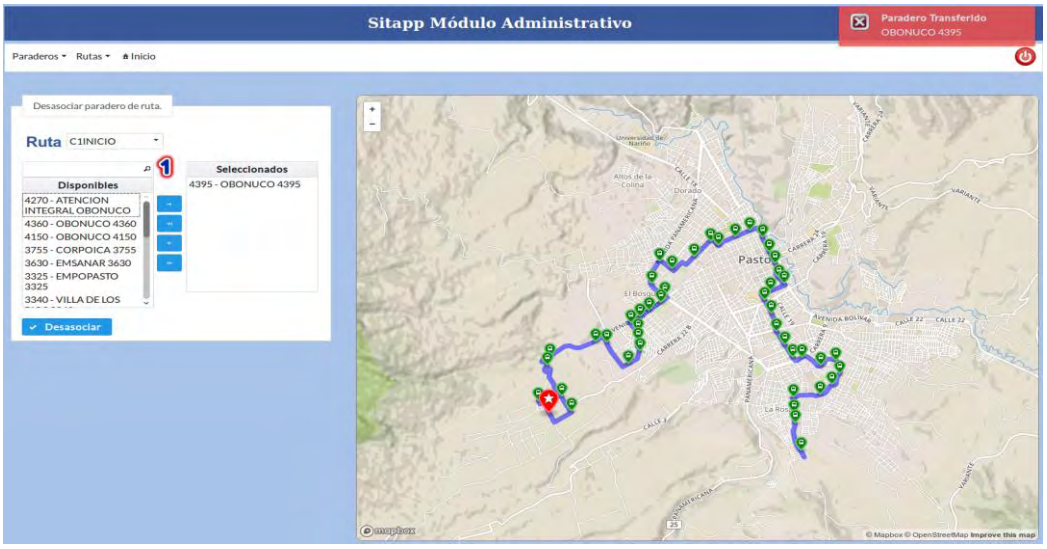
Aplicación en ejecución.

3.5. **Des-asociar paraderos**

Esta opción permite eliminar paraderos asociados a una ruta pero no los borra de la base de datos, únicamente borra el enlace entre la ruta y el paradero pero este último sigue almacenado en la base de datos.

El usuario debe seleccionar la ruta y el sistema lista automáticamente los paraderos asociados a esa ruta en específico y los ubica sobre un mapa, ver figura 11, hacer doble clic sobre el paradero a des-asociar, el cual será enviado al otro lado de la lista, y presionar el botón desasociar el sistema eliminará el enlace que existe entre la ruta y el paradero seleccionado.

Figura 17: Des-asociar paraderos de ruta.



Aplicación en ejecución.

Para ubicar el paradero a desasociar el usuario tiene dos maneras de hacerlo, escribir el código del paradero (número 1) o desplazarse hacia abajo hasta encontrar el paradero deseado.

4. Opción rutas

El usuario tiene las siguientes opciones.

- Eliminar ruta.
- Crear ruta.
- Ver y Editar.
- Paraderos asociados.

Estas opciones se describen a continuación.

4.1. Eliminar ruta

El sistema despliega un listado de las rutas almacenadas en la base de datos y las muestra en una tabla, ver figura 12, además de la información de las rutas, también se tiene un botón, con forma de cesto de basura (número 3), el cual permite eliminar la ruta seleccionada.

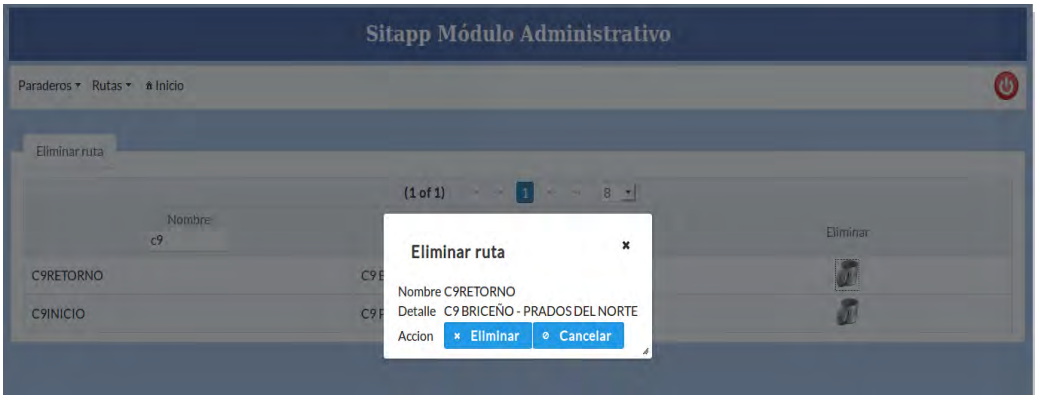
Figura 18: Listado de rutas con la opción de eliminar.

Aplicación en ejecución.

Para localizar la ruta a eliminar el usuario tiene dos maneras para lograrlo. Haciendo uso del cuadro de búsqueda (número 1) ubicado sobre el nombre de la ruta o utilizando los números ubicados sobre el listado de rutas (número 2), una vez localizada la ruta a eliminar se debe presionar el botón con forma de cesto de basura (número 3).

Al momento de presionar el botón eliminar, se despliega una ventana emergente con la información de la ruta a eliminar y dos botones, Aceptar o Cancelar, ver figura 13.

Figura 19: Ventana emergente con información de la ruta seleccionada para ser eliminada.



, aplicación en ejecución.

NOTA: Esta opción **elimina la ruta de la base de datos**, se debe proceder con cuidado.

4.2. **Crear ruta**

Esta opción permite crear rutas en el SETP de la ciudad de Pasto.

El sistema muestra un formulario dentro del cual se debe llenar información de la ruta a crear, la información es obligatoria, ver figura 14.

Figura 20: Formulario de creación de rutas.

Aplicación en ejecución.

Para crear la geometría de la ruta, el usuario debe hacer doble clic sobre el mapa para que el sistema cree un nuevo nodo y esta operación se debe repetir hasta lograr el resultado requerido.

Al hacer clic sobre alguno de los nodos se despliega una ventana emergente. Esta ventana permite eliminar el nodo o crear un nodo intermedio entre el anterior y el siguiente.

Los nodos se pueden dibujar al inicio o fin de la geometría, basta con hacer doble clic y el sistema determinará si lo añade al final o al inicio. Con esto logramos que la ruta pueda ser extendida al inicio o al final.

Cuando se ha logrado el resultado requerido se presiona el botón guardar y el sistema almacena la ruta en la base de datos. Luego se puede asociar los diferentes paraderos que pertenecen a la ruta que se acaba de crear.

4.3. **Ver y editar**

En esta opción, el sistema muestra un listado de las rutas existentes en la base de datos. El listado además de contener información referente a las rutas tiene dos botones, uno con forma de lupa y otro con forma de tintero, estos botones permiten Ver y/o Editar una ruta determinada, ver figura 15.

Figura 21: Lista de rutas con las opciones Ver y Editar.

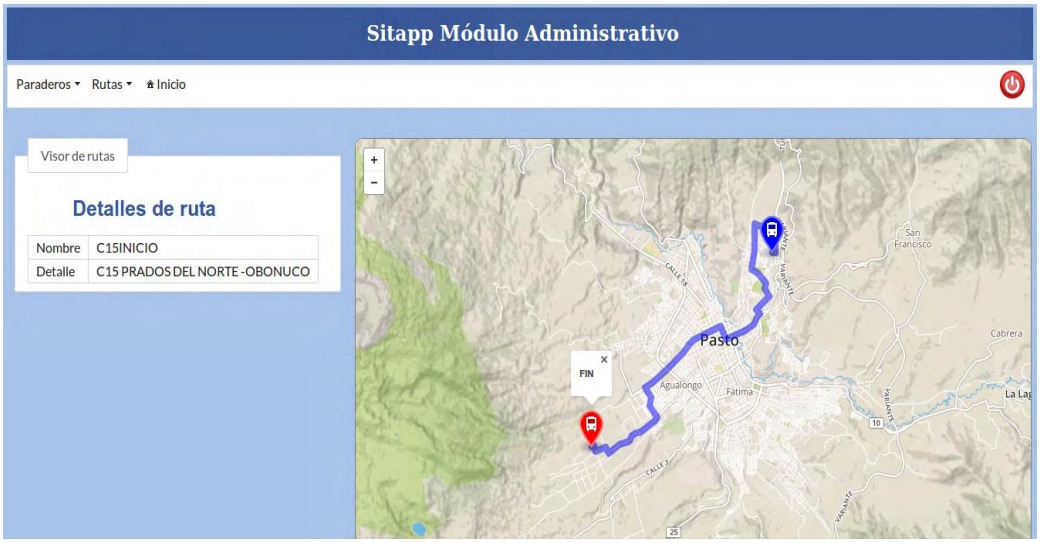
Aplicación en ejecución.

Para ubicar la ruta que se desea editar el usuario tiene tres opciones, escribir el nombre de ruta (número 1), escribir una palabra clave en el cuadro detalle (número 2) o utilizar los números ubicados sobre el listado de rutas. Una vez localizada la ruta se tiene dos opciones, Ver (número 4) o Editar (número 5).

4.3.1. Ver ruta

Al presionar el botón ver, se re-direcciona a una nueva página donde se muestra información de la ruta, nombre, detalle y se pinta sobre un mapa, ver figura 16.

Figura 22: Información y recorrido de ruta seleccionada.



Aplicación en ejecución.

4.3.2. Editar ruta

Al presionar el botón Editar, se re direcciona a una nueva página pero esta permite actualizar la información de la ruta además de su recorrido, ver figura 17.

Figura 23: Editar ruta seleccionada.

Aplicación en ejecución.

El usuario puede editar la información referente al detalle de ruta y el recorrido de la misma. Para editar el recorrido de la ruta, se puede lograr arrastrando los diferentes nodos que se han creado hasta la posición deseada, además, al hacer clic sobre alguno de los nodos se muestra una ventana emergente con dos botones al interior los cuales permiten eliminar o crear un nodo intermedio.

La ruta puede ser extendida hacia el inicio o al final, basta con hacer doble clic y el sistema determinará si lo adiciona al final o al inicio de la geometría.

4.4. **Paraderos asociados**

Esta opción permite ver los paraderos asociados a una determinada ruta, ver figura 18. El usuario selecciona la ruta y el sistema muestra un listado de los paraderos que tiene asociados y además dibuja el recorrido de la ruta con cada uno de los paraderos sobre un mapa.

Figura 24: Paraderos asociados a ruta seleccionada.

Aplicación en ejecución.

Al hacer clic sobre alguno de los paraderos se presenta, en una ventana emergente, el nombre y la ubicación del paradero.

ANEXO D. Manual de programador SITApp

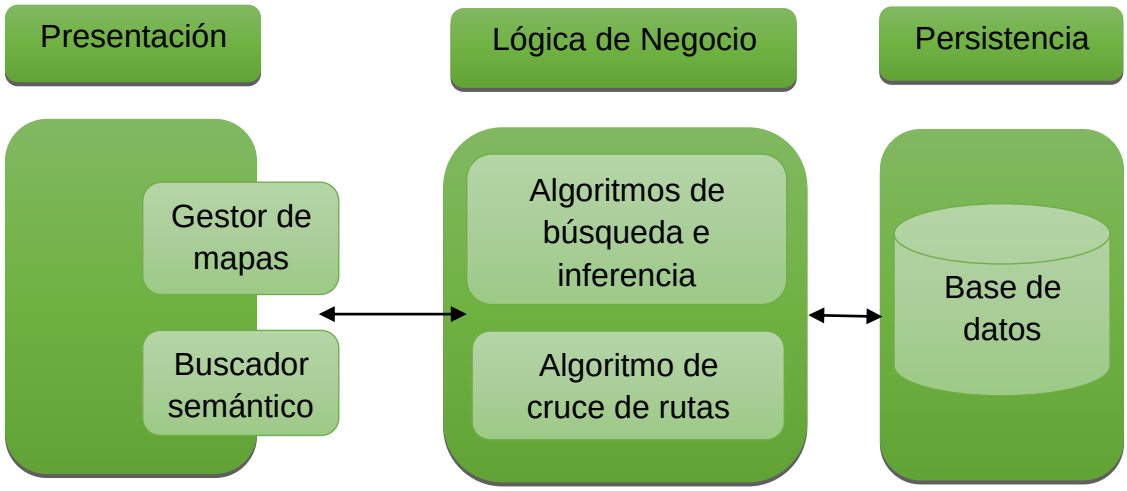
El presente documento es una guía para los programadores que hacen uso de SITApp Android y SITApp Web Service, es un manual práctico para la compresión y descripción de las principales funcionalidades que tiene este aplicativo móvil y su Web Service.

La construcción del aplicativo fue realizada bajo el lenguaje de programación Java, el cual es el lenguaje nativo de programación para Android, quien hace uso a su vez del lenguaje XML para los patrones de diseño de las interfaces gráficas de la aplicación. Se usó el IDE de desarrollo oficial Android Studio en el sistema operativo Debian 8, para el manejo de mapas se usó la librería Mapbox SDK, para la interacción de SITApp con el Web Service se usó la librería Retrofit.

La construcción del Web Service fue realizada bajo el lenguaje de programación Java Enterprise Edition 7 (Java EE7), haciendo uso del IDE Netbeans 8.1. Este aplicativo hace uso de la tecnología RESTful que utiliza los métodos HTTP y permite transferir archivos JavaScript Object Notation (JSON) para establecer comunicación entre el aplicativo en Android y la base de datos. Para el soporte de conexión y consultas hacia la ontología de SITApp se usó la librería Jena 2.7, adicionalmente la librería Lucene ayuda al Web Service a eliminar palabras muertas en las búsquedas del usuario y hacerlas más eficientes.

1. Arquitectura

Figura 1. Arquitectura SITApp

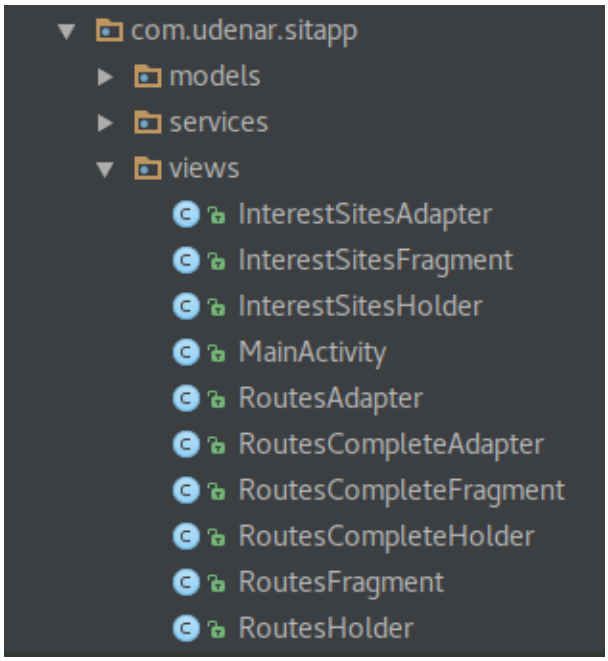


1.1. **Capa de presentación**

En esta capa se encuentran albergados todos los elementos de presentación gráficos para el usuario haciendo que los procesos que realiza el servidor para gestionar toda la información sean transparentes y disminuyendo cargas en el procesamiento de datos desde el lado del cliente, los elementos que le permiten interactuar con la aplicación y sus diferentes opciones son:

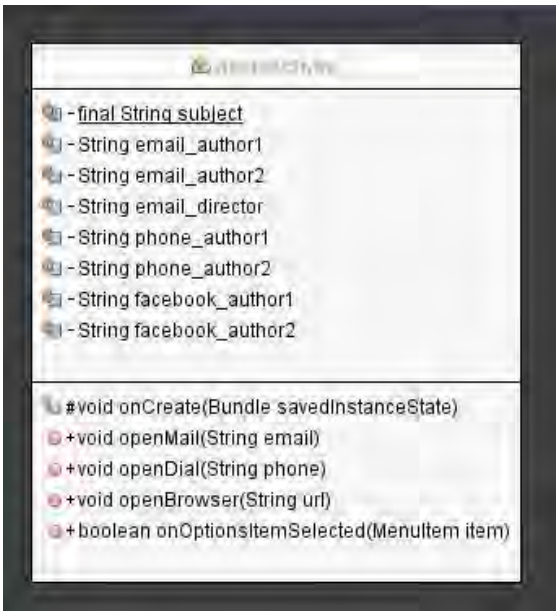
- Los gestores de mapas, que permiten visualizar toda la información georreferenciada acerca del SETP almacenada en el servidor desplegando esta información de distintas maneras de acuerdo a los distintos tipos de consultas que el usuario realice en su momento, mostrando las geometrías de los recorridos las rutas, los marcadores que representan a los paraderos o los sitios de interés de la ciudad de Pasto.
- El buscador semántico encargado de gestionar las consultas del usuario y realizar las peticiones hacia el Web Service, este buscador se comunica con la capa de presentación para mostrar los resultados de la búsqueda sin importar los errores de ortografía del usuario.

Figura 2. Clases del paquete views de SITApp



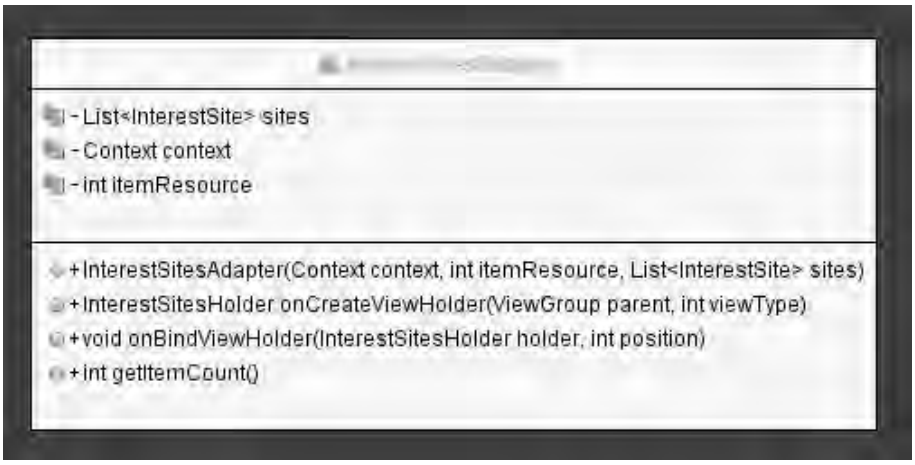
A continuación se describen las clases que componen este paquete:

Figura 3. Clase AboutActivity.java



Esta clase es la encargada de mostrar el activity que posee toda la información de contacto de los desarrolladores de la aplicación.

Figura 4. Clase InterestSitesAdapter.java



Clase que recibe y almacena el listado de sitios de interés.

Figura 5. Clase InterestSitesFragment.java

Clase encargada de crear el fragmento y hacerlo visible en la pantalla.

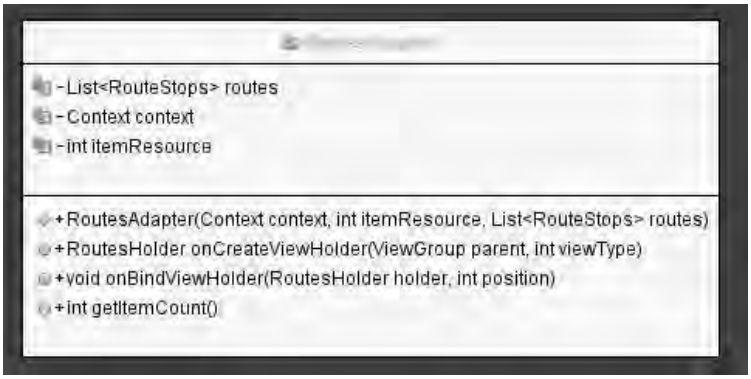
Figura 6. Clase InterestSitesHolder.java

Clase que representa cada uno de los sitios de interés de la lista y se encarga de realizar las acciones necesarias cuando el sitio de interés es seleccionado por el usuario.

Figura 7. Clase MainActivity.java

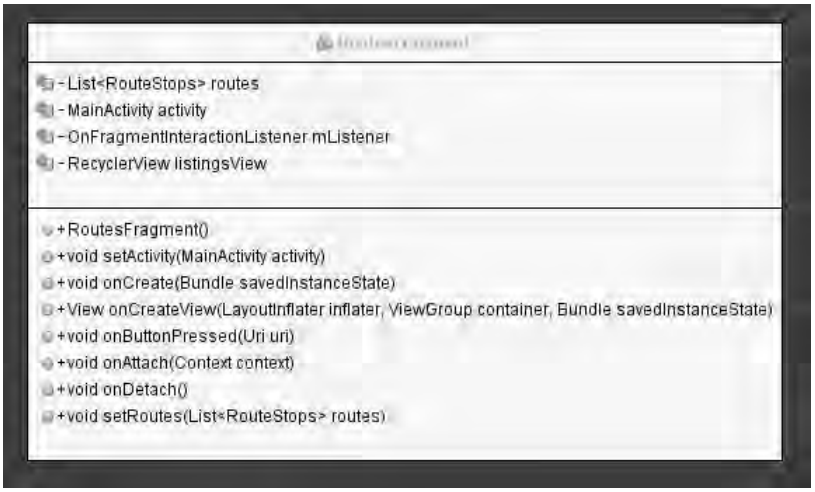
Clase principal que se encarga de gestionar y mostrar el mapa y las listas de rutas y sitios de interés, posee cada uno de los métodos necesarios para establecer la comunicación entre SITApp y el Web Service.

Figura 8. Clase RoutesAdapter.java



Clase que recibe y almacena el listado de las rutas con el paradero de origen y destino.

Figura 9. Clase RoutesFragment.java



Clase encargada de crear el fragmento y hacerlo visible en la pantalla.

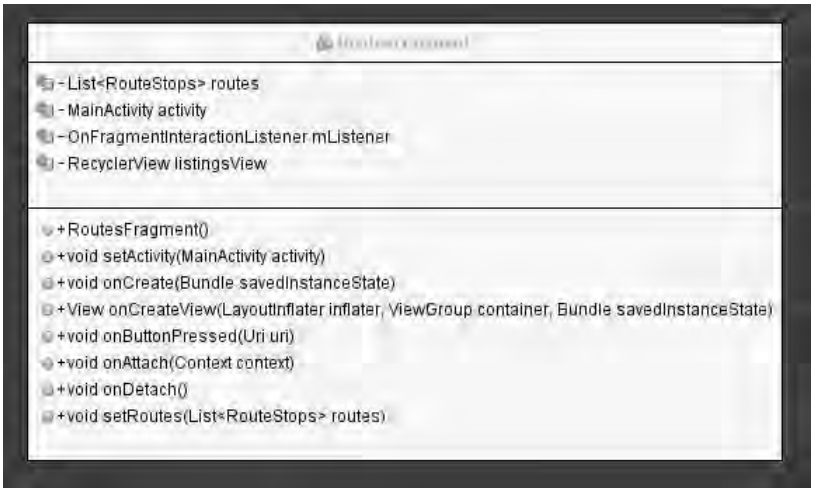
Figura 10. Clase RoutesHolder.java

Clase que representa cada una de las rutas en la lista y se encarga de realizar las acciones necesarias cuando la ruta es seleccionada por el usuario.

Figura 11. Clase RoutesCompleteAdapter.java

Clase que recibe y almacena el listado de las rutas con todos sus paraderos asociados.

Figura 12. Clase RoutesCompleteFragment.java



Clase encargada de crear el fragmento y hacerlo visible en la pantalla.

Figura 13. Clase RoutesCompleteHolder.java

Clase que representa cada una de las rutas en la lista y se encarga de realizar las acciones necesarias cuando la ruta es seleccionada por el usuario.

Figura 14. Clase WelcomeSITApp.java

Esta clase es la encargada de desplegar el menú de bienvenida de la aplicación.

1.2. **Lógica de negocio**

En esta capa se encuentran albergados los distintos algoritmos que se encargan de gestionar las distintas peticiones recibidas por la capa de presentación, aquí se lleva a lugar la tarea de procesamiento y gestión de toda la información georreferenciada mediante:

- Algoritmos de búsqueda e inferencia, encargados de gestionar consultas complejas realizando inferencias semánticamente haciendo uso de la ontología.
- Algoritmo de cruce de rutas, encargado de realizar la búsqueda de la o las rutas que llevan al usuario desde un punto geográfico “A” hasta un punto geográfico “B”.

Además se encuentran los procedimientos encargados de la comunicación entre SITApp y el Web Service.

Esta capa está compuesta por los paquetes “services” de SITApp y “conection”, “queries” y “service” del Web Serice. A continuación se describen cada uno de ellos y sus clases.

Figura 15. Clases del paquete services de SITApp

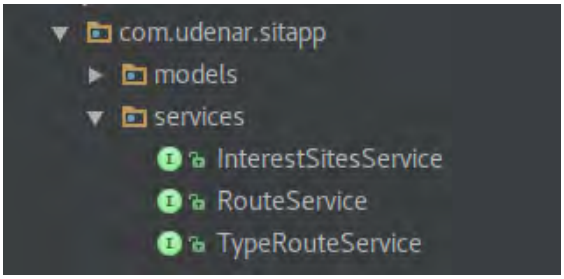
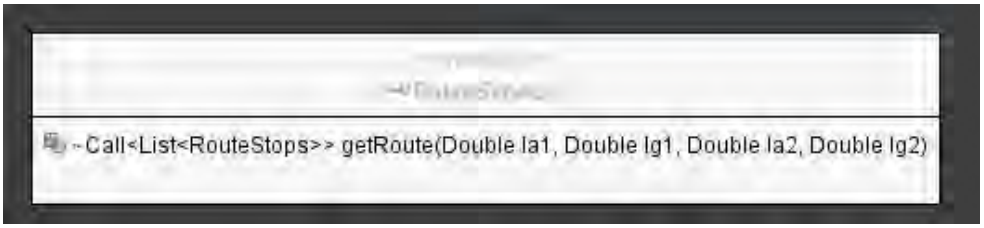


Figura 16. Interface InterestSitesService.java

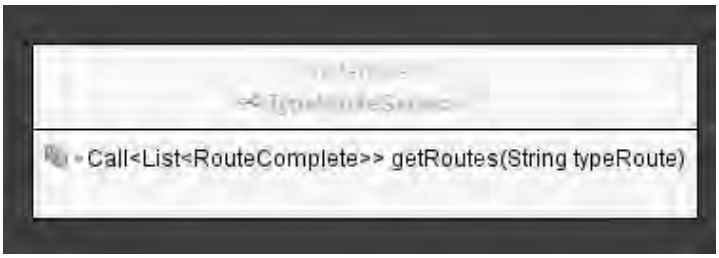
Interface encargada de controlar el método de envío de información hacia el Web service, lo realiza mediante el método GET.

Figura 17. Interface RouteService.java



Interface encargada de controlar el método de envío de información hacia el Web service, lo realiza mediante el método GET.

Figura 18. Interface TypeRouteService.java



Interface encargada de controlar el método de envío de información hacia el Web service, lo realiza mediante el método GET.

Figura 19. Clases del paquete ontology.conection del SITApp Web Service

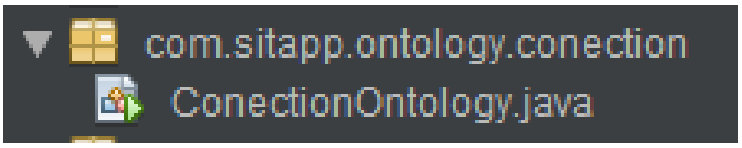
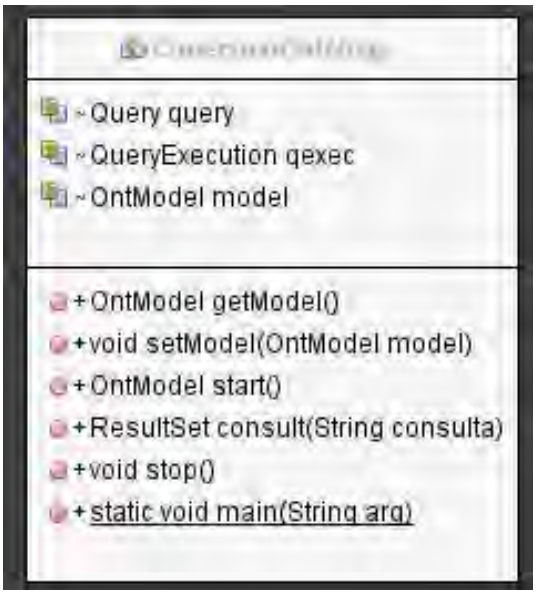


Figura 20. Clase ConectionOntology.java



Clase encargada de establecer la conexión con la ontología que se encuentra almacenada en el servidor.

Figura 21. Clases del paquete ontolgy.queries de SITApp Web Service

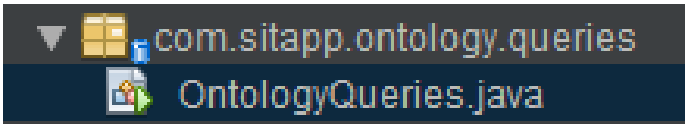
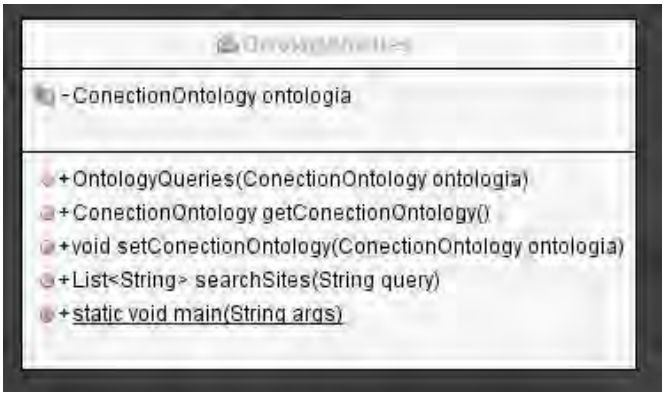
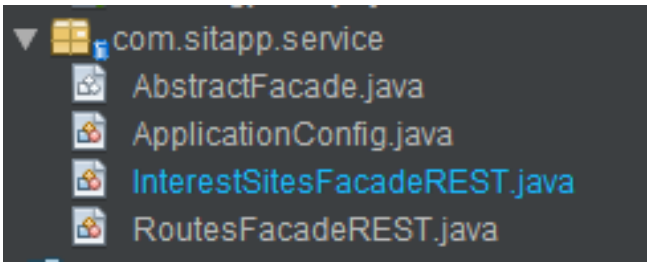


Figura 22. Clase OntologyQuieries.java

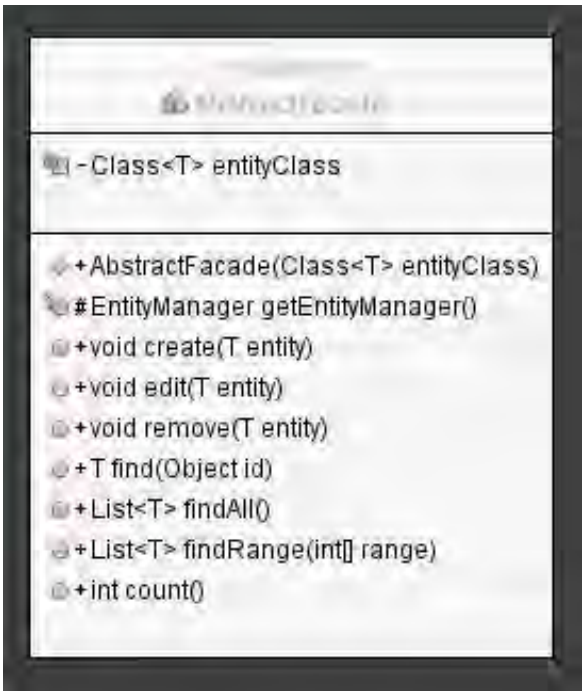


Clase que contiene las consultas SPARQL necesarias para acceder a los datos almacenados en la ontología.

Figura 23. Clases del paquete service de SITApp Web Service



Fuente 24. Clase AbstractFacade.java



Clase abstracta de la cual heredan los servicios del Web Service, contiene métodos necesarios para la comunicación entre el modelo de la base de datos que se encuentra en la capa de persistencia.

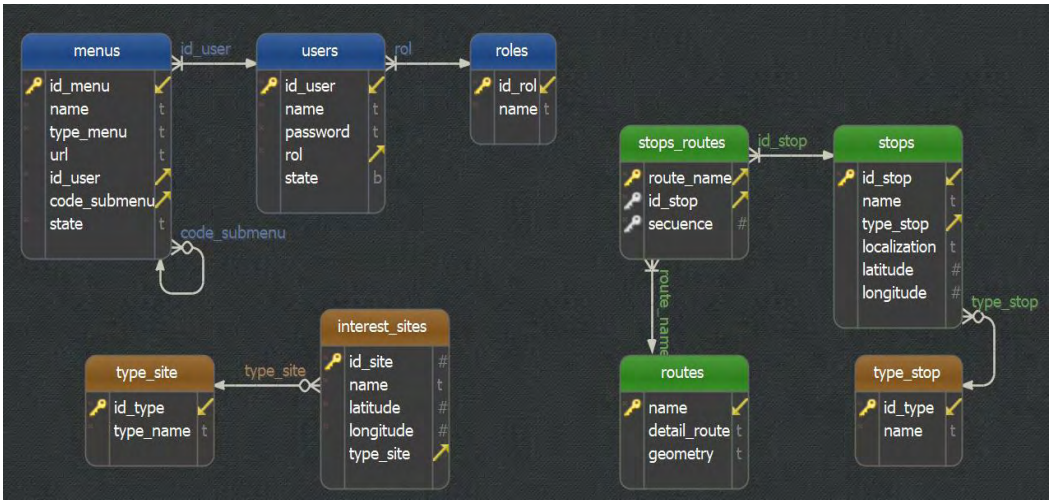
1.3. **Capa de persistencia**

En esta capa se encuentra la base de datos de SITApp, que alberga toda la información georreferenciada acerca de las rutas y paraderos del SETP y los

sitios de interés más relevantes de la ciudad de Pasto, aquí también se encuentran las clases encargadas de mapear este modelo tanto en SITApp como en el Web Service.

A continuación se muestra el diagrama entidad relación de la base de datos de SITApp.

Figura 25. Diagrama entidad relación de SITApp.



A continuación se describen los paquetes encargados de mapear el modelo de la base de datos.

Figura 26. Clases del paquete models de SITApp

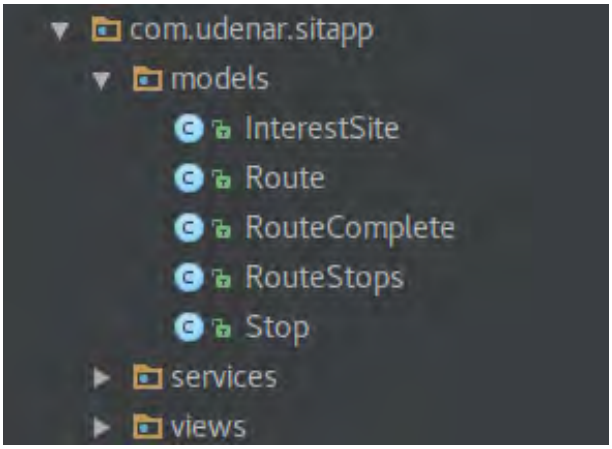
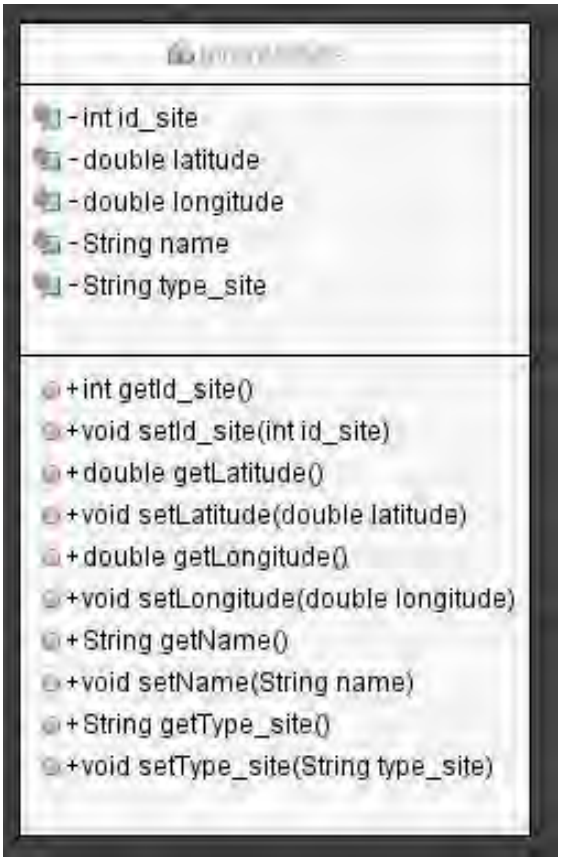
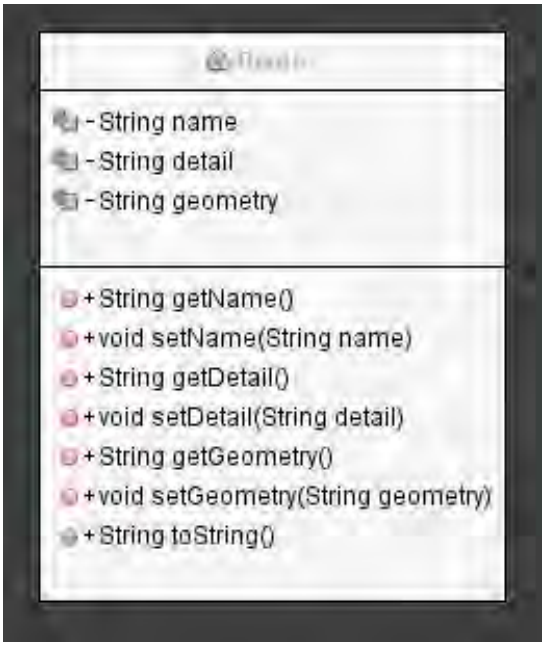


Figura 27. Clase InterestSite.java



Clase que representa la tabla InterestSites de la base de datos.

Figura 28. Clase Route.java



Clase que representa la tabla Routes de la base de datos.

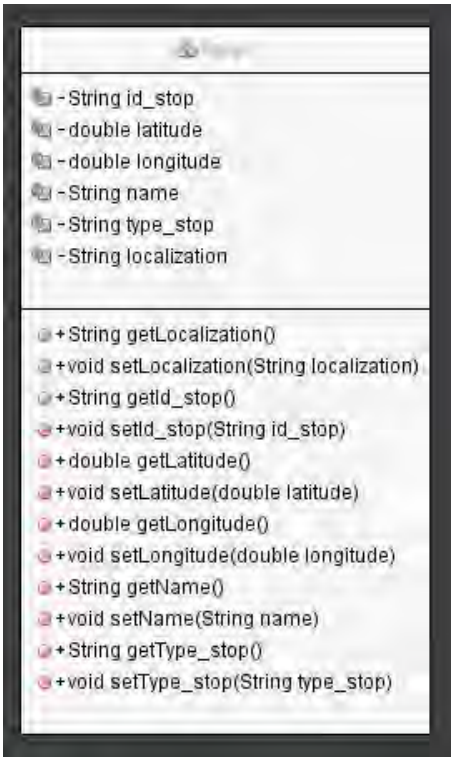
Figura 29. Clase RouteComplete.java

Clase que representa la ruta con todos sus paraderos.

Figura 30. Clase RouteStops.java

Clase que representa la ruta los paraderos de origen y destino.

Figura 31. Clase Stop.java



Clase que representa la tabla Stops de la base de datos.

Figura 32. Clases del paquete model del SITApp Web Service

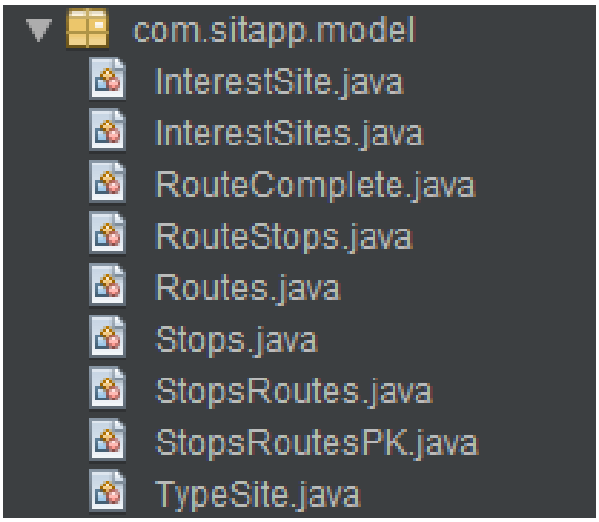
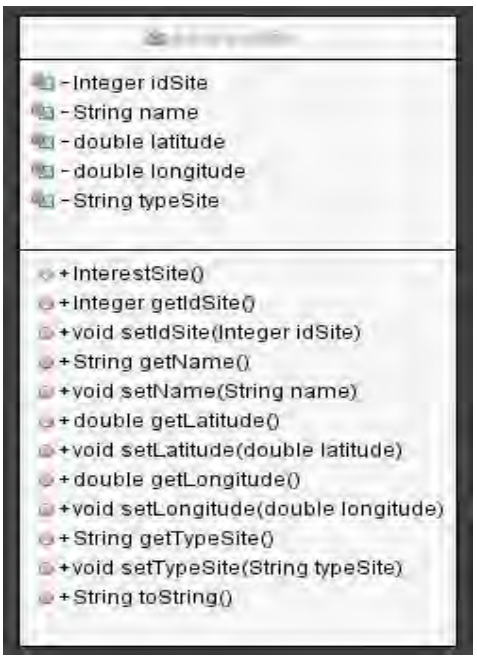


Figura 33. Clase InterestSite.java

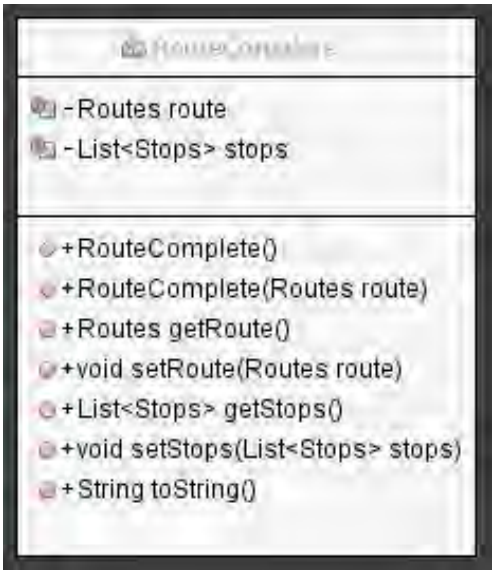


Esta clase es una adaptación de la clase InterestSites, fue desarrollada con el objetivo de enviar la menor cantidad de datos en el archivo JSON hacia SITApp.

Figura 34. Clase InterestSites.java

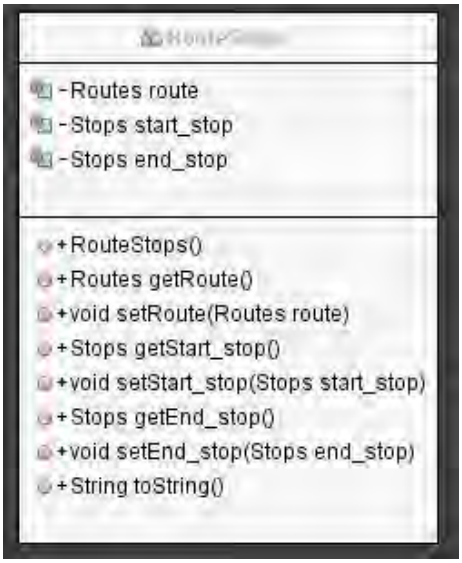
Clase que representa la tabla InterestSites de la base de datos de SITApp.

Figura 35. Clase RouteComplete.java



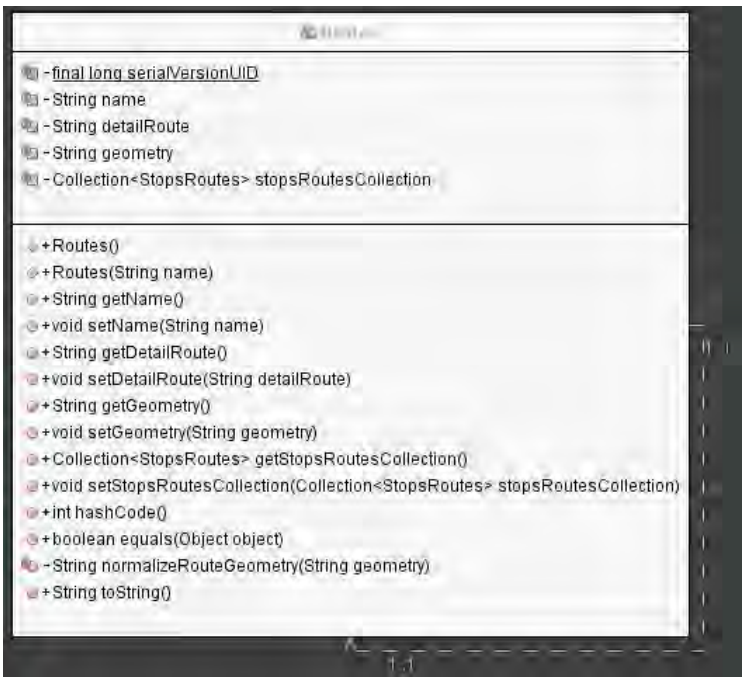
Clase que representa una ruta con todos sus paraderos asociados, fue desarrollada con el objetivo de enviar la menor cantidad de datos en el archivo JSON hacia SITApp.

Figura 36. Clase RouteStops.java



Clase que representa a la ruta con el paradero de origen y destino, fue desarrollada con el objetivo de enviar la menor cantidad de datos en el archivo JSON hacia SITApp.

Figura 37. Clase Routes.java

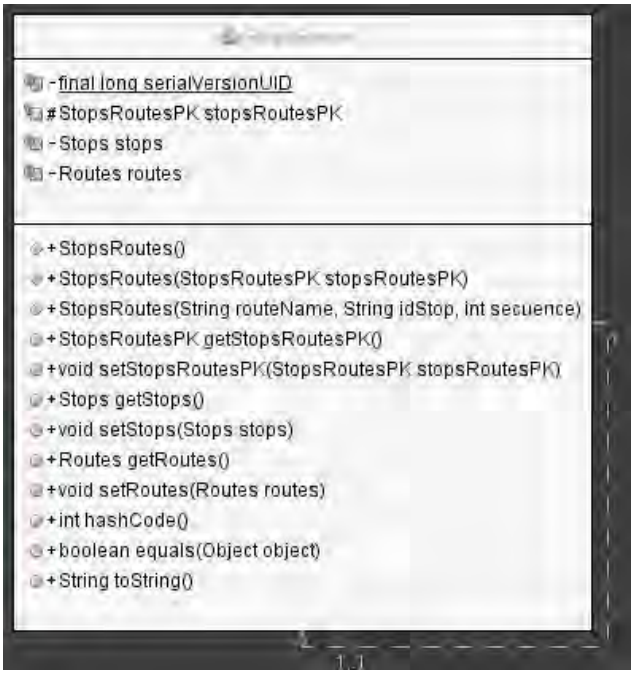


Clase que representa a la tabla Routes de la base de datos de SITApp.

Figura 38. Clase Stops.java

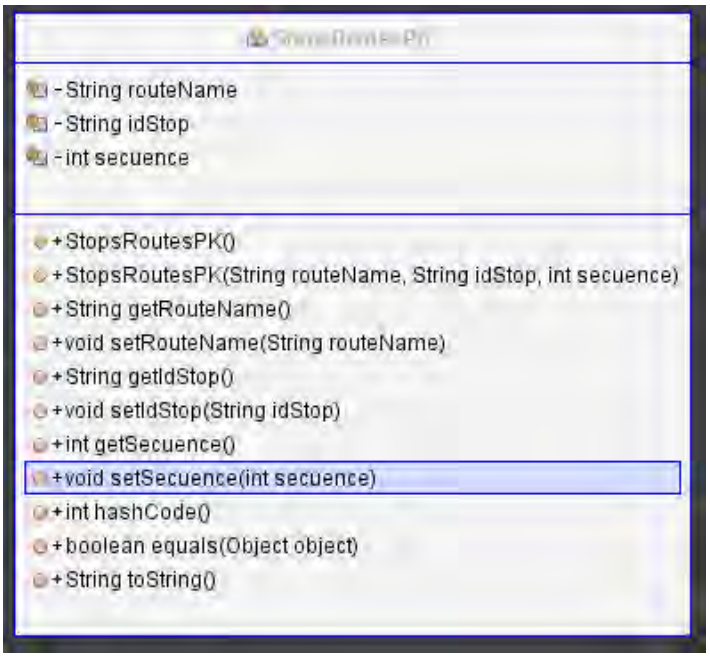
Clase que representa la tabla Stops de la base de datos de SITApp.

Figura 39. Clase RouteStops.java



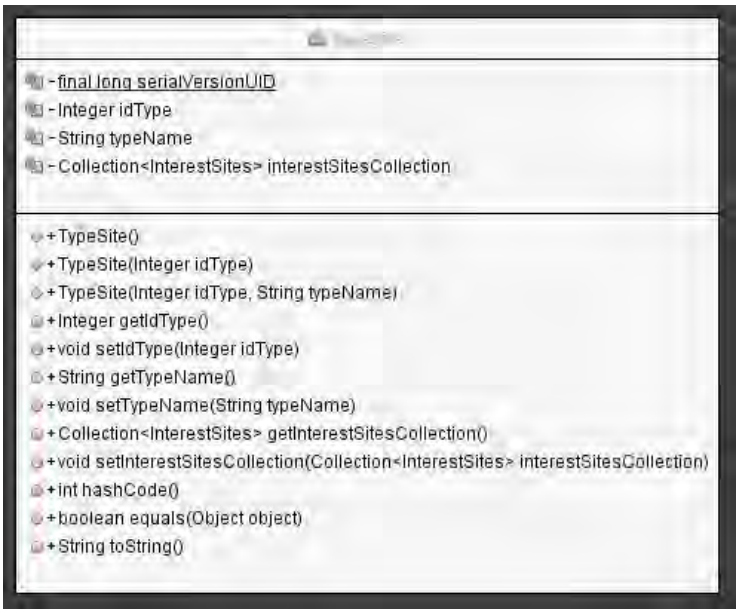
Clase que representa la tabla intermedia que conecta las rutas con cada uno de sus paraderos llamada RouteStops en la base de datos de SITApp.

Figura 40. Clase RouteStopsPK.java



Clase creada por Java EE para representar la llave primaria de la clase intermedia StopsRoutes de la base de datos de SITApp.

Figura 41. Clase TypeSite.java



Clase que representa la tabla TypeSites de la base de datos de SITApp.

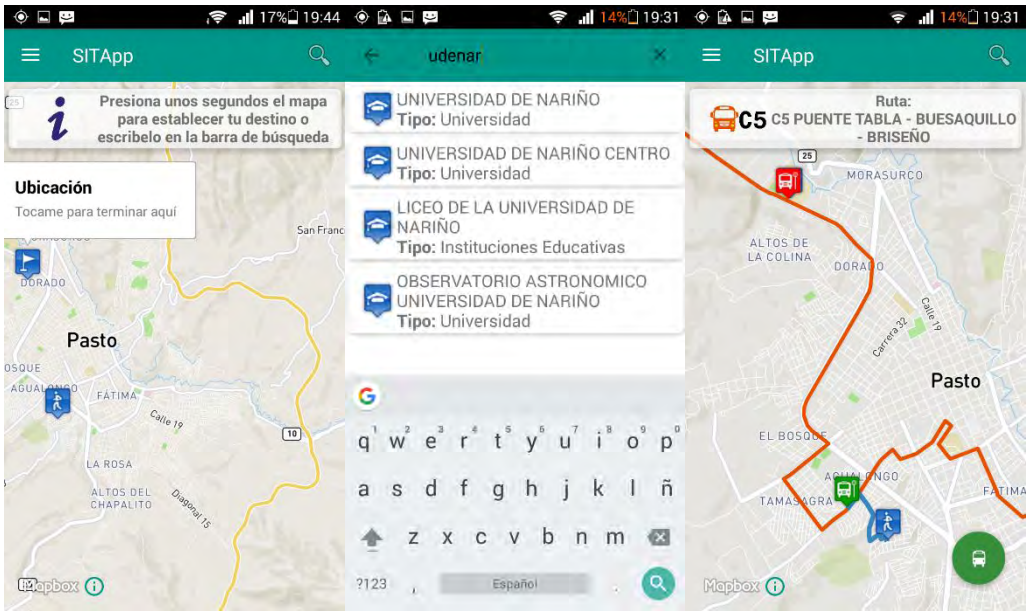
2. Funcionalidades SITApp Android

A continuación se describen las funcionalidades de SITApp Android:

2.1. Búsqueda por un sitio de interés

Esta funcionalidad permite al usuario realizar la búsqueda de rutas mediante la posición GPS del dispositivo y un sitio de interés en la ciudad, este sitio puede ser buscando usando el buscador semántico o presionando prolongadamente el mapa para establecerlo.

Figura 41. Ejemplo búsqueda mediante GPS y un sitio de interés.



El ciclo de vida normal de esta funcionalidad pasa principalmente por la clase **MainActivity.java** que se encuentra en el paquete **com.udenar.sitapp.views** descrito anteriormente.

Cuando el usuario usa el buscador semántico la aplicación usa el escuchador de eventos del componente SearchView como se muestra a continuación:

```
searchView.setOnQueryTextListener(new SearchView.OnQueryTextListener() {
    // When user sends query
    @Override
    public boolean onQueryTextSubmit(String query) {
        // Código omitido
```

```
        launchSitesService(query);
        return true;
```

Este código llama al método **lauchSitesService()** que se encarga de establecer comunicación con el WebService utilizando la librería retrofit como se muestra a continuación:

```
public void launchSitesService(String query) {
    // Build Retrofit to support web service communication
    final Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(web_service)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    // Create InterestSite service
    InterestSitesService service =
retrofit.create(InterestSitesService.class);
    // Send request to SITApp Web Service
    Call<List<InterestSite>> call = service.getSites(query);
    // Bring results from request
    call.enqueue(new Callback<List<InterestSite>>() {
        @Override
        public void onResponse(Call<List<InterestSite>> call,
                               Response<List<InterestSite>> response) {
            // Get InterestSites list
            List<InterestSite> sites = response.body();
            // If there's results
            if (sites.size() > 0) {
                // Closing keyboard
                activity.closeKeyboard();
                // Get Fragment Manager
                final FragmentTransaction transaction =
                        getSupportFragmentManager().beginTransaction();

                // Create Sites Fragment
                sitesFragment = new InterestSitesFragment();
                sitesFragment.setSites(sites);
                sitesFragment.setActivity(activity);

                // Add Sites Fragment to layout container
                transaction.replace(R.id.container, sitesFragment,
TAG_SITE_FRAGMENT);
                transaction.commit();
            }
            // If there's no results
            else {
                Toast.makeText(activity, getText(R.string.no_results),
Toast.LENGTH_LONG).show();
            }
```

```
        }
}
```

Si hay respuesta por parte del Web Service esto significa que encontró uno o más sitios de interés, por lo tanto haciendo uso de fragments, reemplaza el mapa por el listado de sitios de interés encontrados, si el usuario selecciona uno de estos sitios como su lugar de destino se dispara el método **onClick()** de la clase **InterestSitesHolder.java,** a continuación el fragmento de código del método para esta opción de búsqueda:

```
if (context.getState() == 0) {
    if (context.getMyLocation().getPosition() != null) {
        context.getStart_point()
                .title(context.getMyLocation().getTitle())
                .position(context.getMyLocation().getPosition())
                .icon(context.getMyLocation().getIcon());
        context.getEnd_point().title(site.getName())
                .position(new LatLng(site.getLatitude(),
                        site.getLongitude()))
                .snippet("Tipo: : " + site.getType_site())
                .icon(icon);
        context.setLast_state(0);
        context.launchRouteService();
    } else {
        // ALERTA DE ERROR CÓDIGO OMITIDO
    }
}
```

El código establece el origen y destino para la búsqueda de rutas y posteriormente llama al método **lauchRouteService()** que se encuentra en la clase **MainActivity.java.** Este método realiza la petición al Web Service y si hay una respuesta significa que encontró una o más rutas, por lo tanto haciendo uso de fragments muestra el mapa con la ruta creada, a continuación el fragmento de código que realiza esta operación:

```
public void launchRouteService() {
    reloadMapBoxFragment();
    final Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(web_service)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    RouteService service = retrofit.create(RouteService.class);
    Call<List<RouteStops>> call =
service.getRoute(start_point.getPosition().getLatitude(),
            start_point.getPosition().getLongitude(),
            end_point.getPosition().getLatitude(),
            end_point.getPosition().getLongitude());
    //Toast.makeText(activity, getString(R.string.search_route),
Toast.LENGTH_SHORT).show();
```

```
    call.enqueue(new Callback<List<RouteStops>>() {
        @Override
        public void onResponse(Call<List<RouteStops>> call,
Response<List<RouteStops>> response) {
            // Get result
            routes = response.body();
            // If there aren't Routes
            if (routes == null) {
                Toast.makeText(activity, R.string.no_route_found,
Toast.LENGTH_LONG).show();
                setState(0);
                setLast_state(0);
                searchItem.collapseActionView();
            }
            // If some Route was found
            else {
                final_route = routes.get(0);
                createRoute(final_route);
                if (map != null) {
                    map.clear();
                    map.addMarker(start_point);
                    map.addMarker(end_point);
                    map.addMarker(start_stop);
                    map.addMarker(end_stop);
                    map.addPolyline(route);
                    activity.drawDirections(
                            start_point.getPosition(),
                            start_stop.getPosition());
                    CameraPosition position = new CameraPosition.Builder()
                            .target(start_point.getPosition())
                            .zoom(13)
                            .build();
                    map.animateCamera(CameraUpdateFactory
                            .newCameraPosition(position), 10000);
}
//CÓDIGO OMITIDO
```

El método **createRoute()** pertenece a la clase **MainActivity.java** y se encarga de tomar los atributos del objeto **routeStops** los cuales son la geometría del recorrido de la ruta y los dos paraderos, paradero inicial y final, e instanciarlos como una poli-línea y dos marcadores para posteriormente graficarlos en el mapa, a continuación se muestra el código de este método:

```
public void createRoute(RouteStops routeStops) {
    activity.cardView.setVisibility(View.VISIBLE);
    activity.setCardViewResources(routeStops.getRoute().getDetail());
    // Create an Icon object for the marker to use
    IconFactory iconFactory = IconFactory.getInstance(activity);
    Icon icon = iconFactory.fromResource(R.drawable.busstopi);
```

```
    // Set attributes to start_stop from routeStops
    start_stop.title(routeStops.getStart_stop().getName())
            .snippet(routeStops.getStart_stop().getLocalization())
            .position(new LatLng(routeStops.getStart_stop().getLatitude(),
                    routeStops.getStart_stop().getLongitude()))
            .icon(icon);
    icon = iconFactory.fromResource(R.drawable.busstopf);
    // Set attributes to end_stop from routeStops
    end_stop.title(routeStops.getEnd_stop().getName())
            .snippet(routeStops.getEnd_stop().getLocalization())
            .position(new LatLng(routeStops.getEnd_stop().getLatitude(),
                    routeStops.getEnd_stop().getLongitude()))
            .icon(icon);
    route = createRoutePolyline(routeStops.getRoute());
}
```

Al recargarse el fragment que contiene al mapa, este hace que se active el método **onMapReady()** que se encuentra en la clase **MainActivity.java**. Este método es el encargado de graficar los marcadores y las poli-líneas en el mapa y además grafica las indicaciones que el usuario debe seguir para llegar desde su posición GPS hasta el primer paradero, a continuación el fragmento de código que realiza esta operación:

```
// If some route was found then we paint all markers and polylines in map
if (activity.getState() == 4
        && start_point.getPosition() != null
        && end_stop.getPosition() != null
        && start_stop.getPosition() != null
        && end_point.getPosition() != null) {
    map.clear();
    map.addMarker(start_point);
    map.addMarker(end_point);
    map.addMarker(start_stop);
    map.addMarker(end_stop);
    map.addPolyline(route);
    //Draw directions
    if (last_state == 0) {
        activity.drawDirections(start_point.getMarker().getPosition(),
                start_stop.getMarker().getPosition());
    } else {
        activity.drawDirections(start_point.getMarker().getPosition(),
                start_stop.getMarker().getPosition());
        activity.drawDirections(end_stop.getMarker().getPosition(),
                end_point.getMarker().getPosition());
    }
    CameraPosition position = new CameraPosition.Builder()
            .target(start_stop.getPosition())
            .zoom(13)
            .build();
```

```
    map.setCameraPosition(position);
    position = new CameraPosition.Builder()
            .target(start_point.getPosition())
            .zoom(13)
            .build();
    map.animateCamera(CameraUpdateFactory
            .newCameraPosition(position), 10000);
}
```

Si el usuario opta por establecer su destino presionando prolongadamente el mapa se activa el método **onMapLongClick()** que se encuentra en la clase **MainActivity.java**, a continuación un fragmento del código de este método:

```
map.setOnMapLongClickListener(new
MapboxMap.OnMapLongClickListener() {
    @Override
    public void onMapLongClick(@NonNull LatLng point) {
        if (activity.getState() == 0) {
            // CÓDIGO OMITIDO
            end_point.position(point);
            end_point.title(getString(R.string.point));
            end_point.snippet(getString(R.string.end_travel));
            IconFactory iconFactory = IconFactory.getInstance(activity);
            Icon icon = iconFactory.fromResource(R.drawable.site);
            end_point.getMarker().setIcon(icon);
            map.addMarker(end_point);
            map.selectMarker(end_point.getMarker());
            map.setOnInfoWindowClickListener(
                    new MapboxMap.OnInfoWindowClickListener() {
                        @Override
                        public boolean onInfoWindowClick(@NonNull Marker
marker) {
                            if
(map.getSelectedMarkers().contains(end_point.getMarker())) {
                                end_point.title("Destino");
                                        activity.launchRouteService();
                                return true;
                            } else {
                                return false;
                            }
                        }});}
```

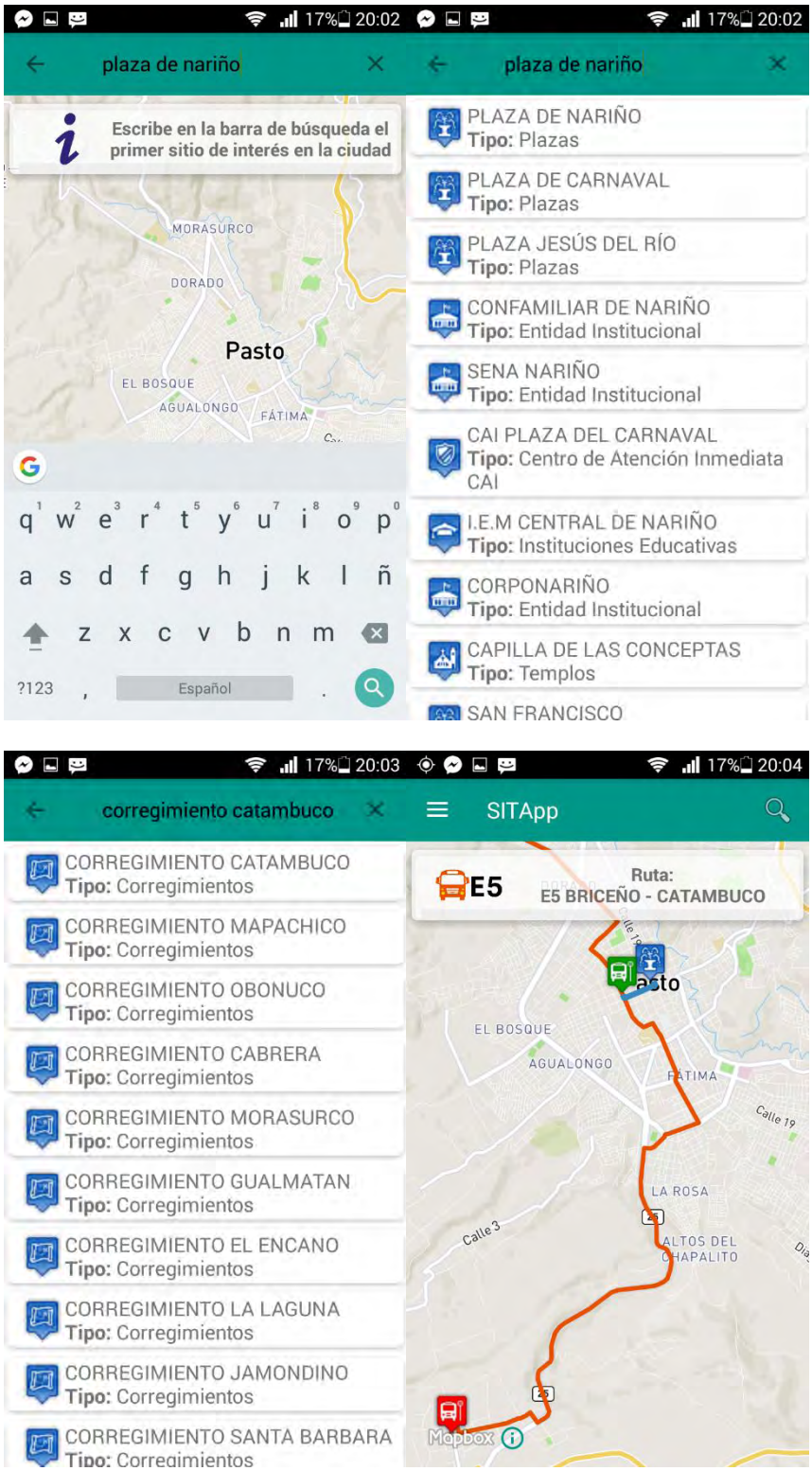
Una vez se presiona la etiqueta que aparece encima del marcador en el mapa se establece el destino y se llama al método **launchRouteService()** de la clase **MainActivity.java** que ya fue descrito anteriormente.

2.2. Búsqueda por dos sitios de interés

Esta funcionalidad permite al usuario realizar una búsqueda de dos sitios de interés haciendo uso del buscador semántico para establecer el punto de

origen y el punto de destino, en la Figura 42 se puede observar un ejemplo de esta funcionalidad:

Figura 42. Ejemplo búsqueda mediante dos sitios de interés



Al igual que la función anterior se utiliza el buscador semántico para realizar la búsqueda de los dos sitios, pero cuando el usuario selecciona el sitio de interés del listado recibido, se dispara el siguiente fragmento de código del método **onClick()** de la clase **InterestSiteHolder()**:

```
// If search by two sites is enabled, for first site
else if (context.getState() == 1) {
    context.getStart_point().title(site.getName())
            .position(new LatLng(site.getLatitude(),
                    site.getLongitude()))
            .snippet("Tipo: " + site.getType_site())
            .icon(icon);
    context.setState(2);
    context.setLast_state(1);
    context.txtCard.setText("Escribe en la barra de búsqueda el segundo
sitio de interés en la ciudad");
    context.getSearchView().setQuery("", false);
    context.cardView.setVisibility(View.VISIBLE);
    /*InterestSitesAdapter adapter = new InterestSitesAdapter(context,
            R.layout.list_item_sites, new ArrayList<InterestSite>());
    context.getSitesFragment().setAdapter(adapter);*/
    context.reloadMapBoxFragment();
}
// If search by touch two sites is enabled, for second site
else if (context.getState() == 2) {
    context.getEnd_point().title(site.getName())
            .position(new LatLng(site.getLatitude(),
                    site.getLongitude()))
            .snippet("Tipo: " + site.getType_site())
            .icon(icon);
    context.setLast_state(2);
    context.launchRouteService();
}
```

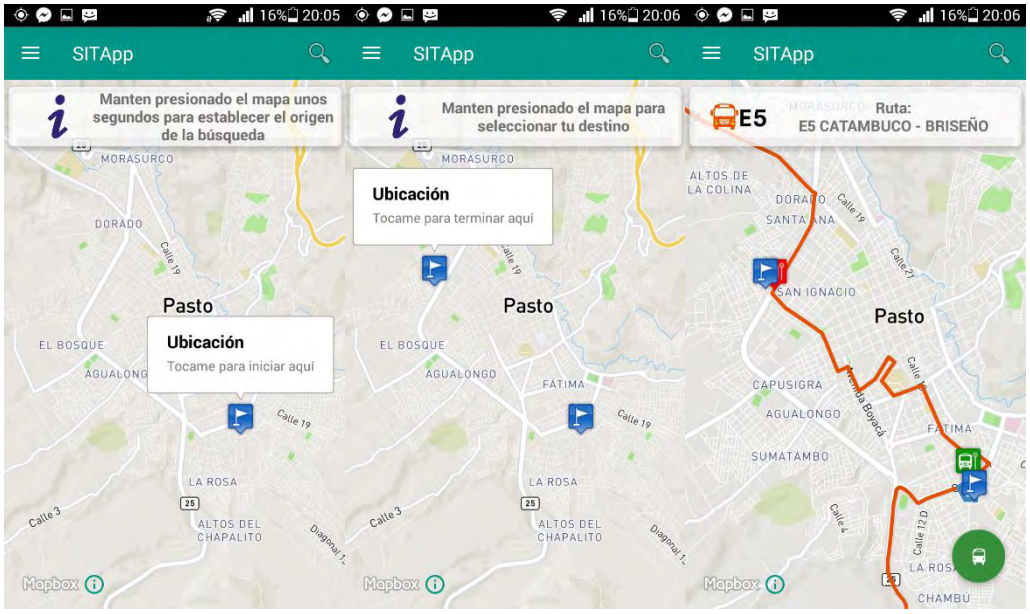
Cuando el estado de la aplicación es 1, esto significa que está buscando el primer sitio de interés por lo tanto al ser seleccionado este primer sitio se cambia el estado a 2, se establece el punto de origen de la búsqueda y se recarga el mapa para mostrarle al usuario gráficamente cual fue su selección, ahora el usuario debe buscar el segundo sitio, vuelve a realizarse el proceso de búsqueda y cuando el segundo sitio es seleccionado se dispara la segunda parte del método cuando el estado es 2, se establece el punto de destino de la búsqueda y se llama al método **launchRouteService()** de la clase **MainActivity.java** que se encarga de comunicarse con el Web Service para buscar si hay una o más rutas que lleven al usuario desde el origen hasta el

destino establecido. Este proceso de búsqueda de rutas se realiza de la misma manera que la funcionalidad anterior.

2.3. **Búsqueda por tocar dos puntos**

Esta funcionalidad permite al usuario establecer su origen y su destino en la ciudad presionando prolongadamente el mapa en la aplicación, en la Figura 43 se puede observar un ejemplo de la funcionalidad:

Figura 43. Ejemplo de búsqueda tocando dos puntos en el mapa



A continuación un fragmento del método **setOnMapLongClickListener()** que se encarga de gestionar las operaciones cuando el mapa es presionado prolongadamente, esta función se encuentra en la clase **MainActivity.java**.

```
if (activity.getState() == 3) {
    if (number_markers == 2) {
        //map.clear();
        map.deselectMarkers();
        if (start_point.getPosition() != null) {
            if (map.getMarkers().contains(start_point.getMarker())) {
                map.setOnInfoWindowClickListener(null);
                map.removeMarker(start_point.getMarker());
            }
        }
        start_point.position(point);
        start_point.title(getString(R.string.point));
        start_point.snippet(getString(R.string.start_travel));
        start_point.getMarker().setFlat(true);
```

```
        IconFactory iconFactory = IconFactory.getInstance(activity);
        Icon icon = iconFactory.fromResource(R.drawable.site);
        start_point.getMarker().setIcon(icon);
        map.selectMarker(start_point.getMarker());
        map.addMarker(start_point);
        map.setOnInfoWindowClickListener(
                new MapboxMap.OnInfoWindowClickListener() {
                    @Override
                    public boolean onInfoWindowClick(@NonNull Marker marker) {
                        start_point.snippet("Lat: "
                                + start_point.getPosition().getLatitude()
                                + "\nLong: "
                                + start_point.getPosition().getLongitude());
                        number_markers--;
                        map.setOnInfoWindowClickListener(null);
                        map.deselectMarkers();
                        activity.txtCard.setText("Manten presionado el mapa para
seleccionar tu destino");
                        return true;
                    }
                });
    } else if (number_markers == 1) {
        map.deselectMarkers();
        //map.clear();
        if (end_point.getPosition() != null) {
            if (map.getMarkers().contains(end_point.getMarker())) {
                map.removeMarker(end_point.getMarker());
            }
        }
        end_point.position(point);
        end_point.title(getString(R.string.point));
        end_point.snippet(getString(R.string.end_travel));
        IconFactory iconFactory = IconFactory.getInstance(activity);
        Icon icon = iconFactory.fromResource(R.drawable.site);
        end_point.getMarker().setIcon(icon);
        map.addMarker(end_point);
        map.selectMarker(end_point.getMarker());
        map.setOnInfoWindowClickListener(
                new MapboxMap.OnInfoWindowClickListener() {
                    @Override
                    public boolean onInfoWindowClick(@NonNull Marker marker) {
                        if (map.getSelectedMarkers().contains(end_point.getMarker())) {
                            end_point.title("Destino");
                            end_point.snippet("Lat: "
                                    + end_point.getPosition().getLatitude()
                                    + "\nLong: "
                                    + end_point.getPosition().getLongitude());
                            number_markers--;
```

```
                    activity.launchRouteService();
                    activity.setLast_state(3);
                    map.setOnInfoWindowClickListener(null);
                    return true;
                } else {
                    return false;
                }
            }
        });
    }
}
```

El método **onInfoWindowsClick()** es llamado cuando el usuario presiona el título del marcador para establecer el origen inicialmente, este marcador se fija y el usuario tiene la posibilidad de presionar nuevamente la pantalla prolongadamente para establecer su destino de la misma manera que el origen. Una vez fijado el destino se llama a la función **launchRouteService()** de la clase **MainActivity.java** que se encarga de comunicarse con el Web Service para buscar si hay una o más rutas que lleven al usuario desde el origen hasta el destino establecido, que funciona de la misma manera que en las dos opciones anteriores de búsqueda.

2.4. **Mostrar recorridos de rutas**

Esta funcionalidad permite al usuario saber los recorridos de las rutas y todos sus paraderos autorizados. La búsqueda está discriminada por rutas estratégicas o rutas complementarias, en la Figura 44 se puede observar un ejemplo de esta funcionalidad.

Figura 44: Ruta C9, recorrido y parada

A continuación un fragmento del método **getStrategicsComplementariesRoutes()** encargado de gestionar la peticion hacia el Web Service de SITApp recibiendo como respuesta un listado de rutas con sus diferentes paraderos.

```
private void getStrategicsComplementariesRoutes(String search) {

    final Retrofit retrofit = new Retrofit.Builder()
            .baseUrl(web_service)
            .addConverterFactory(GsonConverterFactory.create())
            .build();
    TypeRouteService service = retrofit.create(TypeRouteService.class);
```

```
    Call<List<RouteComplete>> call = service.getRoutes(search);
    //Toast.makeText(activity, getString(R.string.search_route),
Toast.LENGTH_SHORT).show();

    call.enqueue(new Callback<List<RouteComplete>>() {
        @Override
        public void onResponse(Call<List<RouteComplete>> call,
Response<List<RouteComplete>> response) {
            List<RouteComplete> routes = response.body();
            if (!routes.isEmpty()) {
                final FragmentTransaction transaction =
                        getSupportFragmentManager().beginTransaction();

                // Create Routes Complete Fragment
                routesCompleteFragment = new RoutesCompleteFragment();
                routesCompleteFragment.setRoutes(routes);
                routesCompleteFragment.setActivity(activity);

                // Add Sites Fragment to layout container
                transaction.replace(R.id.container, routesCompleteFragment,
TAG_ROUTES_COMPLETE_FRAGMENT);
                transaction.commit();
            }
        }

        @Override
        public void onFailure(Call<List<RouteComplete>> call, Throwable t)
{

        }
    });
}
```

Se crea y se ubica como fragment principal al fragment **routesCompleteFragment** que se encarga de mostrar el listado de rutas al usuario, cuando es seleccionada una de las rutas es llamado el método **onClick()** de la clase **RoutesCompleteHolder** que se muestra a continuación:

```
public void onClick(View view) {

    // 5. Handle the onClick event for the ViewHolder
    if (this.routeStops != null) {
        IconFactory iconFactory = IconFactory.getInstance(context);
        Icon icon = iconFactory.fromResource(R.drawable.busstop);
        context.getRoute_stops().clear();
        for (int i = 1; i < routeStops.getStops().size(); i++) {
            if (i < routeStops.getStops().size() - 1) {
                context.getRoute_stops().add(
                        new MarkerViewOptions()
```

```
                        .title(routeStops.getStops().get(i).getName())
                        .position(new
LatLng(routeStops.getStops().get(i).getLatitude(),
                                routeStops.getStops().get(i).getLongitude()))

.snippet(routeStops.getStops().get(i).getLocalization())
                        .icon(icon)
            );
        } else {
            icon = iconFactory.fromResource(R.drawable.busstopf);
            context.getRoute_stops().add(
                    new MarkerViewOptions()
                        .title(routeStops.getStops().get(i).getName())
                        .position(new
LatLng(routeStops.getStops().get(i).getLatitude(),
                                routeStops.getStops().get(i).getLongitude()))

.snippet(routeStops.getStops().get(i).getLocalization())
                        .icon(icon)
            );
        }
    }
    icon = iconFactory.fromResource(R.drawable.busstopi);
    context.getRoute_stops().add(
            new MarkerViewOptions()
                .title(routeStops.getStops().get(0).getName())
                .position(new
LatLng(routeStops.getStops().get(0).getLatitude(),
                        routeStops.getStops().get(0).getLongitude()))
                .snippet(routeStops.getStops().get(0).getLocalization())
                .icon(icon)
    );
    context.setLast_state(5);

context.setRoute(context.createRoutePolyline(routeStops.getRoute()));
    context.setCardViewResources(routeStops.getRoute().getDetail());
    context.reloadMapBoxFragment();
    context.cardView.setVisibility(View.VISIBLE);
  }
}
```

Al final de este método se recarga el fragment de Mapbox e inmediatamente se ejecuta el método asíncrono **onMapReady()**, específicamente se ejecuta el siguiente fragmento de código encargado de mostrar en el mapa el recorrido de la ruta y cada uno de los paraderos autorizados.

```
public void onMapReady(MapboxMap mapboxMap) {

  //Código omitido ...
```

```
    else if ((activity.getState() == 5 || activity.getState() == 6)
            && activity.getLast_state() == 5) {
        map.clear();
        for (MarkerViewOptions marker : activity.getRoute_stops()) {
            map.addMarker(marker);
        }
        map.addPolyline(route);
        CameraPosition position = new CameraPosition.Builder()
                .target(new LatLng(1.2084236, -77.2787569))
                .zoom(12)
                .build();
        map.setCameraPosition(position);
    }
}
```

3. Algoritmos de SITApp Web Service

SITApp Web Service alberga los algoritmos de conexión y consulta hacia la base de datos, la ontología, además los algoritmos de inferencia y de cruce de rutas.

3.1. Algoritmo de conexión y búsqueda de sitios en ontología

La clase **ConectionOntology.java** es la encargada de gestionar las conexiones y las consultas hacia la ontología, a continuación el código fuente de esta clase.

```
public class ConectionOntology {

    Query query;
    QueryExecution qexec;
    OntModel model;

    public OntModel getModel() {
        return model;
    }

    public void setModel(OntModel model) {
        this.model = model;
    }

    public OntModel start() {
        model = ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
        InputStream in = FileManager.get().open("/home/user/sitapp/sitapp.owl");
        if (in == null) {
            throw new IllegalArgumentException("Archivo No Encontrado");
```

```
        }
        model.read(in, "");
        return model;
    }

    public ResultSet consult(String consulta) {
        query = QueryFactory.create(consulta);
        qexec = QueryExecutionFactory.create(query, model);
        ResultSet results = qexec.execSelect();
        return results;
    }

    public void stop() {
        qexec.close();
    }
}
```

La clase **OntologyQueries.java** posee el algoritmo de búsqueda e inferencia en SPARQL.

```
public class OntologyQueries {

    private ConectionOntology ontologia;

    public OntologyQueries(ConectionOntology ontologia) {
        this.ontologia = ontologia;
    }

    public ConectionOntology getConectionOntology() {
        return ontologia;
    }

    public void setConectionOntology(ConectionOntology ontologia) {
        this.ontologia = ontologia;
    }


    public List<String> searchSites(String query){
        List<String> sites = new ArrayList<>();
        String[] words = query.split(" ");
        String filter = "";
        for (int i = 0; i < words.length ; i++){
            if (i != words.length-1){
                filter += "REGEX (?sin, \"" + words[i] + "\")||"
                        + "REGEX (?pal, \"" + words[i] + "\")||";

            } else {
                filter += "REGEX (?sin, \"" + words[i] + "\")||"
```

```
                + "REGEX (?pal, \"" + words[i] + "\")";

            }
        }
        String sparql = "PREFIX ont:<http://www.semanticweb.org/sitapp.owl#>\n"
                + "SELECT distinct ?id\n"
                + "where\n"
                + "{\n"
                + "?Sitios_interes ont:id ?id.\n"
                + "?Sitios_interes ont:sinonimos ?sin.\n"
                + "?Sitios_interes ont:es_descrito_por ?keyword.\n"
                + "?keyword ont:palabra ?pal.\n"
                + "FILTER (\n"
                + filter
                + ").\n"
                + "}\n"
                + "group by ?id ?sin ?keyword ?pal\n"
                + "order by ?id";
        ResultSet rs = ontologia.consult(sparql);
        while (rs.hasNext()){
            QuerySolution sol = rs.nextSolution();
            sites.add(sol.get("?id").toString().
                    replace("^^http://www.w3.org/2001/XMLSchema#string", ""));
        }
        return sites;
    }
}
```

La consulta SPARQL relaciona las entidades Sitios Interés y Palabras Clave mediante las relaciones definidas en el modelo, internamente la ontología busca todas las coincidencias en la consulta y retorna como resultado un listado de sitios de interés según los criterios de búsqueda.

3.2. Servicio de búsqueda de sitios de interés

Este servicio web se encuentra en la clase **InterestSitesFacadeREST.java**, encargado de recibir la consulta enviada desde SITApp Android y llamar a la case **OntologyQueries.java** para que se realice la inferencia en la ontología y posteriormente se consulta a la base de datos el listado de sitios de interés retornado por la ontología con el fin de traer todos los datos georreferenciados almacenados referente a cada uno de los sitios de interés y se devuelve este listado de sitios mediante un archivo tipo JSON.

```
@GET
@Path("sites/{query}")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
public List<InterestSite> find(@PathParam("query") String query) {
```

```
		System.out.println("Web Service Llamado, Consulta: " + query);
		ConectionOntology ont = new ConectionOntology();
		ont.start();
		OntologyQueries querys = new OntologyQueries(ont);
		String query2 = query;
		query = deleteDeadWords(query);
		List<String> sitesIds = querys.searchSites(query.toUpperCase());
		ont.stop();
		Query nativeQuery;
		if (sitesIds.isEmpty()) {
			nativeQuery = em.createNativeQuery("SELECT id_site, name,
type_site, latitude, "
					+ "longitude, jarowinkler('"
					+ query2
					+ "', name ) FROM interest_sites "
					+ "order by 6 desc "
					+ "limit 10");
		} else {
			String cad = "";
			for (int i = 0; i < sitesIds.size(); i++) {
				if (i != sitesIds.size() - 1) {
					cad += sitesIds.get(i) + ",";
				} else {
					cad += sitesIds.get(i);
				}
			}
			nativeQuery = em.createNativeQuery("SELECT id_site, name,
type_site"
					+ ", latitude, longitude, jarowinkler('"
					+ query2
					+ "', name ) FROM interest_sites "
					+ "WHERE id_site in ("
					+ cad
					+ ") order by 6 desc "
					+ "limit 30;");
		}
		List<Objects[]> sitesInit = nativeQuery.getResultList();
		List<InterestSite> sites = new ArrayList();
		for (Object[] o : sitesInit) {
			System.out.print(o[0]);
			InterestSites s =
(InterestSites)em.createNamedQuery("InterestSites.findByIdSite")
					.setParameter("idSite", o[0])
					.getSingleResult();
			InterestSite is = new InterestSite();
			is.setName(s.getName());
			is.setLatitude(s.getLatitude());
			is.setLongitude(s.getLongitude());
```

```
            is.setTypeSite(s.getTypeSite().getTypeName());
            sites.add(is);
        }
        return sites;
}
```

3.3. Servicio de búsqueda por tipo de ruta

Este servicio web gestiona la búsqueda en la base de datos la información referente a las rutas, estratégicas o complementarias, y los paraderos autorizados asociados a cada una de las rutas como se muestra en el siguiente fragmento de código de la clase **RoutesFacadeREST.java**.

```
@GET
@Path("byRouteType/{routeType}")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
public List<RouteComplete> findByRouteType(@PathParam("routeType")
String routeType) {
        if (routeType.length() != 1) return null;
        Query qr;
        String sql = "SELECT name FROM routes WHERE name LIKE '" +
routeType.toUpperCase() + "%'";
        qr = em.createNativeQuery(sql);
        List<String> routes = qr.getResultList();
        if (routes.isEmpty()) {
                return null;
        }
        List<RouteComplete> finalList = new ArrayList<>();
        for (int i = 0; i < routes.size(); i++) {
                List<StopsRoutes> stopsroutes =
em.createNamedQuery("StopsRoutes.findByRouteName")
                                .setParameter("routeName", routes.get(i))
                                .getResultList();
                System.out.print(routes.get(i));
                if (!stopsroutes.isEmpty()) {
                        RouteComplete custom = new
RouteComplete(stopsroutes.get(0).getRoutes());
                        ArrayList<Stops> stops = new ArrayList<>();
                        for (StopsRoutes s : stopsroutes) {
                                stops.add(s.getStops());
                        }
                        custom.setStops(stops);
                        finalList.add(custom);
                }
        }

        return finalList;
}
```

3.4. **Algoritmo de cruce de rutas**

El algoritmo busca inicialmente 500 metros a la redonda para el punto de origen “A” y para el punto de destino “B” seleccionados por el usuario desde SITApp Android, luego analiza si alguna de las rutas posee la combinación de paraderos cercanos a los puntos “A” y “B” y retorna mediante un archivo JSON el listado de rutas que le sirven al usuario junto con dos paraderos para cada ruta, uno de origen y uno de destino.

```
@GET
@Path("route/{start_lat}/{start_long}/{end_lat}/{end_long}")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
public List<RouteStops> routeAlgorithm(
                @PathParam("start_lat") Double start_lat,
                @PathParam("start_long") Double start_long,
                @PathParam("end_lat") Double end_lat,
                @PathParam("end_long") Double end_long) {

        // Building native query to find stops near Start Point 500 metters arround
        Query query = em.createNativeQuery("SELECT id_stop FROM stops
s\n"
                        + "WHERE ST_CONTAINS(\n"
                        + "ST_BUFFER(ST_POINT( ? , ? ), 0.005), \n"
                        + "ST_POINT(s.latitude, s.longitude));");

        // Setting parameters
        query.setParameter(1, start_lat);
        query.setParameter(2, start_long);

        // Getting Result
        List<String> stops1 = query.getResultList();

        // If no results algorithm end
        if (stops1.isEmpty()) {
                return null;
        }

        // Making first condition 'stop1','stop2', ... ,'stopN'
        String first_condition = "";
        for (int i = 0; i < stops1.size(); i++) {
                if (i != stops1.size() - 1) {
                        first_condition += "'" + stops1.get(i) + "',";
                } else {
                        first_condition += "'" + stops1.get(i) + "'";
                }

        }
```

```
System.out.println("Posicion Inicio: " + start_lat + "," + start_long);
System.out.println("Condicion #1: " + first_condition);

// Building native query to find stops near End Point 500 metters arround
query = em.createNativeQuery("SELECT id_stop FROM stops s\n"
                + "WHERE ST_CONTAINS(\n"
                + "ST_BUFFER(ST_POINT( ? , ? ), 0.005), \n"
                + "ST_POINT(s.latitude, s.longitude));");

// Setting parameters
query.setParameter(1, end_lat);
query.setParameter(2, end_long);

// Getting Results
List<String> stops2 = query.getResultList();

// If no results algorithm end
if (stops2.isEmpty()) {
        return null;
}

// Making second condition 'stop1','stop2', ... ,'stopN'
String second_condition = "";
for (int i = 0; i < stops2.size(); i++) {
        if (i != stops2.size() - 1) {
                second_condition += "'" + stops2.get(i) + "',";
        } else {
                second_condition += "'" + stops2.get(i) + "'";
        }
}
System.out.println("Posicion Llegada: " + end_lat + "," + end_long);
System.out.println("Condicion #2: " + second_condition);

/**
 * Finding routes with two nearest stops, start stop and end stop.
 * getting as result, its name, start stop id, end stop id, start stop
 * sequence, end stop sequence, distance from start point to start stop
 * and distance from end point to start stop
 */
String qr = "SELECT\n"
                + "t1.route_name, t1.id_stop, t2.id_stop, t1.secuence,"
                + "t2.secuence,"
                + "ST_DISTANCE(ST_POINT( ? , ? ),"
                + "ST_POINT(t1.latitude,t1.longitude)) + "
                + "ST_DISTANCE(ST_POINT( ? , ? ),"
                + "ST_POINT(t2.latitude,t2.longitude)) "
                + "FROM\n"
```

```
                        + "(SELECT route_name, s.id_stop, latitude, longitude,
secuence "
                        + "FROM stops_routes sr JOIN stops s\n"
                        + "ON sr.id_stop=s.id_stop\n"
                        + "WHERE\n"
                        + "s.id_stop in\n"
                        + "( " + first_condition + " )"
                        + ")t1 JOIN\n"
                        + "(SELECT route_name, s.id_stop, latitude, longitude,
secuence "
                        + "FROM stops_routes sr JOIN stops s\n"
                        + "ON sr.id_stop=s.id_stop\n"
                        + "WHERE\n"
                        + "s.id_stop in "
                        + "( " + second_condition + " ))t2\n"
                        + "ON t1.route_name=t2.route_name AND t1.id_stop !=
t2.id_stop\n"
                        + "WHERE t1.secuence<t2.secuence\n"
                        + "ORDER BY 6;";
        System.out.print(qr);

        // Building native query
        query = em.createNativeQuery(qr);
        query.setParameter(1, start_lat);
        query.setParameter(2, start_long);
        query.setParameter(3, end_lat);
        query.setParameter(4, end_long);

        //List<Object[]> final_result = query.getResultList();
        // Getting result
        List<Object[]> result = query.getResultList();
        System.out.print(result.size());

        // If there are results
        if (!result.isEmpty()) {
                List<Object[]> oneResutPerRoute = new ArrayList<>();
                List<String> routesNames = new ArrayList<>();
                // Algorithm to get only a single result by route and the best
                // of each one
                for (int i = 0; i < result.size(); i++) {
                        if (i == 0) {
                                oneResutPerRoute.add(result.get(i));
                                routesNames.add(result.get(i)[0].toString());
                        } else if (!routesNames.contains(result.get(i)[0].toString()))
{
                                oneResutPerRoute.add(result.get(i));
                                routesNames.add(result.get(i)[0].toString());
                        }
```

```
                }

                List<RouteStops> routeFinal = new ArrayList<>();

                // Building final list result
                for (Object[] o : oneResutPerRoute) {
                        // Getting route by name
                        List<Routes> route =
em.createNamedQuery("Routes.findByName")
                                        .setParameter("name", o[0])
                                        .getResultList();
                        // Getting start stop by id
                        List<Stops> stop1 =
em.createNamedQuery("Stops.findByIdStop")
                                        .setParameter("idStop", o[1])
                                        .getResultList();
                        // Getting end stop by id
                        List<Stops> stop2 =
em.createNamedQuery("Stops.findByIdStop")
                                        .setParameter("idStop", o[2])
                                        .getResultList();
                        RouteStops routeFinalitem = new RouteStops();
                        routeFinalitem.setRoute(route.get(0));
                        routeFinalitem.setStart_stop(stop1.get(0));
                        routeFinalitem.setEnd_stop(stop2.get(0));
                        routeFinal.add(routeFinalitem);
                }
                // Return result

                return routeFinal;
        } else {

                return null;
        }
}
```

ANEXO E. Manual de programador Módulo Administrativo SITApp

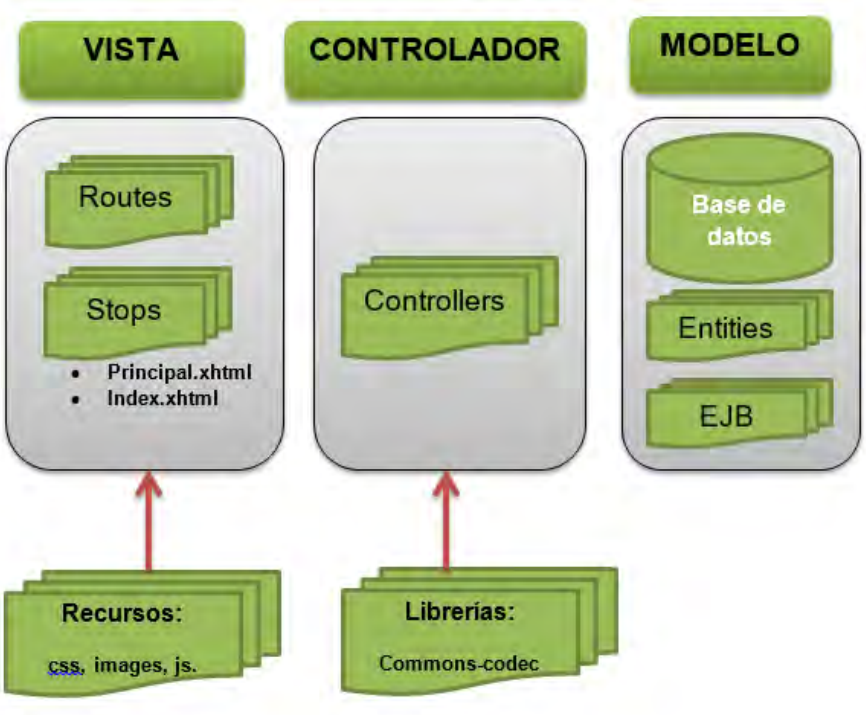
El presente documento es una guía para los programadores que hacen uso del módulo administrativo de la aplicación móvil denominada SITApp, es un manual práctico para la compresión y descripción de las principales funcionalidades que tiene este módulo.

La construcción del módulo administrativo fue realizada en su totalidad bajo el sistema operativo Linux Mint 18.1 KDE Edition. El lenguaje de programación utilizado fue Java, en específico se hizo uso del framework Java Server faces (jsf), primefaces, javascript, css, omnifaces, commons-codec y mapbox para el manejo de mapas, el entorno de desarrollo empleado fue Netbeans versión 8.1. El sistema está montado sobre el servidor de aplicaciones Glassfish versión 4.1.

4. Arquitectura

Este módulo está desarrollado bajo el modelo MVC (Modelo Vista Controlador). En la Figura 1, se puede observar la arquitectura.

Figura 1: Arquitectura módulo administrativo SITApp.



A continuación se describe cada uno de los elementos de la arquitectura del módulo administrativo de SITApp.

- ✓ **Vista.**

Permite la interacción del usuario con el usuario final, en donde se presenta la vista con el menú de opciones para las distintas opciones disponibles, los datos ingresados por el usuario son procesados por el paquete controlador y los resultados se muestran al usuario haciendo uso de etiquetas de primefaces y un mapa de mapbox. Este paquete está compuesto por páginas con extensión xhtml de primefaces y hace uso de algunos recursos como imágenes, archivos css, javascript, además este paquete hace uso de una librería llamada omnifaces la cual permite la transformación de datos primitivos a objetos java.

- ✓ **Controlador**.

Contiene la lógica de negocios de la aplicación. Se encarga de validar los datos ingresados por el usuario y procesarlos para dar respuesta a las diferentes peticiones. Permite la conexión con la base de datos y hace uso de métodos especializados para realizar operaciones CRUD (Create, Read, Update and Delete) en la base de datos. Por cada archivo con extensión xhtml existe una clase Java que gestiona las peticiones realizadas por esa vista. Además este paquete hace uso de una librería llamada commons-codec, la cual es utilizada para encriptar la contraseña del usuario.

A continuación se realiza una descripción de las clases que componen este paquete.

Figura 2. Clases del paquete Controlador del módulo administrativo de SITApp.

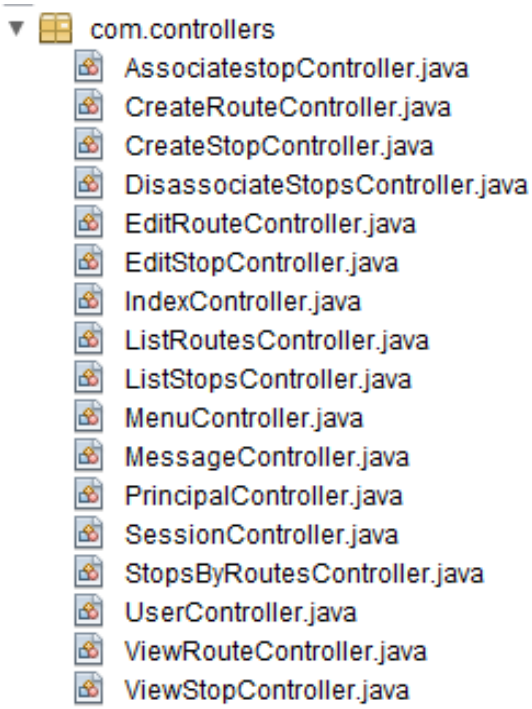
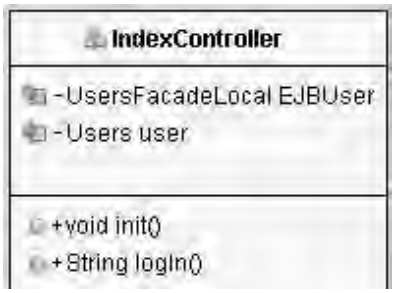
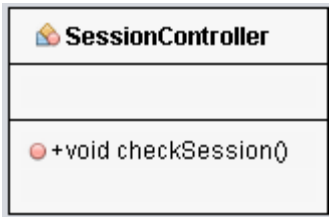


Figura 3. Clase IndexController.java.



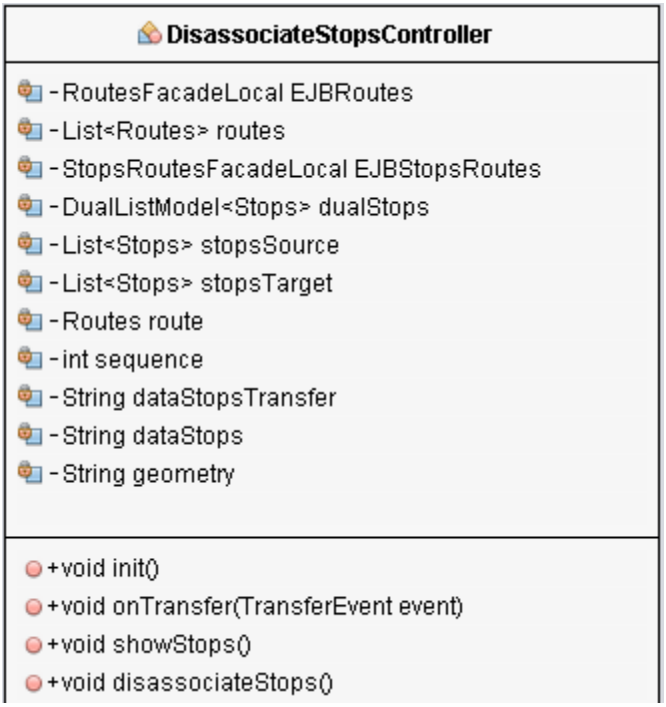
Clase encargada de realizar el proceso de logueo del módulo administrativo de SITApp.

Figura 4. Clase SessionController.java.



Clase encargada de verificar que el usuario haya iniciado sesión en el sistema.

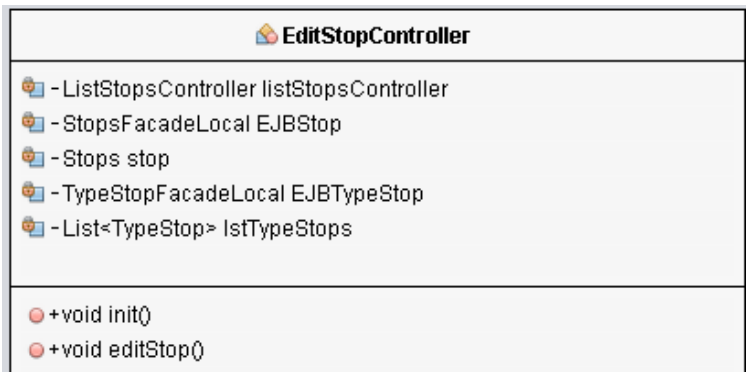
Figura 5. Clase IndexController.java



Clase encargada de desasociar un paradero de una determinada ruta del SETP.

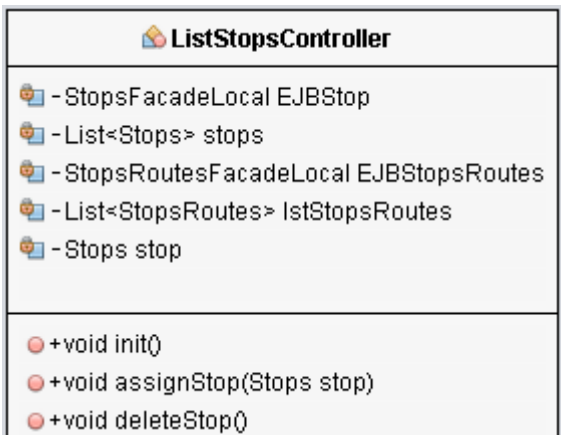
Observación: Los campos dataStopsTransfer, dataStops y geometry son utilizados para mostrar información en el mapa cada vez que se seleccionada una ruta o paradero del SETP, con esto podemos darle una ayuda al usuario para que este sepa que acciones se están realizando cada vez que él hace un cambio en la vista.

Figura 6. Clase EditStopsController.java.



Esta clase permite editar la información de un determinado paradero del SETP.

Figura 7. Clase ListStopsController.java.



Clase encargada de mostrar todos los paraderos del SETP.

Figura 8. Clase ViewStopController.java.

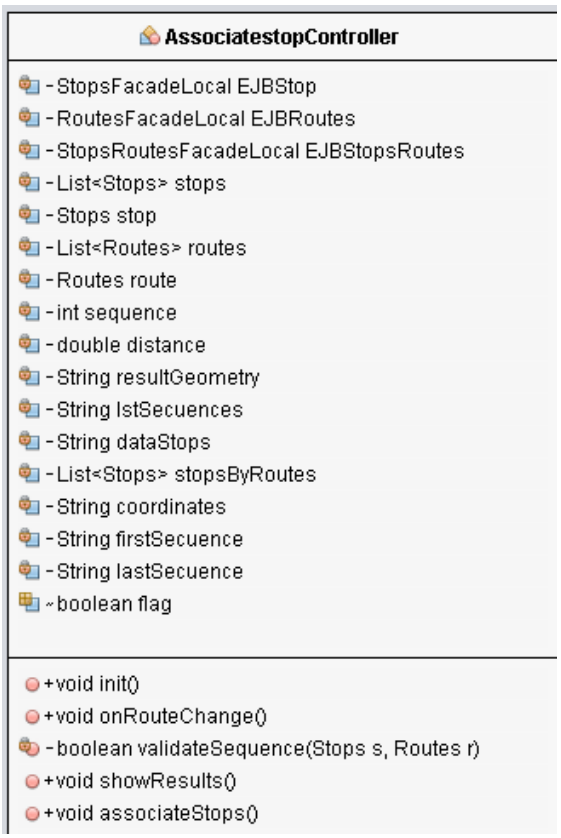
Clase encargada de mostrar la información del paradero seleccionado del listado de paraderos.

Observación: En la vista se accede a cada uno de los atributos que posee el objeto "**stop**".

Figura 9. Clase CreateStopsController.java.

Clase encargada de crear nuevos paraderos en la base de datos del SETP.

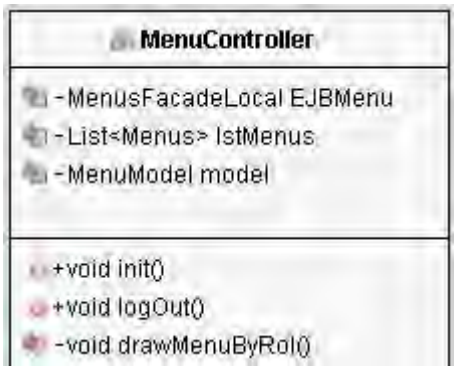
Figura 10. Clase AssociateStopsController.java.



Clase encargada de asociar paraderos a una determinada ruta del SETP.

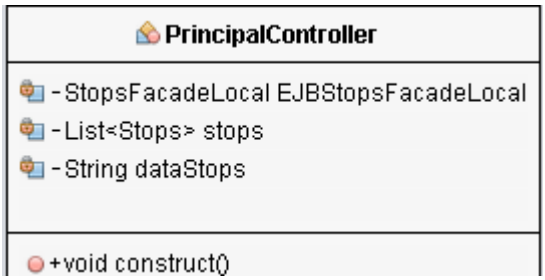
Observación: La información de los campos resultGeometry, dataStops, lstSecuences y coordinates es utilizada en Javascript para actualizar el mapa y mostrar al usuario los resultados de las acciones que esta realizando.

Figura 11. Clase MenuController.java.



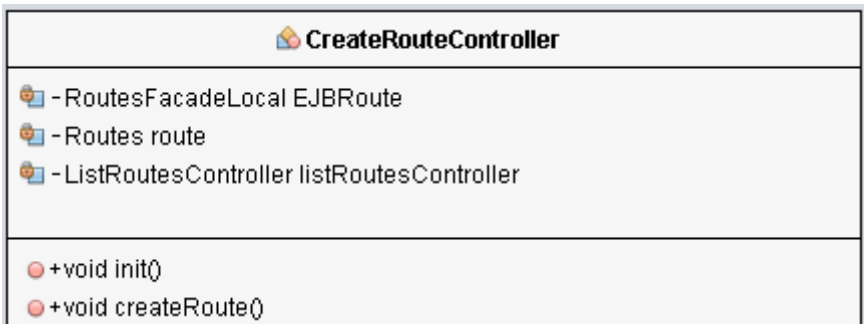
Clase encargada de pintar la barra de opciones en la vista principal y además permite cerrar la sesión del usuario.

Figura 12. Clase PrincipalController.java.



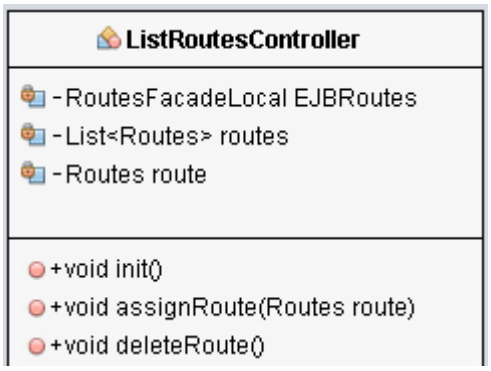
Clase encargada de pintar el mapa y los paraderos en la pagina principal.

Figura 13. Clase CreateRoutesController.java.



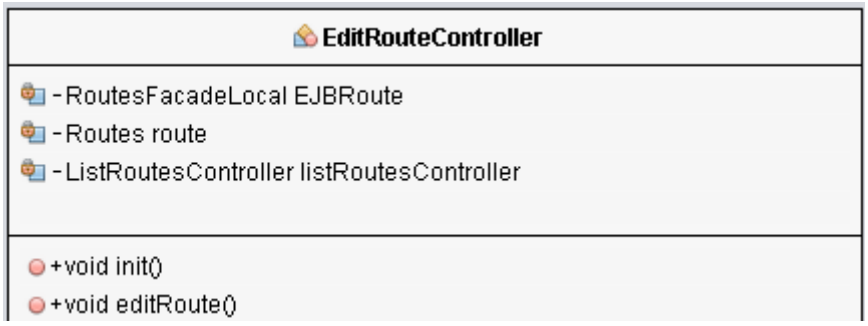
Clase encargada de crear nuevas rutas en el SETP.

Figura 14.Clase ListRoutesController.java.



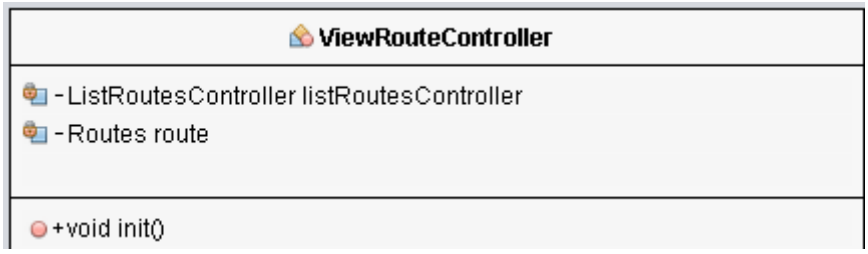
Clase encargada de mostrar todas las rutas del SETP.

Figura 15. Clase EditRoutesController.java.



Esta clase permite editar la información de una determinada ruta del SETP.

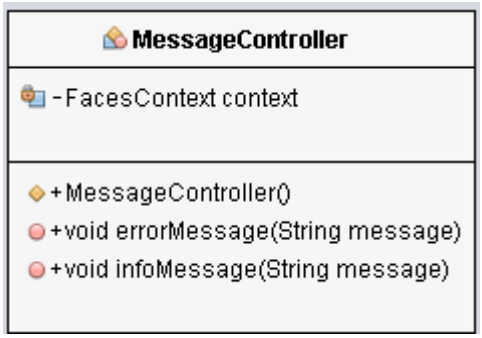
Figura 16. Clase ViewRoutesController.java.



Clase encargada de mostrar la información de la ruta seleccionada del listado de rutas.

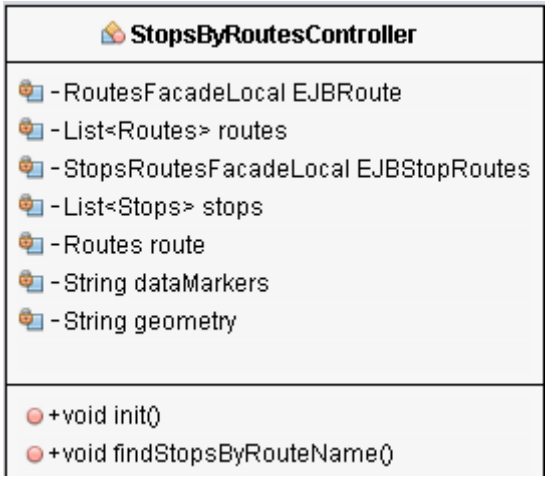
Observación: En la vista se accede a cada uno de los atributos que posee el objeto "**route**".

Figura 17. Clase MessagesController.java.



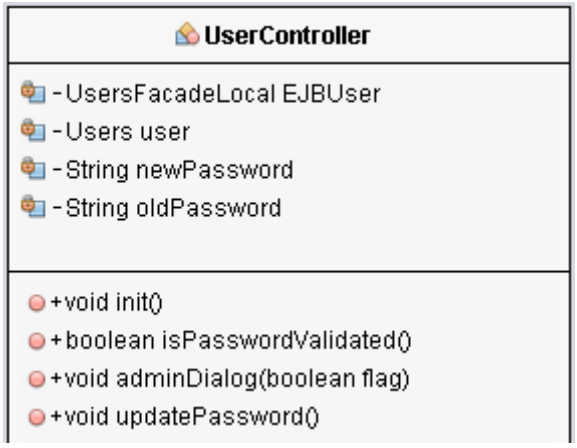
Clase encargada de mostrar los diferentes mensajes.

Figura 18. Clase StopsByRoutesController.java.



Clase que permite gestionar los paraderos por ruta.

Figura 19. Clase UserController.java.



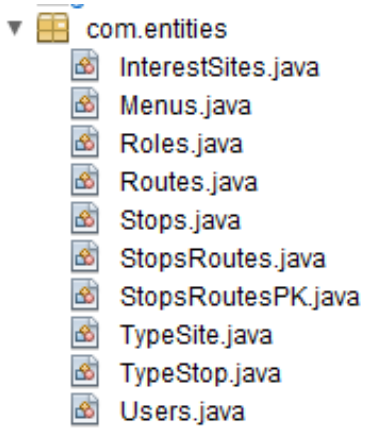
Clase encargada de gestionar la información del usuario administrador del sistema.

- ✓ **Modelo**.

Está compuesto por dos paquetes.

- **Entities**: Contiene el modelo de la base de datos representado en clases escritas en lenguaje de programación Java, las cuales emulan cada una de las tablas de la base de datos. El mapeo de las tablas, con todos sus atributos y restricciones, lo realiza el framework JPA (Java Persistence Api).
- **EJB**: Contiene un conjunto de clases que se encargan de realizar el acceso a los registros contenidos en la base de datos, al momento de intentar realizar cualquier operación sobre la base de datos, insertar, leer, actualizar o eliminar registros, se debe hacer uso de este paquete.

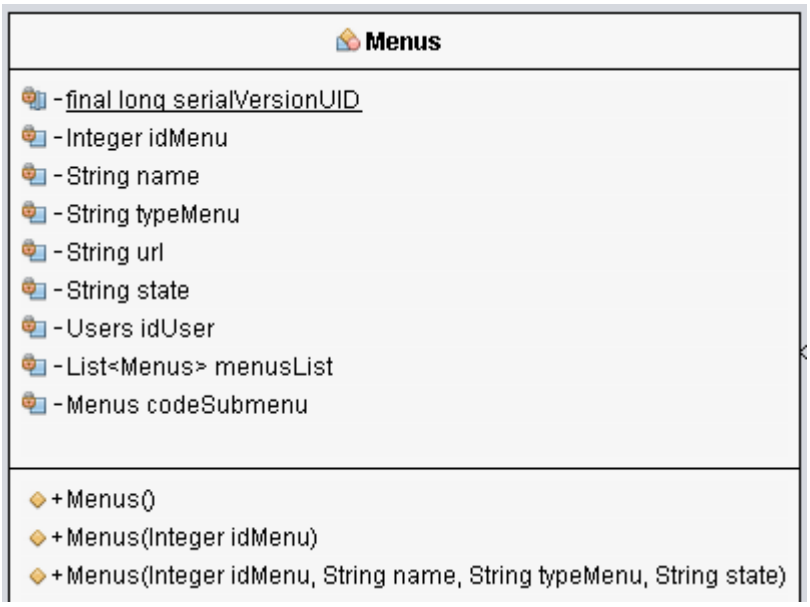
Figura 20. Clases del paquete com.entities.



Las clases que componen el paquete Entities son una representación de las tablas de la base de datos es decir, cada clase representa una tabla de la base de datos.

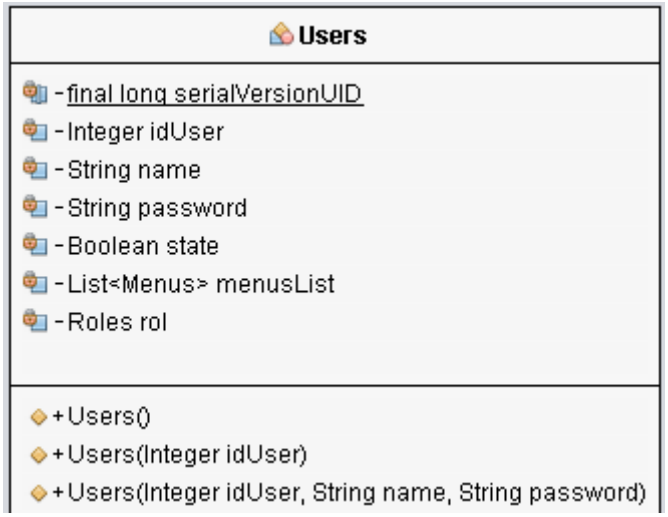
A continuación se realiza una descripción de las clases que componen este paquete.

Figura 21. Clase Menus.java.



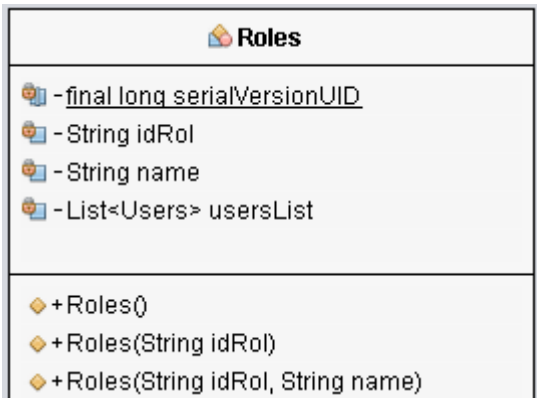
Esta clase permite administrar la información de las diferentes tareas que se pueden realizar a través del módulo administrativo de SITApp.

Figura 22. Clase Users.java.



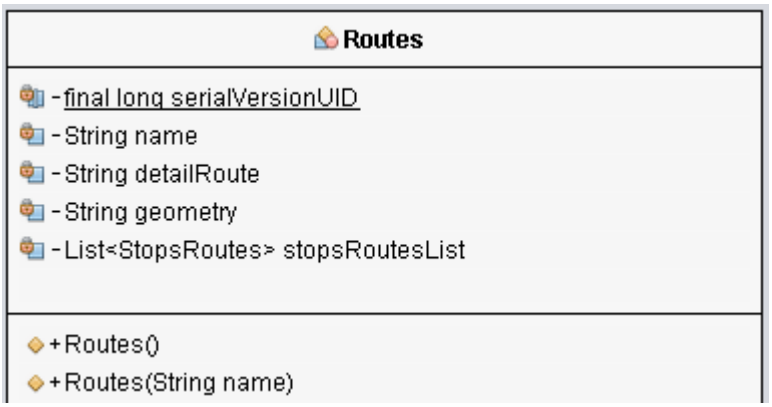
Esta clase permite administrar la información del usuario administrador de SITApp.

Fiugra 23. Clase Roles.java.



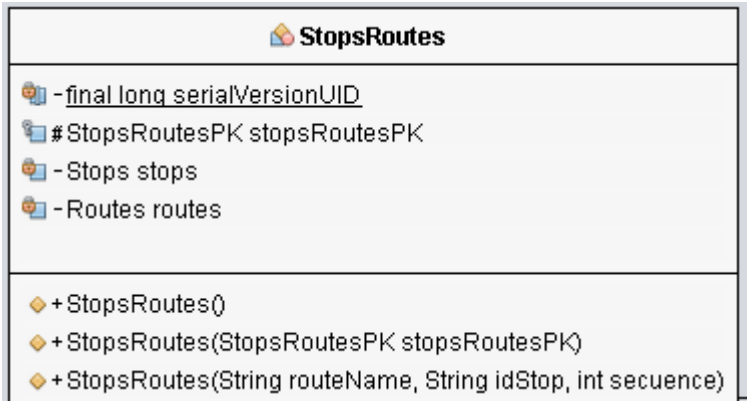
Esta clase permite administrar la información de los diferentes roles que puede tener un usuario en el módulo administrativo de SITApp.

Figura 24. Clase Routes.java.



Esta clase permite administrar la información de las rutas del SETP.

Figura 25. Clase StopsRoutes.java.



Esta clase permite administrar la información de la relación entre paraderos y rutas del SETP.

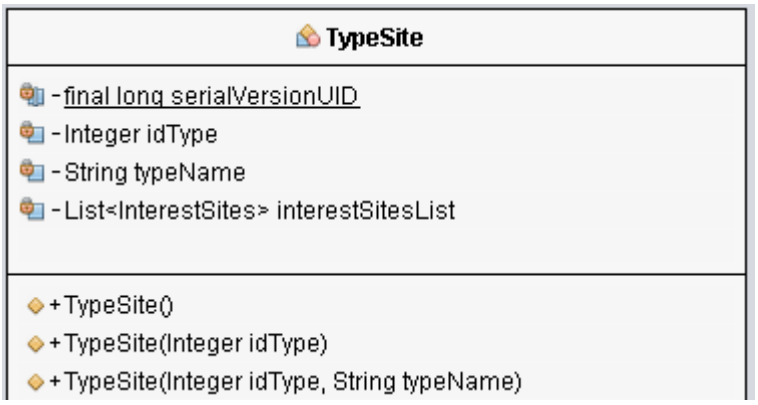
Figura 26. Clase Stops.java.

Esta clase permite adminstrar la información de los paraderos del SETP.

Figura 27. Clase TypeStops.java.

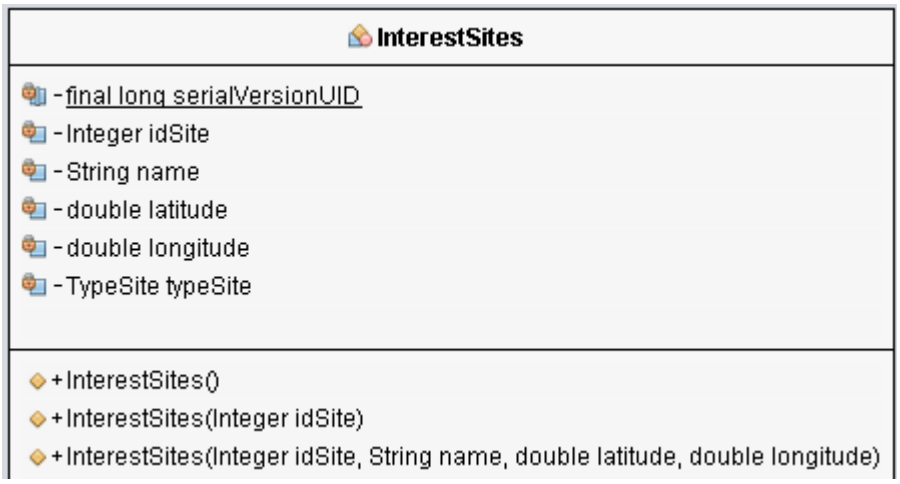
Esta clase permite adminstrar la información de los tipos de paraderos del SETP.

Figura 28. Clase TypeSite.java.



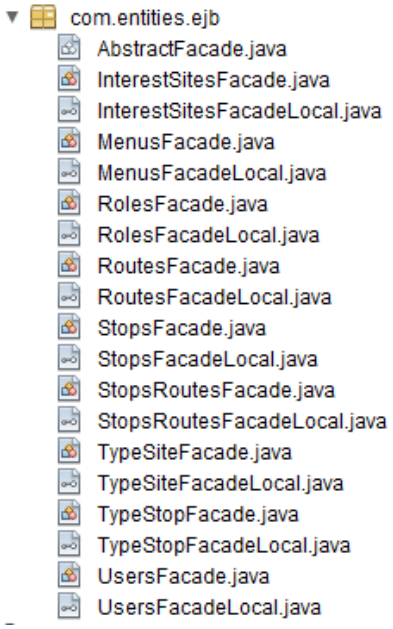
Esta clase permite adminstrar la información de los tipos de sitios de la ciudad de Pasto.

Figura 29. Clase InterestSite.java.



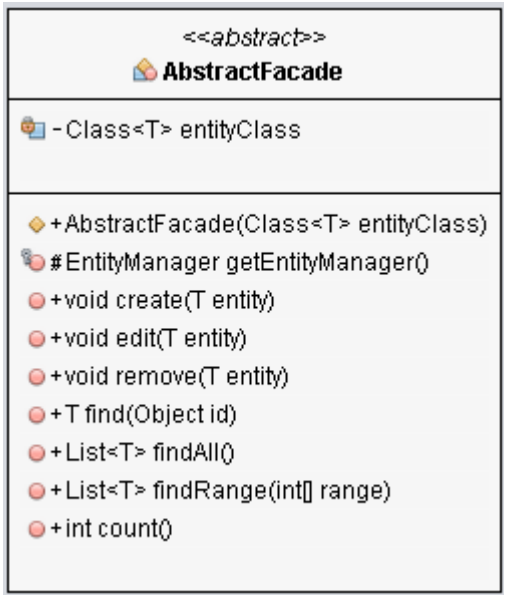
Esta clase permite adminstrar la información de los sitios de interés de la ciudad de Pasto.

Figura 30. Clases del paquete com.entities.ejb



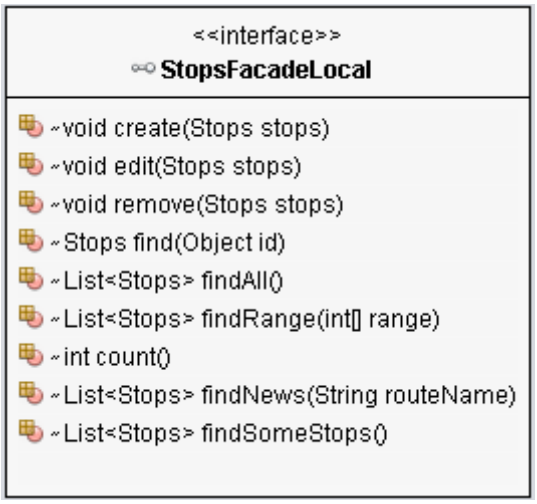
A continuación se realiza una descripción de las clases que componen el paquete EJB.

Figura 31. Clase AbstractFacade.java.



Clase abstracta, de la cual heredan todas las demás clases que conforman este paquete, que tiene todos los métodos CRUD pero estos métodos son abstractos.

Figura 32. Interface StopsFacadeLocal.java.

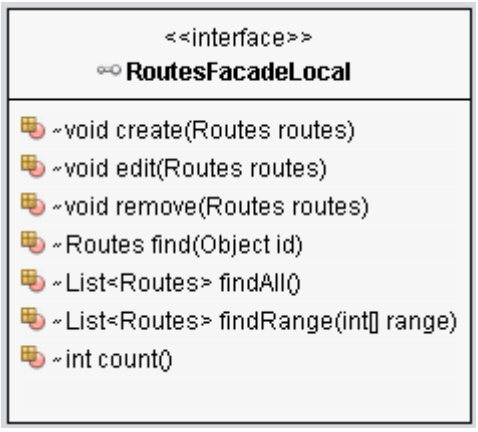


Esta interface posee la declaración de los métodos utilizados para definir el comportamiento de la clase **StopsFacade.java**, esta clase es la responsable de definir el comportamiento de los métodos presentes en la interfaz. Una descripción de cada uno de estos métodos se realiza en la clase StopsFacade.java.

Figura 33. Clase StopsFacade.java.

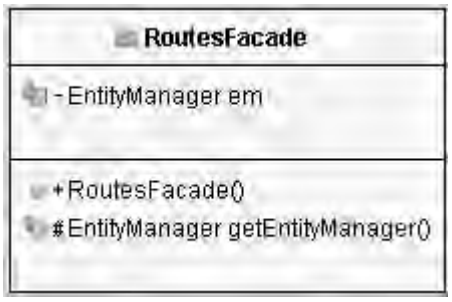
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla stops.

Figura 34. Interface RoutesFacadeLocal.java.



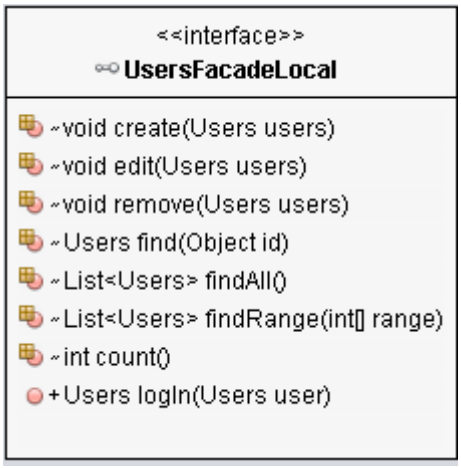
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **RoutesFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase RoutesFacade.java. Una descripción de estos métodos se realiza en la clase RoutesFacade.java

Figura 35. Clase RoutesFacade.java.



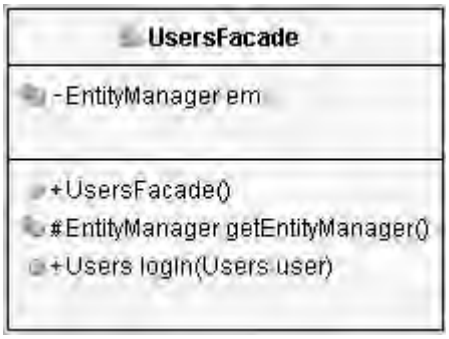
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla routes.

Figura 36. Interface UsersFacadeLocal.java.



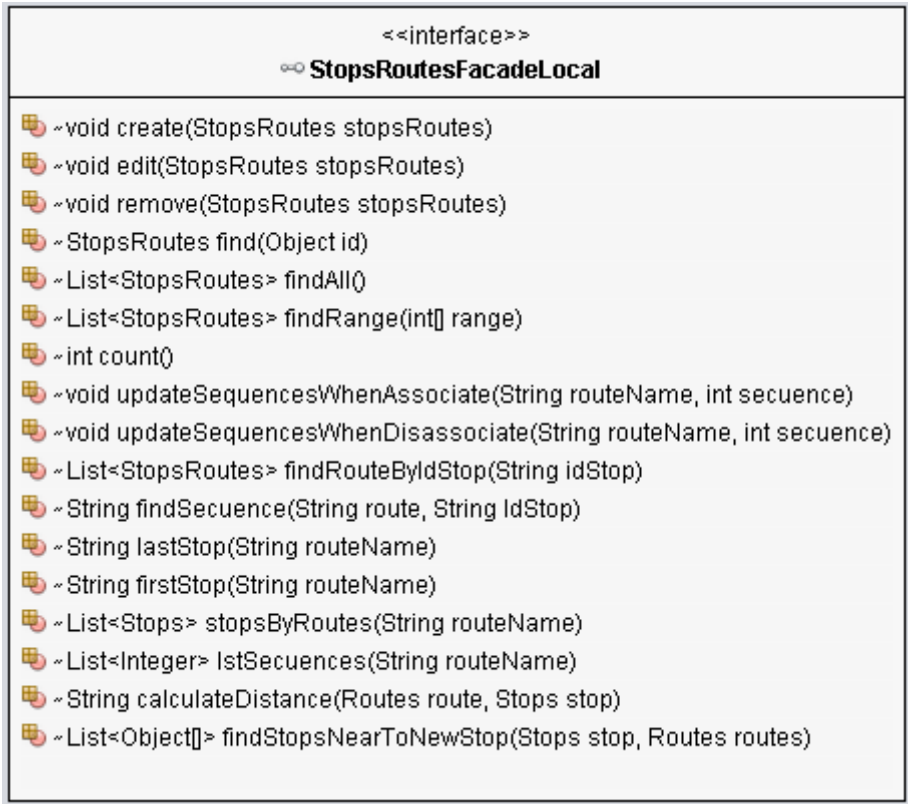
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **UsersFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase UsersFacade.java. Una descripción de estos métodos se realiza en la clase UsersFacade.java

Figura 37. Clase UsersFacade.java.



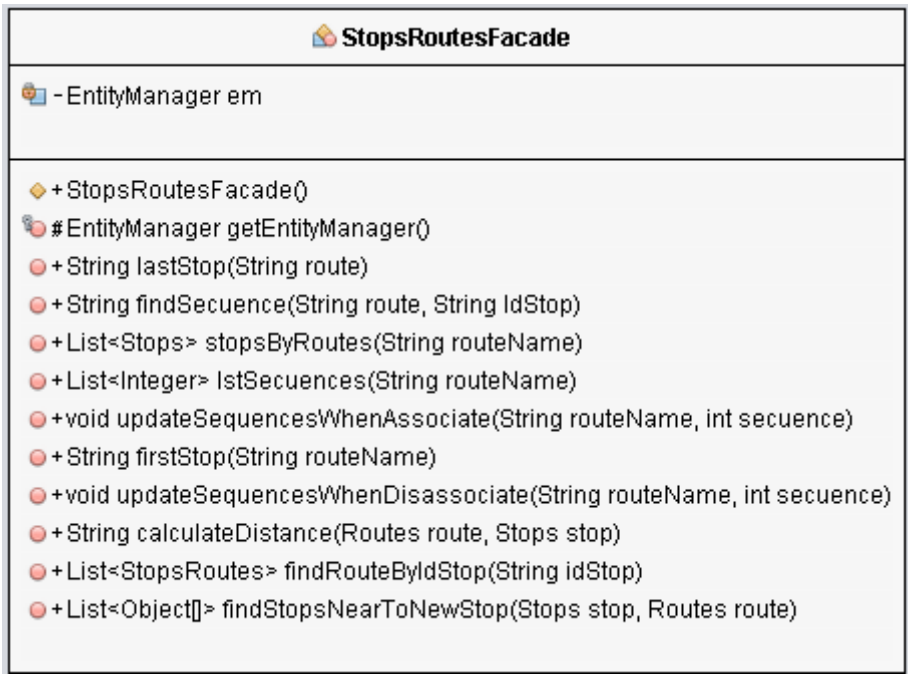
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla users.

Figura 38. Interface StopsRoutesFacadeLocal.java.



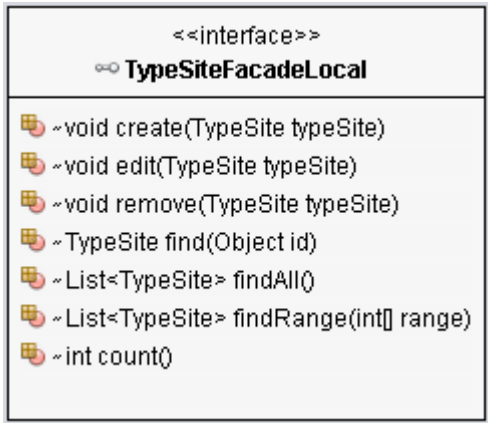
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **StopsRoutesFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase StopsRoutesFacade.java. Una descripción de estos métodos se realiza en la clase StopsRoutesFacade.java

Figura 39. Clase StopsRoutesFacade.java.



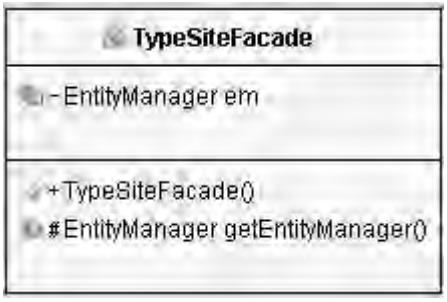
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla StopsRoutes.

Figura 40. Interface TypeSiteFacadeLocal.java.



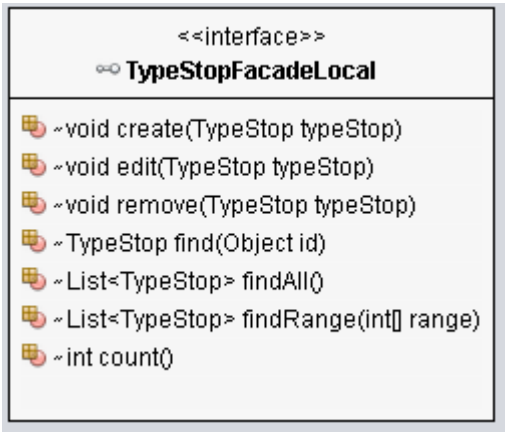
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **TypeSiteFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase TypeSiteFacade.java. Una descripción de estos métodos se realiza en la clase TypeSiteFacade.java

Figura 41. Clase TypeSiteFacade.java.



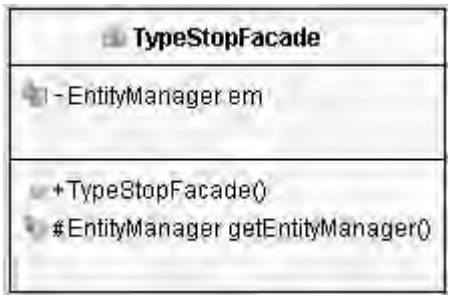
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla type_sites.

Figura 42. Interface TypeStopFacadeLocal.java.



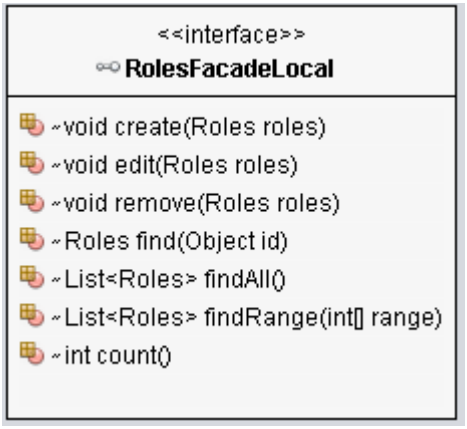
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **TypeStopFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase TypeStopFacade.java. Una descripción de estos métodos se realiza en la clase TypeStopFacade.java

Figura 43. Clase TypeStopFacade.java.



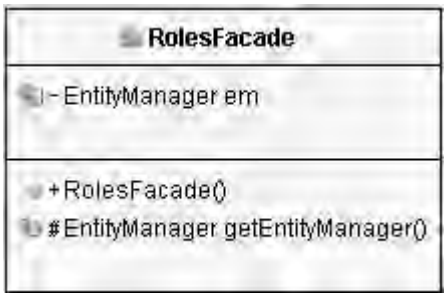
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla type_stop.

Figura 44. Interface RolesFacadeLocal.java.



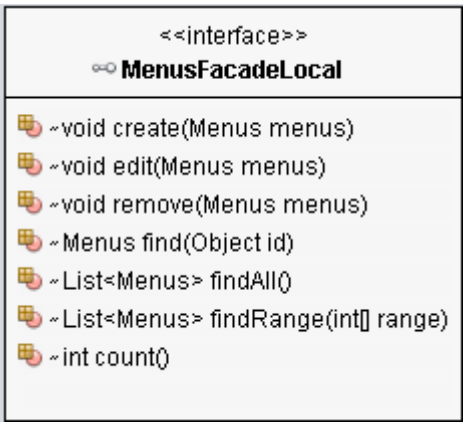
Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **RolesFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase RolesFacade.java. Una descripción de estos métodos se realiza en la clase RolesFacade.java

Figura 45. Clase RolesFacade.java.



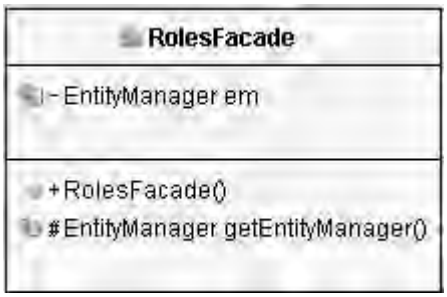
Esta clase posee la implementación de los métodos necesarios para la administración de la tabla roles.

Figura 46. Interface MenusFacadeLocal.java.



Esta interfaz posee la declaración de los métodos utilizados para definir el comportamiento de la clase **MenusFacade.java**. Por lo tanto la clase encargada de realizar la implementación de estos métodos es la clase MenusFacade.java. Una descripción de estos métodos se realiza en la clase MenusFacade.java

Figura 47. Clase MenusFacade.java.



Esta clase posee la implementación de los métodos necesarios para la administración de la tabla menus.

5. Funcionalidades del Módulo Administrativo

A continuación se describen las funcionalidades del módulo administrativo del aplicativo móvil denominado SITApp.

Cabe destacar que por cada archivo con extenxión .xhtml existe una clase java y un archivo JavaScript, esto con el fin de dar solución a las peticiones realizadas por el cliente hacia el servidor y además lograr mostrar el resultado sobre el navegador.

5.1. Eliminar paradero

En esta función el sistema muestra un listado de todos los paraderos, que tiene el Sistema Estratégico de Transporte Público del municipio de Pasto (SETP), el usuario debe seleccionar el paradero que desea eliminar. Cuando el usuario selecciona un elemento para ser eliminado se despliega un dialogo con información del paradero seleccionado y dos botones, Eliminar o Cancelar.

Los elementos incorporados en la vista son un dataTable, un dialog y tres commandButton, todos ellos elementos de primefaces.

Figura 48: Eliminar paradero del SETP del municipio de Pasto.

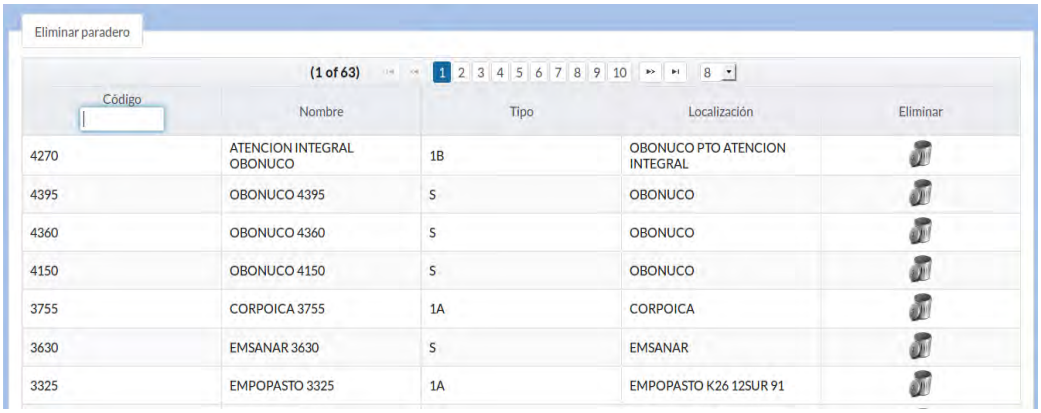
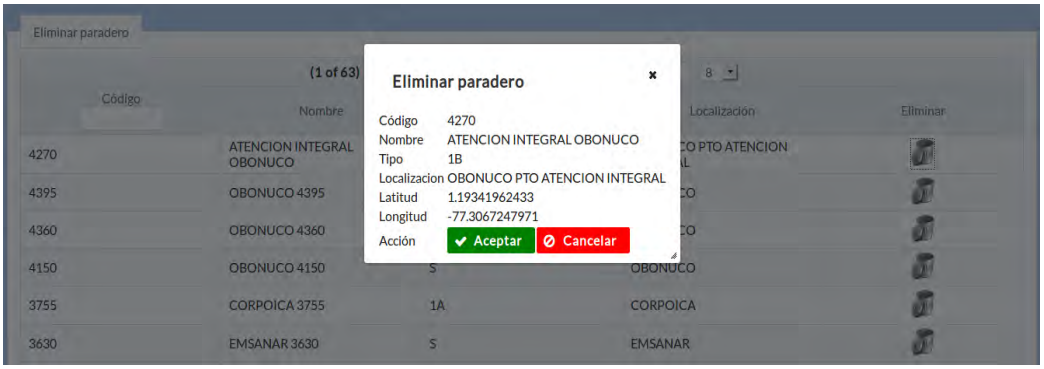


Figura 49: Confirmación de eliminación de paradero.



La selección pasa al controlador **ListStopsController.java**, esta clase se encarga de gestionar la eliminación de paraderos del SETP. Al momento que el usuario presiona el botón **Aceptar**, se ejecuta el método **deleteStop()**, el cual elimina el paradero y todas las relaciones que este tenga con alguna ruta del SETP. En caso de presionar el botón **Cancelar**, el dialogo se cierra. A continuación, se presenta un fragmento del código que realiza la tarea de eliminar un paradero de la base de datos.

```
        try {
            lstStopsRoutes                                                    =
EJBStopsRoutes.findRouteByIdStop(stop.getIdStop());
            if (lstStopsRoutes != null) {
                for (StopsRoutes sr : lstStopsRoutes) {
                    EJBStopsRoutes.updateSequencesWhenDisassociate(
                            sr.getStopsRoutesPK().getRouteName(),
                            sr.getStopsRoutesPK().getSecuence());
                }
            }
            /**
             * This instruction Delete stop of the data base.
             */
            EJBStop.remove(stop);
            this.setStops(EJBStop.findAll());
            messageController.infoMessage("Paradero Eliminado");

        } catch (Exception e) {

    messageController.errorMessage("Paradero      NO      Eliminado      "      +
e.getMessage());
        }
```

Una vez eliminado el paradero se actualiza el array de paraderos y el dataTable que contiene esta información. Al finalizar todo el proceso de eliminación del paradero se muestra el mensaje respectivo al usuario final.

5.2. **Crear paraderos**

Esta función permite crear nuevos paraderos en el SETP del municipio de Pasto. El usuario debe ingresar la información solicitada, todos los datos solicitados son obligatorios, además el usuario debe hacer uso del mapa para ubicar el paradero, en la figura 50 se puede observar el formulario para la creación de paraderos. Para pintar el paradero sobre el mapa, el usuario debe hacer doble clic sobre este elemento.

Los campos solicitados surgieron de la información entregada por AVANTE.

Los elementos presentes en el formulario para creación de paraderos son fielset, un panelGrid, cinco outputLabel, tres inputText, un selectOneMenu, un dialog, tres botones y un mapa, todos estos elementos son etiquetas de primefaces a excepción del mapa, este elemento es proporcionado por MapBox.

Figura 50: Formulario para la creación de paraderos en el SETP.

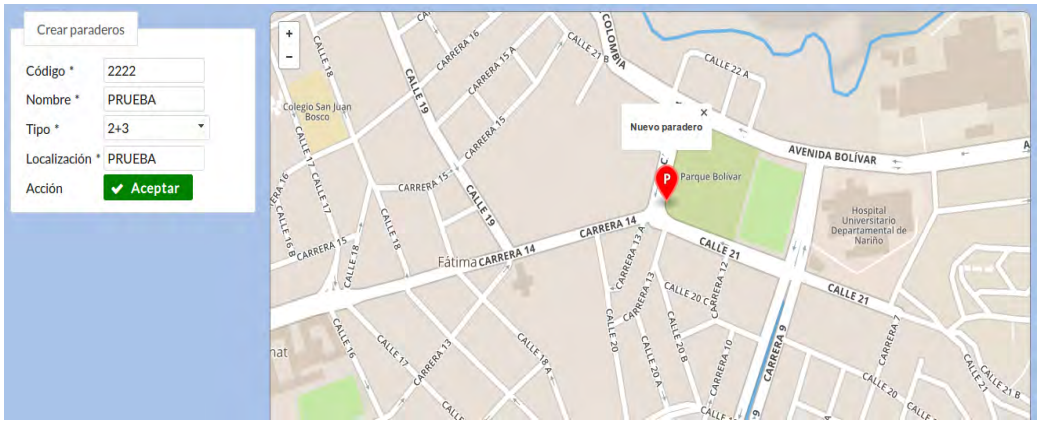
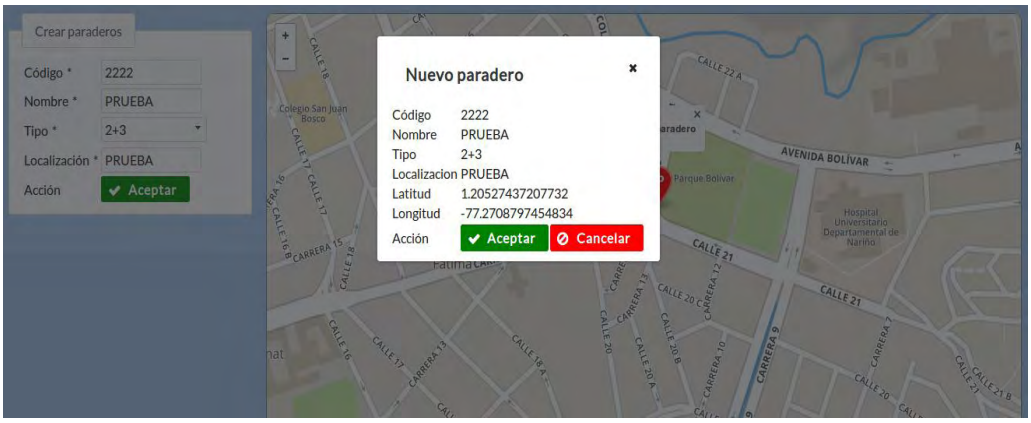


Figura 51: Dialogo con información del nuevo paradero.



Los datos ingresados por el usuario son enviados al controlador **CreateStopsController.java**, esta clase se encarga de crear y almacenar los nuevos paraderos en la base de datos. En el momento en que el usuario presiona el botón **Aceptar**, el cual está dentro del dialogo de confirmación, se ejecuta el método **createStops()**. A continuación se muestra un fragmento de código, el cual crea y almacena los nuevos paraderos.

```
        try {
            stop.setName(stop.getName().toUpperCase());
            stop.setTypeStop(stop.getTypeStop().toUpperCase());
            stop.setLocalization(stop.getLocalization().toUpperCase());
            EJBStop.create(stop);
            this.listStopsController.setStops(EJBStop.findAll());

            messageController.infoMessage("Paradero creado");

        } catch (Exception e) {
```

```
            messageController.errorMessage("Paradero NO creado");
        }
```

Para pintar el paradero sobre el mapa se hace uso de JavaScript. A continuación se muestra la sentencia encargada de dibujar el paradero sobre el mapa.

```
markerStop = L.marker(L.latLng(latitude, longitude), {
    title: 'Nuevo Paradero',
    draggable: true,
    icon: L.mapbox.marker.icon({
        'marker-color': '#FF0000',
        'marker-symbol': 'parking',
        'marker-size': 'large'
    })
})
        .bindPopup("<strong>Nuevo paradero</strong>")
        .addTo(map);
```

Después de crear y almacenar el nuevo paradero se procede a actualizar el array de paraderos y el dataTable que contiene esta información.

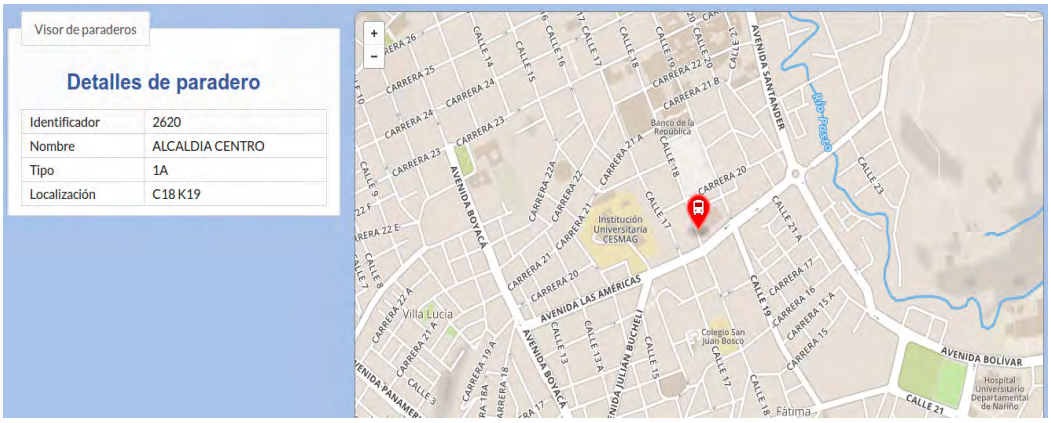
5.3. **Ver y editar**

Esta función permite visualizar o editar un paradero del SETP del municipio de Pasto. El sistema despliega un listado de todos los paraderos del SETP y además en la parte derecha de cada registro se muestran dos botones con las opciones **Ver** y **Editar**.

Para esta vista se hizo uso de un fielSet, un dataTable y dos commandButtons, estos elementos son de primefaces.

- **VER**: Permite visualizar la información de un paradero seleccionado y además se muestra el paradero sobre un mapa, ver figura 52.

Figura 52: Visualización de paradero 2620, ALCALDIA CENTRO.



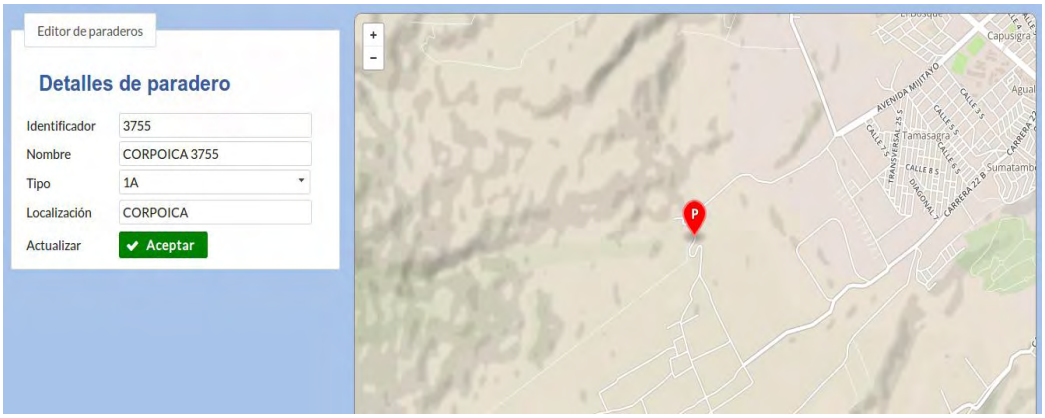
Los elementos presentes en la vista de paraderos son un fielset, un h1, ocho outputLabel estos son elementos de primefaces a excepción del h1, el cual es un elemento de HTML, además se tiene un mapa el cual es proporcionado por MapBox.

El paradero seleccionado pasa al controlador **ViewStopController.java**. Esta clase se encarga de enviar los datos del elemento hacia la vista de paraderos. En cuanto se presiona el botón ver, en el listado de paraderos, se re-direcciona a la vista de paraderos, la cual muestra toda la información del paradero además de su ubicación en el mapa. A continuación se presenta un fragmento de código para mostrar el paradero.

```
var marker = L.marker(L.latLng(latitud, longitud),
 {
    icon: L.mapbox.marker.icon({
       'marker-color': '#FF0000',
       'marker-symbol': 'bus',
       'marker-size': 'large'
    })
 })
 .bindPopup(nombre).addTo(map);
```

- **EDITAR**: Permite actualizar la información de un paradero. Al igual que en la función VER, se muestra la información del paradero pero con la posibilidad de que esta sea editada. En la vista se puede observar los detalles del paradero así como también el lugar donde está ubicado en el mapa, ver figura 53.

Figura 53: Edición de paradero 3755, CORPOICA.



En esta vista se hizo uso de un fieldSet, 5 outputLabel, 5 inputText y un botón, estos son elementos de primefaces. Además se hizo uso de un h1, elemento de HTML, y un mapa proporcionado por MapBox.

Cuando el usuario presiona el botón **Aceptar**, la información del paradero es enviada al controlador **EditStopController.java**, y se ejecuta el método **editStop()**. A continuación se muestra un fragmento de código para la edición del paradero.

```
        try {
            stop.setName(stop.getName().toUpperCase());
            stop.setTypeStop(stop.getTypeStop().toUpperCase());
            stop.setLocalization(stop.getLocalization().toUpperCase());
            EJBStop.edit(stop);
            this.listStopsController.setStops(EJBStop.findAll());

            messageController.infoMessage("Paradero actualizado");

        } catch (Exception e) {
            messageController.errorMessage("Paradero NO actualizado");
          }
        }
```

Además el paradero se muestra sobre un mapa, así que se hizo uso de JavaScript para realizar esta acción. A continuación se muestra un fragmento de código encargado de ubicar el paradero sobre el mapa. Para editar la ubicación del paradero solo es necesario mover el marcador hacia la posición deseada.

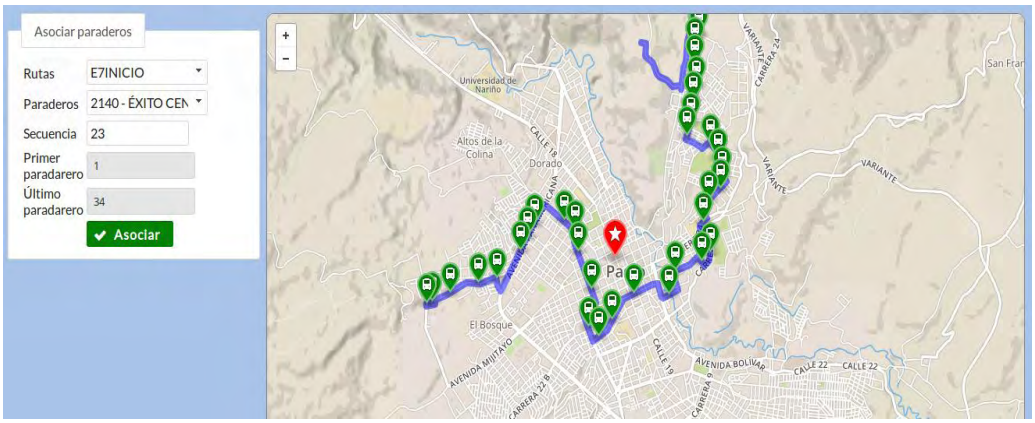
```
var marker = L.marker(L.latLng(latitud, longitud),
        {
            draggable: true,
```

```
            icon: L.mapbox.marker.icon({
                'marker-color': '#FF0000',
                'marker-symbol': 'parking',
                'marker-size': 'large'
            })
        })
        .addTo(map);
    /**
     * when marker is draged its new coordinates are assign at the form's
components
     */
    marker.on('dragend', function () {
        latitud = marker.getLatLng().lat;
        longitud = marker.getLatLng().lng;
        document.getElementById("frmEditS:latitude").value = latitud;
        document.getElementById("frmEditS:longitude").value = longitud;
    });
```

5.4. **Asociar paraderos**

Esta función permite asociar paraderos a una ruta del SETP del municipio de Pasto, ver figura 54.

Figura 54: Asociar paraderos del SETP.



Esta vista contiene un fieldSet, 8 outputLabel, dos selectOneMenu, tres inputText, dos de ellos de solo lectura, estos elementos son de primefaces y un mapa proporcionado por MapBox.

El usuario debe seleccionar la ruta y el paradero que desea asociar. El resultado de la sección se puede ver en un mapa donde el nuevo paradero se

pinta de color rojo y los paraderos que ya forman parte de la ruta se pintan de color verde, además, se traza el recorrido del bus con una línea de color azul.

Los elementos seleccionados así como la secuencia son enviados al controlador, **AssociateStopsController.java**, el cual se encarga de asociar los paraderos a la ruta seleccionada. A continuación se muestra un fragmento de código que hace esta tarea.

```
try {
            EJBStopsRoutes.updateSequencesWhenAssociate(route.getName(),
sequence);
            EJBStopsRoutes.create(new StopsRoutes(route.getName(),
stop.getIdStop(), sequence));
            setFirstSecuence(EJBStopsRoutes.firstStop(route.getName()));
            setLastSecuence(EJBStopsRoutes.lastStop(route.getName()));
            setStops(EJBStop.findNews(route.getName()));

            messageController.infoMessage("PARADERO ASOCIADO");

        } catch (Exception e) {
            messageController.errorMessage("Error al procesar");
        }
```

Una validación que se realiza en esta parte corresponde a la distancia entre la geometría de la ruta y el paradero. La distancia máxima entre estos dos elementos no debe superar los 2 metros, con esto garantizamos que el paradero que se desea asociar a esa ruta si corresponde. A continuación se muestra la consulta sql para realizar esta tarea.

```
String consult = "SELECT st_distance(st_geomfromtext(?1), st_point(?2,
?3))*10000 "
            + "FROM routes "
            + "WHERE routes.name = ?4";
```

Esta consulta se puede realizar gracias al uso de Postgis. Los parámetros que se le pasa a la consulta corresponden a la geometría de la ruta seleccionada, las coordenadas, latitud y longitud, del paradero seleccionado. Esta consulta retorna la distancia entre la geometría y los dos puntos enviados como parámetro.

Al igual que en las funciones anteriores, en esta también se hizo uso de JavaScript para mostrar resultados sobre el navegador. A continuación se muestra un fragmento de código que permite visualizar los datos en el navegador.

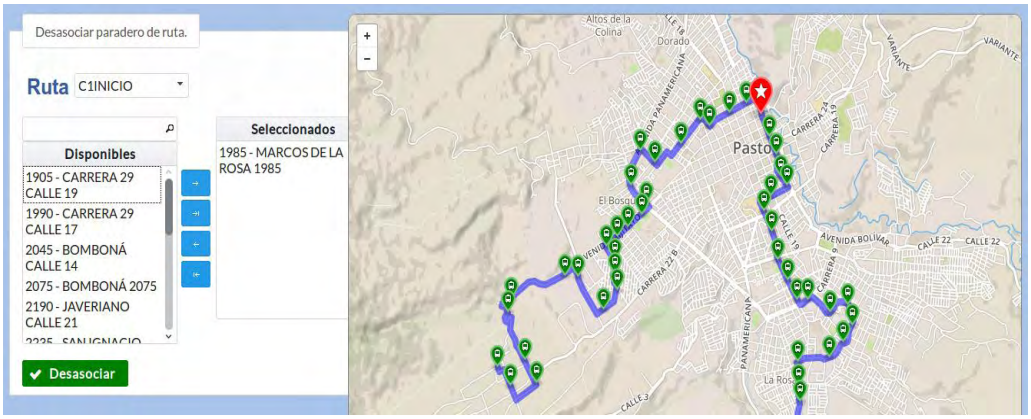
```
    if (geometry !== "") {
        for (var i = 0; i < geometry.length - 1; i = i + 2) {
```

```
            var latlon = L.latLng(geometry[i], geometry[i + 1]);
            points.push(latlon);
        }
        polyline = L.polyline(points, {
            color: 'blue',
            opacity: 0.5,
            weight: 8
        }).addTo(map);
    }
    counter = 0;
    /**
     * Drawing new stop
     */
    latlng = L.latLng(coordinates[0], coordinates[1]);
    marker = L.marker(latlng, {
        icon: L.mapbox.marker.icon({
            'marker-symbol': 'star',
            'marker-color': '#FF0000',
            'marker-size': 'large'
        })
    }).bindPopup("<strong>NUEVO PARADERO<br> secuencia: " + secuence +
"</strong>")
            .addTo(map);
    markers.push(marker);
    /**
     * Drawing stops associates to route
     */
    for (var i = 0; i < dataStops.length - 1; i = i + 3) {
        latlng = L.latLng(dataStops[i], dataStops[i + 1]);
        marker = L.marker(latlng, {
            icon: L.mapbox.marker.icon({
                'marker-symbol': 'bus',
                'marker-color': '#088A08'
            })
        }).bindPopup("<strong>" + dataStops[i + 2] + "<br>Secuencia: " +
lstSecuence[counter] + "</strong>")
                .addTo(map);
        markers.push(marker);
        counter = counter + 1;
    }
```

5.5. **Desasociar paraderos**

Esta función permite desasociar paraderos de una ruta del SETP. En esta opción el usuario debe seleccionar la ruta y el sistema muestra todos los paraderos asociados a esa ruta en específico. El usuario debe seleccionar el paradero a desasociar y presionar el botón **Desasociar**, ver figura 55.

Figura 55: Desasociar paraderos ruta C1INICIO.



Esta vista contiene elementos, de primefaces, tales como un fieldSet, un pickList y un commandButton, además se utilizó un h1, elemento de HTML, y un mapa de MapBox.

La ruta y los paraderos asociados se muestran sobre el mapa y cada vez que el usuario selecciona un paradero este es resaltado de color rojo y con un tamaño superior a los demás.

La información de los paraderos seleccionados es enviada al controlador, **AssociateStopsController.java**, que es la clase encargada de gestionar este tipo de peticiones. Dentro de esta clase se encuentra un método llamado **dissasociateStops()**, el cual da respuesta a la petición hecha por el usuario. A continuación se presenta un fragmento del código que realiza esta tarea.

```
            try {
                sequence =
Integer.parseInt(EJBStopsRoutes.findSecuence(route.getName(),
stop.getIdStop()));
                EJBStopsRoutes.remove(new StopsRoutes(route.getName(),
stop.getIdStop(), sequence));

EJBStopsRoutes.updateSequencesWhenDisassociate(route.getName(),
sequence);
                messageController.infoMessage("Paradero eliminado");
            } catch (Exception e) {
                messageController.errorMessage(e.toString());
            }
```

Para mostrar la ruta, los paraderos y además para resaltar el paradero seleccionado se hizo uso de JavaScript. A continuación se muestra un fragmento de código.

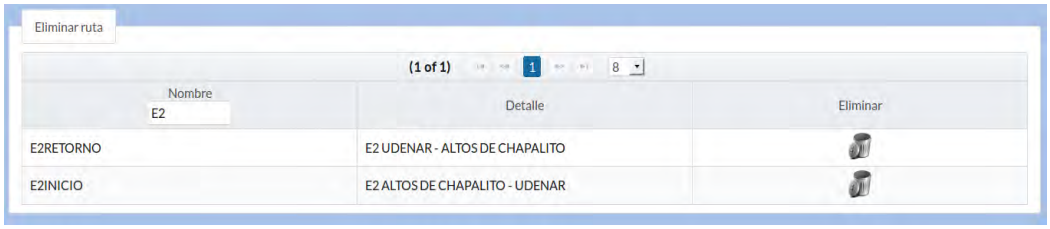
```
    /*
     * draw polyline
     */
    for (var i = 0; i < geometry.length - 1; i = i + 2) {
        var latlon = L.latLng(geometry[i], geometry[i + 1]);
        points.push(latlon);
    }
    polyline = L.polyline(points, {
        color: 'blue',
        opacity: 0.5,
        weight: 8
    }).addTo(map);
    /*
     * draw stops
     */

    if (state === false) {
        for (var i = 0; i < dataStops.length - 1; i = i + 4) {
            var latlng = L.latLng(dataStops[i], dataStops[i + 1]);
            marker = L.marker(latlng, {
                icon: L.mapbox.marker.icon({
                    'marker-symbol': 'bus',
                    'marker-color': '#088A08',
                    'marker-size': 'small'
                })
            }).bindPopup("<strong>" + dataStops[i + 2] + " <br>  " + dataStops[i + 3]
+ "</strong>")
                    .addTo(map);
            markers.push(marker);
        }
```

5.6. **Eliminar rutas**

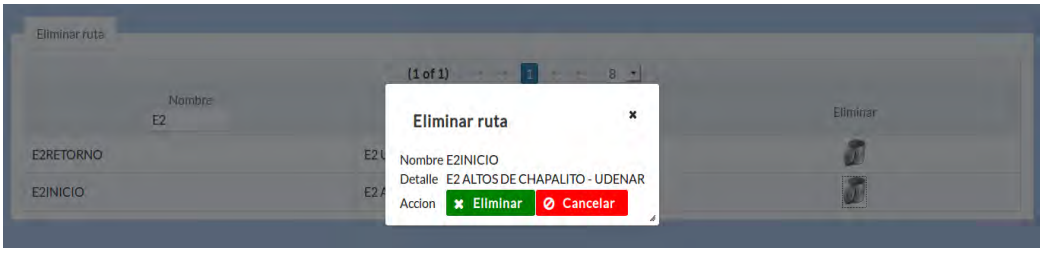
En esta opción el sistema muestra un listado de todas las rutas que tiene el SETP de la ciudad de Pasto. A la derecha de cada registro se muestra un botón, el cual permite eliminar una ruta del SETP, ver figura 56.

Figura 56: Eliminar ruta E2INICIO del SETP.



Cuando el usuario presiona el botón, con forma de bote de basura, se despliega un dialogo con información de la ruta seleccionada, ver figura 57.

Figura 57: Dialogo con información de ruta E2INICIO.



La información de la ruta seleccionada es enviada al controlador, **ListRoutesController.java**, clase encargada de gestionar la eliminación de rutas. Dentro de esta clase hay un método llamado **deleteRoute()** el cual elimina una ruta de la base de datos del SETP del municipio de Pasto. A continuación se presenta un fragmento del código encargado de eliminar una ruta.

```
try {
            EJBRoutes.remove(route);
            this.setRoutes(EJBRoutes.findAll());

            messageController.infoMessage("Ruta Eliminada");

        } catch (Exception e) {

            messageController.errorMessage("Ruta NO Eliminada");
        }
```

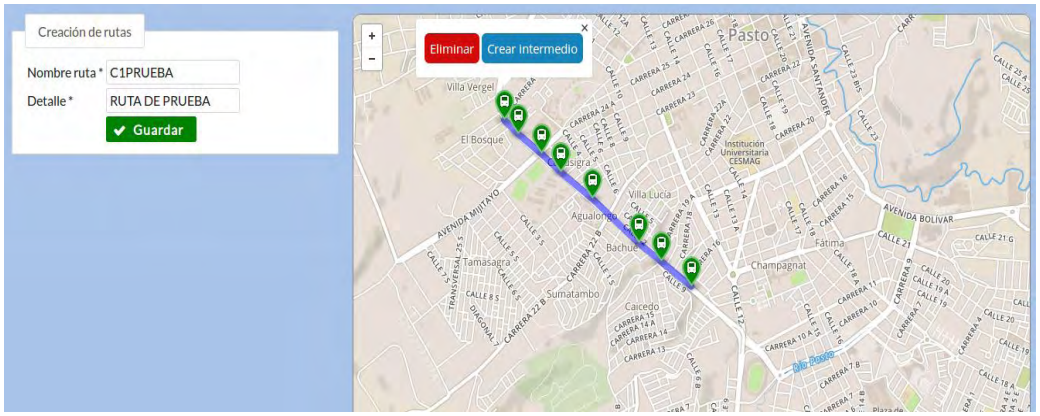
Después de eliminar la ruta se procede a actualizar el array y el dataTable que contiene la información de rutas del SETP.

5.7. **Crear ruta**

Esta función permite la creación de nuevas rutas en el SETP del municipio de Pasto. El usuario debe diligenciar los datos solicitados en el formulario para la creación de rutas, ver figura 58.

Este formulario fue creado utilizando dos outputLabel, dos inputText, un commandButton, etiquetas de primefaces, y un mapa de Mapbox.

Figura 58: Creación de rutas en el SETP.



El usuario puede hacer doble clic sobre el mapa para ir agregando nodos a la geometría de la nueva ruta. Además si se hace clic sobre alguno de los nodos se despliega una ventana emergente con dos botones, los cuales permiten eliminar o crear un marcador entre dos marcadores.

Los datos ingresados en el formulario para la creación de rutas son enviados al controlador **CreateRoutesController.java**, clase que gestiona la creación de rutas. Dentro de esta clase se encuentra un método denominado **createRoutes()**, el cual crea y almacena la nueva ruta en la base de datos. A continuación se muestra un fragmento del código encargado de crear rutas.

```
        try {
            route.setName(route.getName().toUpperCase());
            route.setDetailRoute(route.getDetailRoute().toUpperCase());
            EJBRoute.create(route);
            this.listRoutesController.setRoutes(EJBRoute.findAll());

            messageController.infoMessage("Ruta creada");
        } catch (Exception e) {
            messageController.errorMessage("Ruta NO creada");
        }
    }
```

Para la visualización e interacción del usuario con el mapa se utilizó JavaScript. A continuación se muestra un fragmento de código.

```
map.on('dblclick', function (e) {
    latlong = e.latlng;
    if (geometry.length === 0) {
        geometry.push(latlong);
        polyline = L.polyline(geometry, {
            color: 'blue',
            opacity: 0.5,
```

```
            weight: 8
        }).addTo(map);
        drawMarker(latlong, false);
    }
```

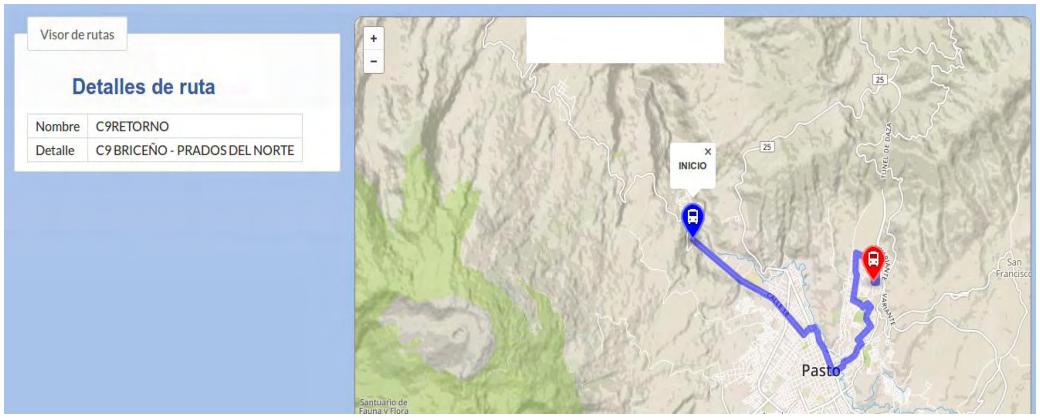
5.8. **Ver y editar**

En esta parte el usuario puede visualizar la información de una ruta determinada o actualizar su información.

Para la construcción de esta vista se hizo uso de un fieldSet, un dataTable y un commandButton.

- **VER**: Permite visualizar la información de la ruta seleccionada por el usuario, se puede observar los detalles así como también el recorrido que realiza, ver figura 59.

Figura 59: Información de ruta C9RETORNO.



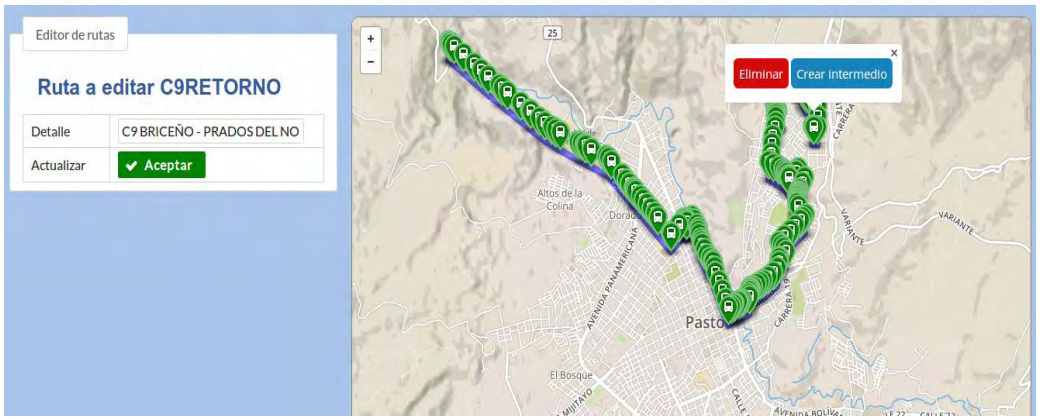
Esta vista contiene elementos de primefaces tales como un fieldSet, cuatro outputLabel, un h1, elemento de HTML, y un mapa de MapBox.
La selección del usuario es enviada al controlador, **ViewRoutesController.java**, clase encargada de enviar la información de la ruta seleccionada hacia el navegador del usuario. Para mostrar la información de la ruta seleccionada se hizo uso de JavaScript. A continuación se muestra un fragmento del código que permite visualizar la información de la ruta seleccionada.

```
/**
 * Drawing polyline
 */
  for (var i = 0; i < coordinate.length - 1; i = i + 2) {
     var latlon = L.latLng(coordinate[i], coordinate[i + 1]);
     points.push(latlon);
  }
```

```
  var polyline = L.polyline(points, {
      color: 'blue',
      opacity: 0.5,
      weight: 8
}).addTo(map);
```

- **EDITAR**: Esta opción permite actualizar la información de una determinada ruta del SETP. En la vista se muestran los detalles de ruta y además el recorrido que esta realiza. El usuario puede mover cada uno de los marcadores hacia la posición deseada y además al hacer clic sobre uno de ellos se muestra una ventana emergente que permite eliminar un marcador o crear uno intermedio, ver figura 60.

Figura 60: Editar ruta C9RETORNO.



La selección es enviada al controlador **EditRoutesController.java**, clase encargada de gestionar estas peticiones. Dentro de esta clase existe un método llamada **editRoutes()**, el cual se encarga de la actualización de rutas. A continuación se muestra un fragmento de código que se encarga de permitir la edición de ruta.

```
        try {

            route.setName(route.getName().toUpperCase());
            route.setDetailRoute(route.getDetailRoute().toUpperCase());
            EJBRoute.edit(route);
            this.listRoutesController.setRoutes(EJBRoute.findAll());

            messageController.infoMessage("Ruta actualizada");

        } catch (Exception e) {
```

```
            messageController.errorMessage("Ruta NO actualizada");
        }
```

Para mostrar el recorrido de ruta y además permitir el movimiento de los marcadores se hizo uso de JavaScript. A continuación se muestra un fragmento de código encargado de realizar esta tarea.

```
    /**
     * new marker has several events
     */
    auxMarker.on('mousedown', function (e) {
        latlong = e.latlng;
    });
    auxMarker.on('mouseup', function (e) {
        latLongFinal = e.latlng;
    });
    auxMarker.on('click', function (e) {
        latlong = e.latlng;
        $('#del').click(function () {
            deleteMarker(latlong);
        });
        $('#add').click(function () {
            createMarkerIntermediate(latlong);
        });
    });
    auxMarker.on('dragend', editRoute);

    /**
     * calculate distance between two markers
     */
    var distanceToLastMarker = parseInt(lstMarkers[lstMarkers.length -
1].getLatLng().distanceTo(auxMarker.getLatLng()).toFixed(0));
    var distanceToFirstMarker =
parseInt(lstMarkers[0].getLatLng().distanceTo(e.latlng).toFixed(0));

    if (distanceToLastMarker < distanceToFirstMarker) {
        lstMarkers.push(auxMarker);
        updatePolyine();
    } else {
        lstMarkers.unshift(auxMarker);
        updatePolyine();
    }
});

polyline = L.polyline(points, {
    color: 'blue',
    opacity: 0.5,
    weight: 8
```

```
}).addTo(map);

drawMarkers(points);
addEventMarkers();
```

El usuario puede hacer doble clic sobre el mapa para agregar nuevos puntos a la geometría.

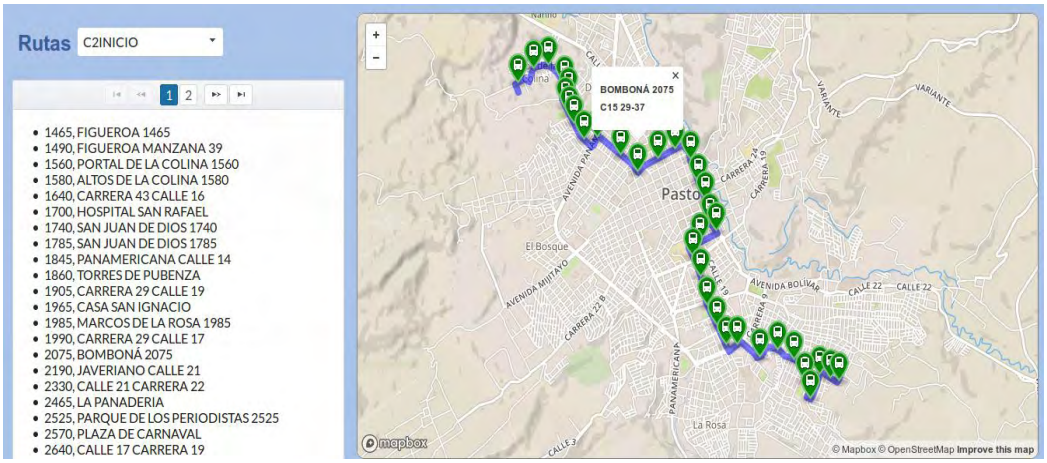
5.9. **Paraderos asociados**

Esta opción permite ver los paraderos asociados a una determinada ruta. El usuario puede ver un listado de todos los paraderos además se pinta sobre el mapa el recorrido de la ruta y los diferentes paraderos asociados a la ruta seleccionada, ver figura 61.

Esta vista tiene un h1, elemento de HTML, un selectOneMenu, un dataList, elementos de primefaces, y un mapa de MapBox.

El usuario puede hacer clic sobre alguno de los marcadores y se desplegará una ventana emergente con información acerca de la ubicación del paradero.

Figura 61: Paraderos asociados ruta C2INICIO.



La información de la ruta seleccionada es enviada al controlador **StopsByRoutesController.java**, esta clase es la encargada de consultar en la base de datos los paraderos asociados a una determinada ruta del SETP. Dentro de esta clase se encuentra un método llamado **findStopsByRouteName()** el cual consulta en la base de datos los paraderos

asociados a la ruta seleccionada. A continuación se muestra un fragmento de código utilizado para lograr mostrar los paraderos asociados a una ruta.

```
        try {
            stops = EJBStopRoutes.stopsByRoutes(route.getName());
            geometry = route.getGeometry();
            if (stops.size() > 0) {
                for (Stops stop : stops) {
                    dataMarkers += stop.getLatitude() + ";";
                    dataMarkers += stop.getLongitude() + ";";
                    dataMarkers += stop.getName() + ";";
                    dataMarkers += stop.getLocalization() + ";";
                }
            } else {
                messageController.errorMessage("RUTA SIN PARADEROS");
            }
        } catch (Exception e) {
            messageController.errorMessage(e.getMessage());
        }
```

Del mismo modo que en funciones anteriores, en este caso también se utilizó JavaScript para mostrar los resultados en el navegador. A continuación se muestra un fragmento de código utilizado para mostrar los resultados de la selección en el navegador.

```
    /**
     * draw markers on the map
     */
    for (var i = 0; i < dataMarkers.length - 1; i = i + 4) {
        var latlng = L.latLng(dataMarkers[i], dataMarkers[i + 1]);
        marker = L.marker(latlng, {
            icon: L.mapbox.marker.icon({
                'marker-symbol': 'bus',
                'marker-color': '#088A08'
            })
        }).bindPopup("<strong>" + dataMarkers[i + 2] + " <br>  " + dataMarkers[i +
3] + "</strong>")
                .addTo(map);
        markers.push(marker);
    }

    /*
     * draw polyline
     */
    for (var i = 0; i < geometry.length - 1; i = i + 2) {
        var latlon = L.latLng(geometry[i], geometry[i + 1]);
```

```
    points.push(latlon);
}

polyline = L.polyline(points, {
    color: 'blue',
    opacity: 0.5,
    weight: 8
});
```

ANEXO F. Construcción de la ontología

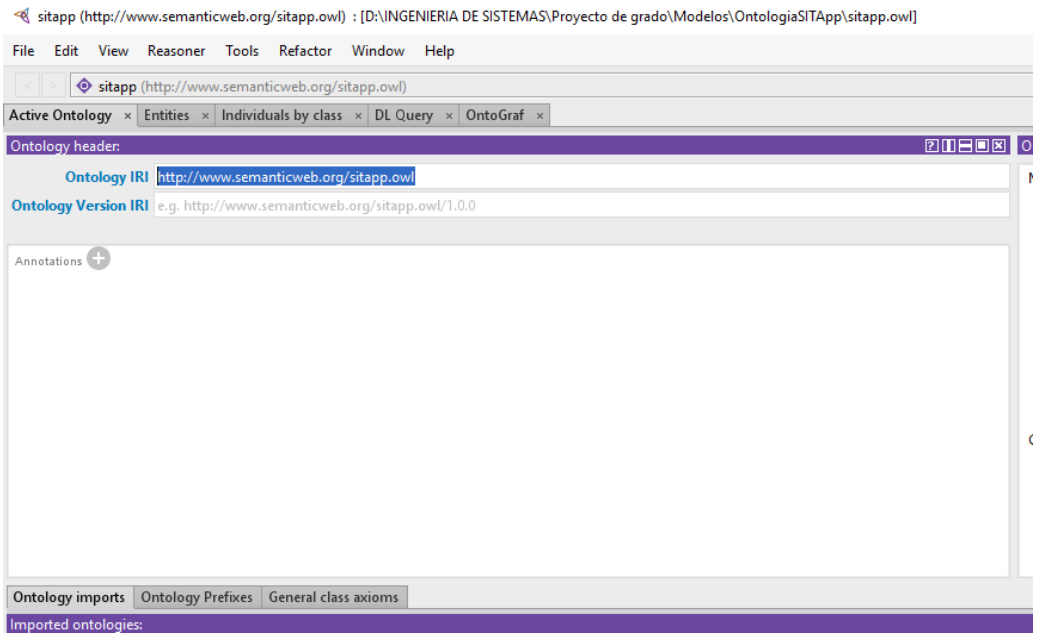
Para la creación de la ontología se hizo uso de la herramienta libre Protegé que se puede descargar desde el siguiente enlace https://protege.stanford.edu/products.php#desktop-protege.

Tomando como base los pasos realizados en el apartado 5.3. Fase 3. Diseño y construcción de la ontología del proyecto SITApp, a continuación se describen cada uno de los pasos para la creación de la ontología en la herramienta.

1. Creacion del IRI de la ontología

Internationalized Resource Identifier (IRI) es el recurso identificador internacionalizado que es básicamente una URL o dirección web que no necesariamente existe, pero que puede ser publicada y al asociarle la ontología a esta dirección web le dará semántica al contenido de este sitio. La creación del IRI se realiza en la primera pestaña de la herramienta, para la ontología de SITApp se utilizo la dirección http://www.semanticweb.org/sitapp.owl como se muestra en la Figura 1:

Figura 1. Creación del IRI de la ontología.



2. Creación de las clases

Para la creación de las clases procedemos a utilizar la pestaña Entities en la cual encontraremos inicialmente la definición de la clase madre owl:Thing de la cual heredaran nuestras clases Palabras_clave y Sitios_interes. Para

agregarlas basta con hacer click en el botón que se encuentra encima de owl:Thing que nos permite crear las subclases como lo vemos en la Figura 2, de esta manera creamos las dos clases mencionadas y tendremos como resultado lo que se muestra en la Figura 3:

Figura 2. Creación de la clase Sitios_interes.

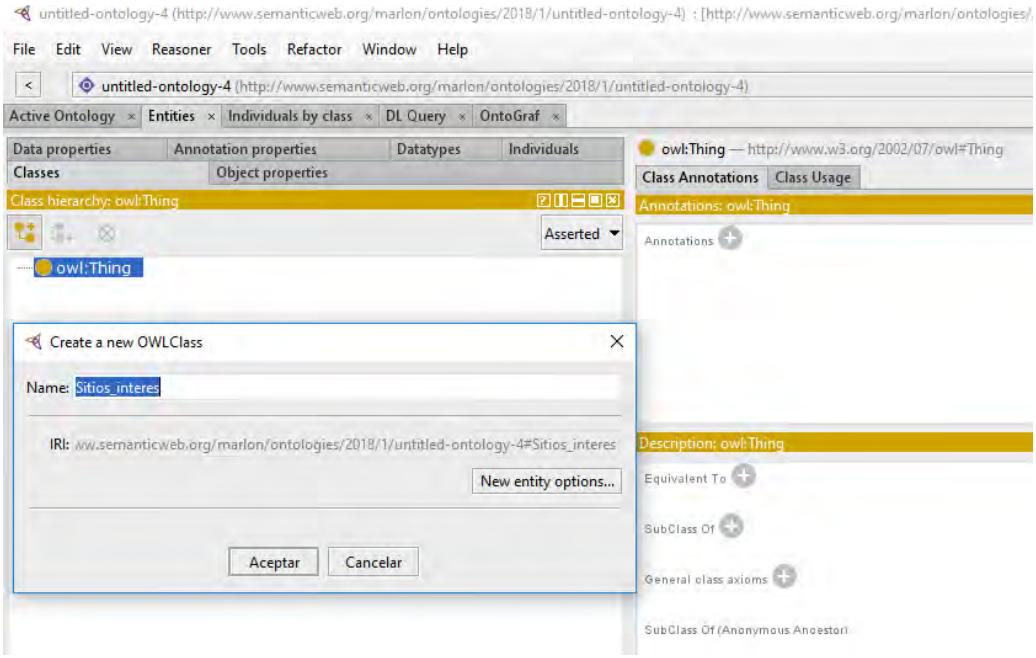
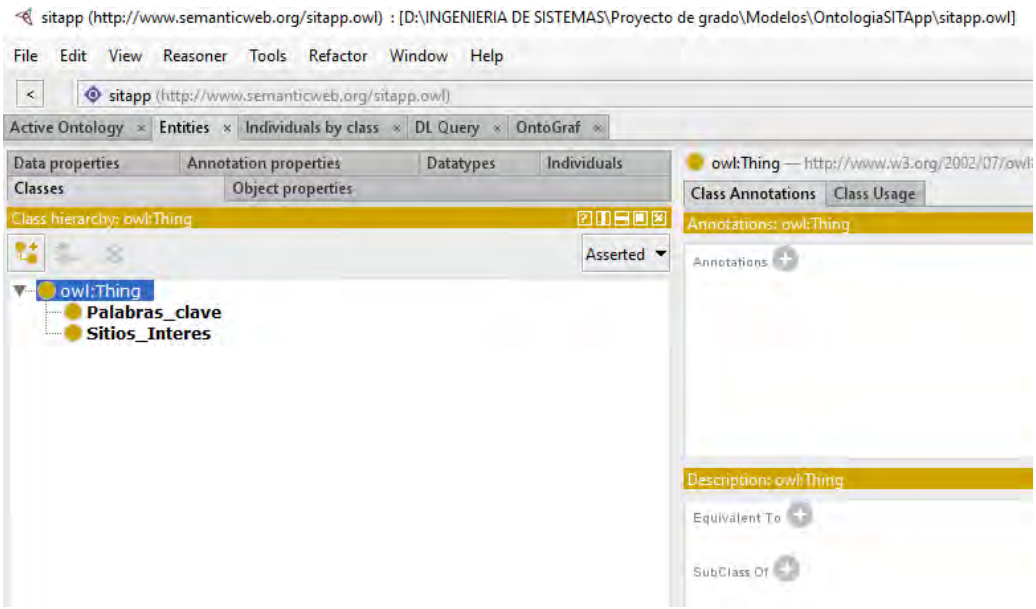


Figura 3. Resultado de la creación de las clases Palabras_clave y Sitios_interes.



3. Creación de las relaciones entre clases

Ahora accedemos a la pestaña secundaria Object Properties en la cual definiremos las relaciones entre las dos clases Sitios_interes y Palabras_clave. Las relaciones que se crearán son “describe”, la cual describe la propiedad Palabras_clave “describe” Sitios_interes y “es_descrito_por”, la cual describe la propiedad Sitio_interes “es_descrito_por” Palabras_clave. Para crear estas relaciones hacemos click sobre la propiedad principal owl:topObjectProperty y encima de esta propiedad existe un botón que nos permite crear las propiedades que hemos descrito, al hacer click en el nos mostrará un recuadro como se muestra en la Figura 4, al finalizar la creación de las propiedades tendremos como resultado lo que se muestra en la Figura 5:

Figura 4. Creación de la propiedad describe.

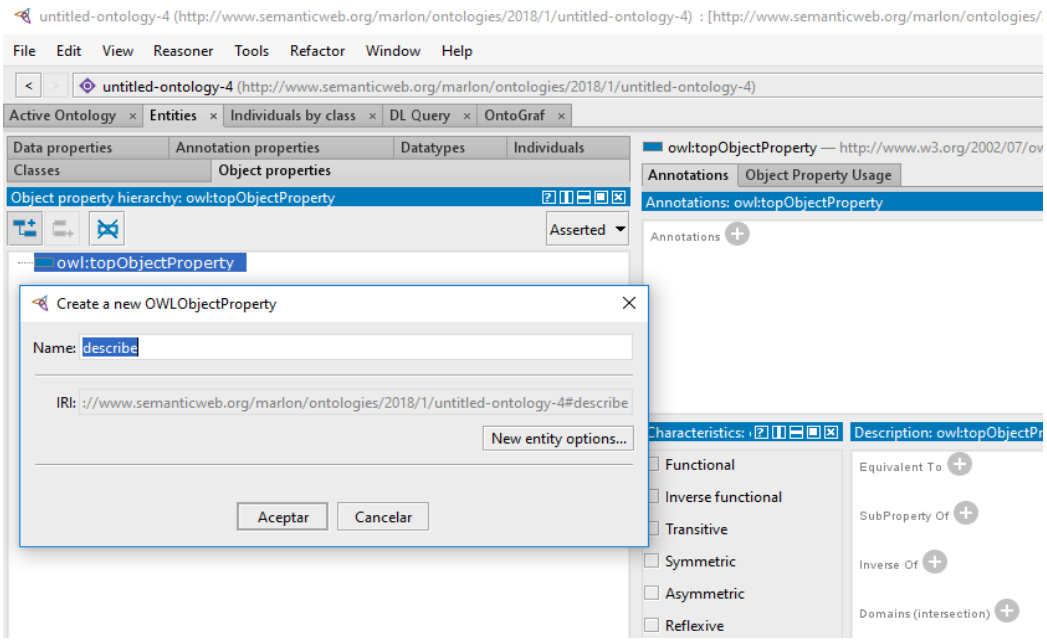
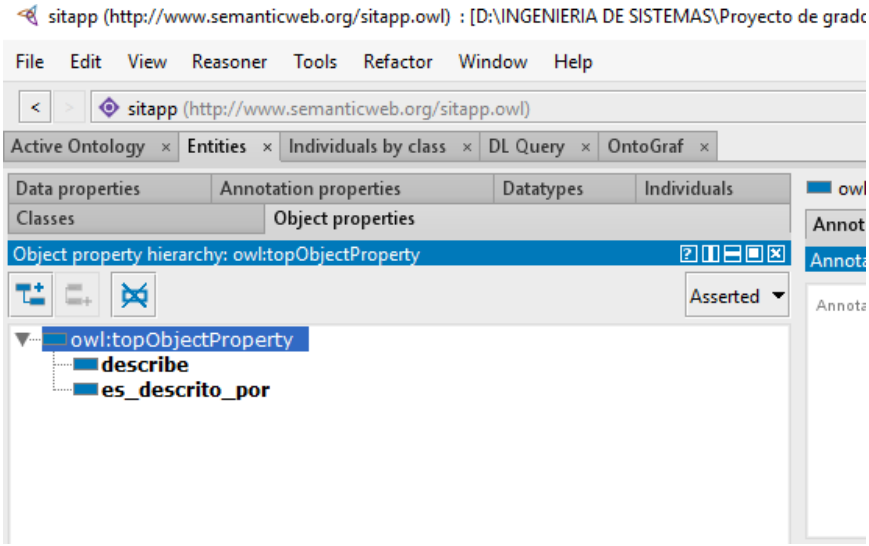


Figura 5. Resultado de la creación de las propiedades describe y es_descrito_por.



Ahora debemos describir el rango y el dominio de cada una de las propiedades, hay que destacar que las propiedades son inversas entre sí. El dominio de la propiedad describe es “describe” es Palabra_clave y el rago es Sitios_interes y se especifica la propiedad inversa la cual es “es_descrito_por”, al ser la propiedad “es_descrito_por” inversa a la anterior el dominio, rango y su propiedad inversa serán Sitios_interes, Palabras_clave y describe respectivamente como se muestra en las Figuras 6 y 7:

Figura 6. Descripción propiedad “describe”

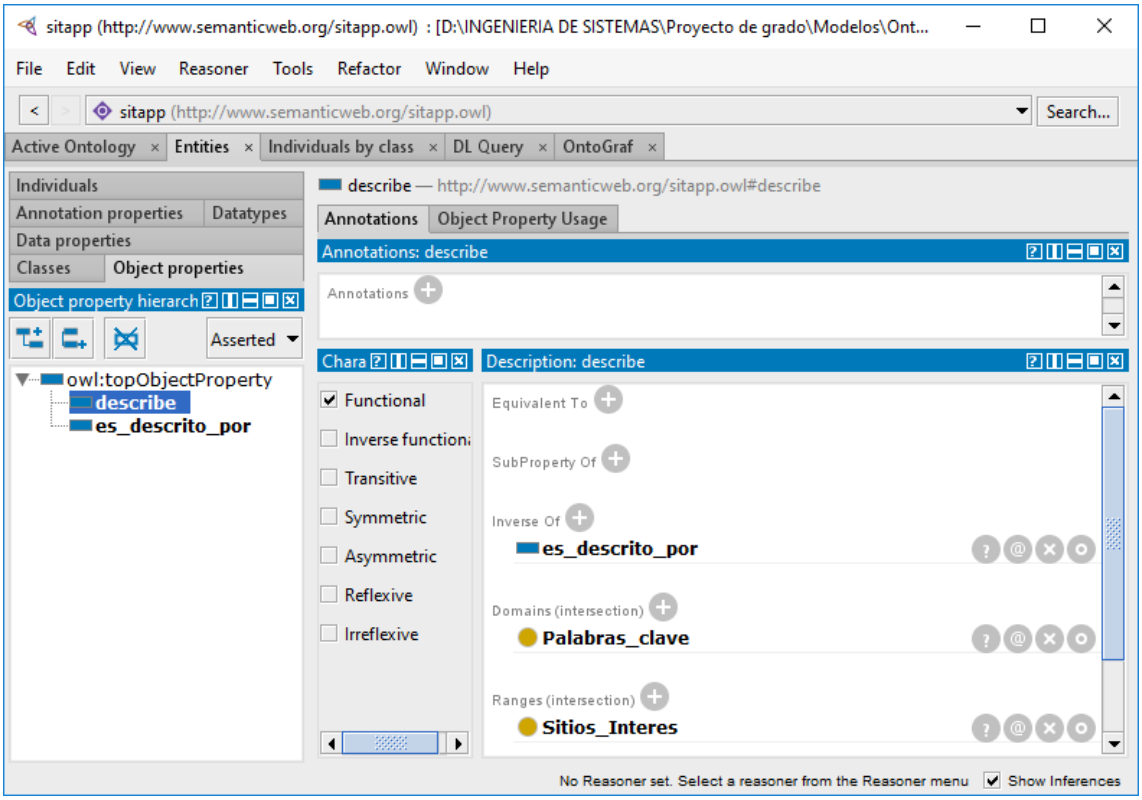
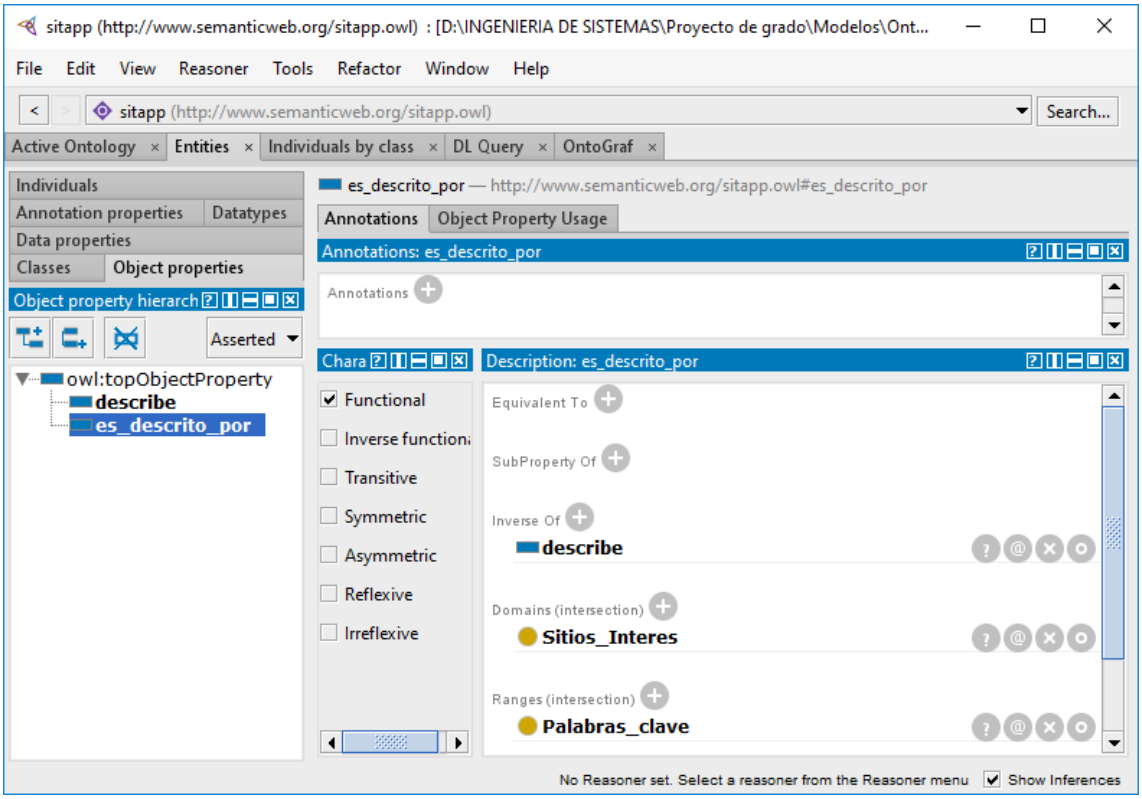


Figura 7. Descripción de la propiedad “es_descrito_por”.



4. Creación de las propiedades de los datos.

La creación de los datos nos permite determinar para cada uno de los individuos que pertenecen a determinada clase, su dominio y su rango, siendo el dominio la clase a la que el individuo pertenece y el rango el tipo de dato ya sea entero, carácter u otro tipo de dato. El proceso es similar a los pasos anteriores, se accede a la pestaña secundaria Data properties, se hace click en la propiedad principal owl:topDataProperty y posteriormente se clickea el botón que se encuentra encima de esta propiedad para crear las propiedades id, palabra y sinónimos. Las propiedades id y sinónimos tienen como dominio la clase Sitios_interes y la propiedad palabra tiene como dominio la clase Palabras_clave, las 3 propiedades tienen como rango xsd:string, el resultado obtenido de la creación de cada una de las propiedades se describe a continuación en las Figuras 8, 9 y 10:

Figura 8. Descripción de la propiedad “id”.

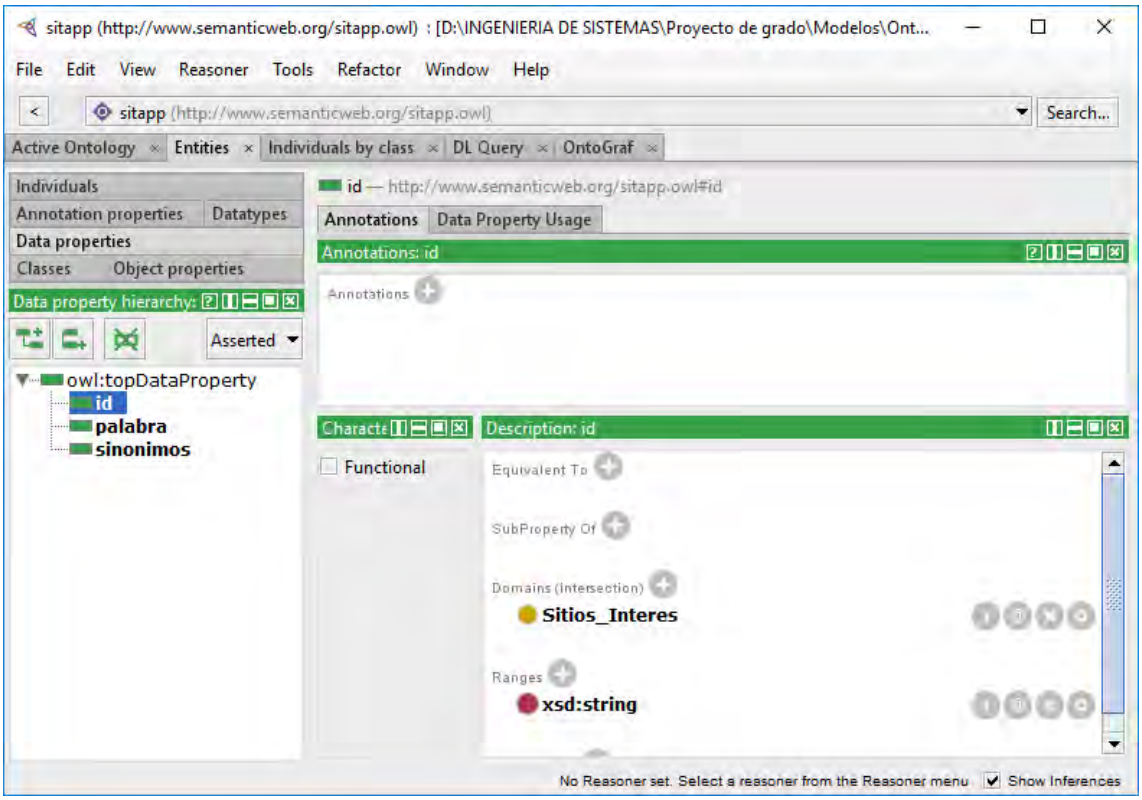


Figura 9. Descripción de la propiedad “palabra”.

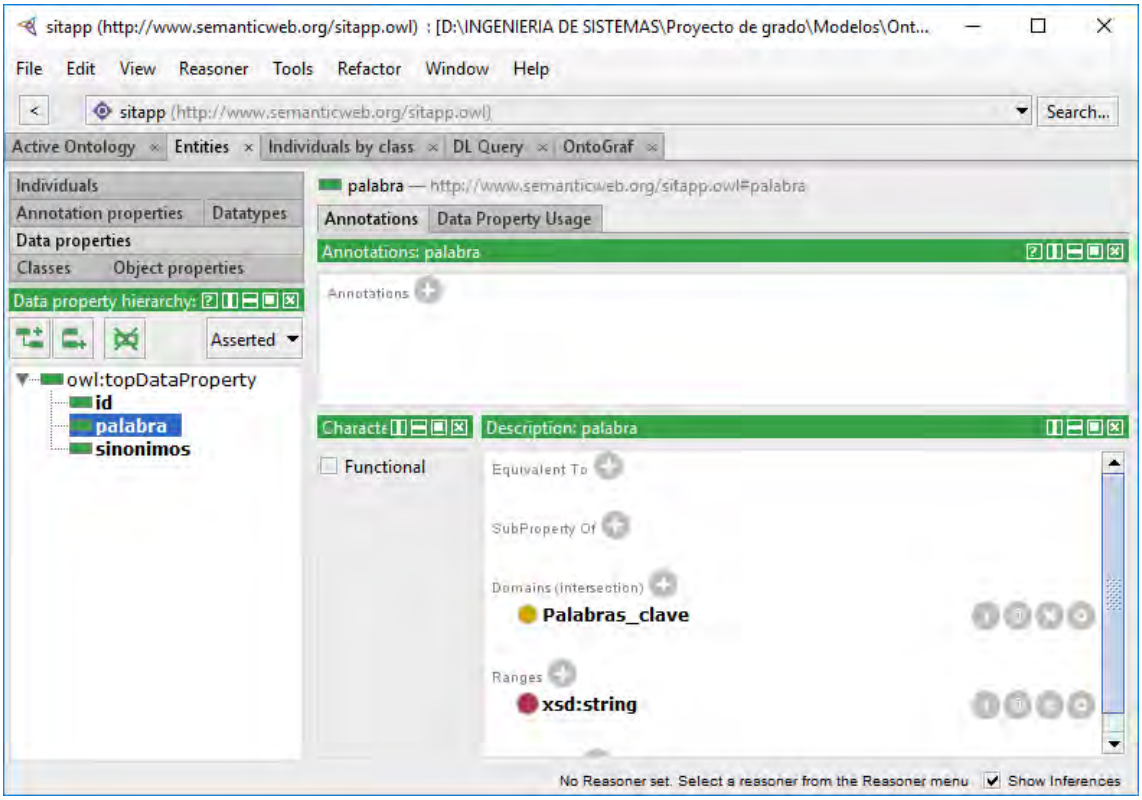
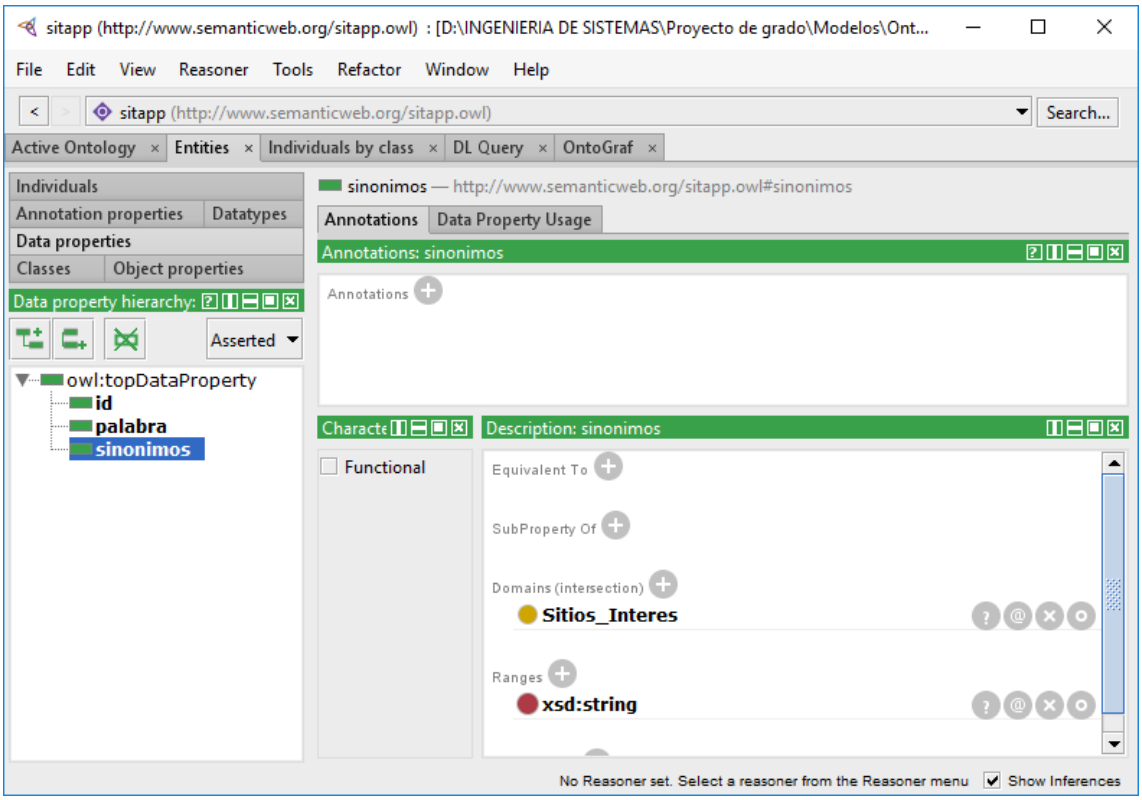


Figura 10. Descripción de la propiedad “id”.



5. **Creación de los individuos por clase.**

Los individuos se crean en la pestaña secundaria individuals, cada individuo posee 3 propiedades que se describen a continuación:

- Types: Clase a la que pertenece (Sitios_interes o Palabra_clave).
- Object property assertions: relación por la cual es descrito para el caso de los individos de la clase Sitios_interes.
- Data property assertions: propiedades de los datos que posee cada individuo, siendo id y sinónimo para los individuos que pertenecen a la clase Sitios_interes y palabra para los individuos que pertenecen a la clase Palabras_clave.

A continuación se muestran dos ejemplos de como sería el resultado final de los individuos por cada clase:

Figura 11. Ejemplo individuo para la clase Sitios_interes.

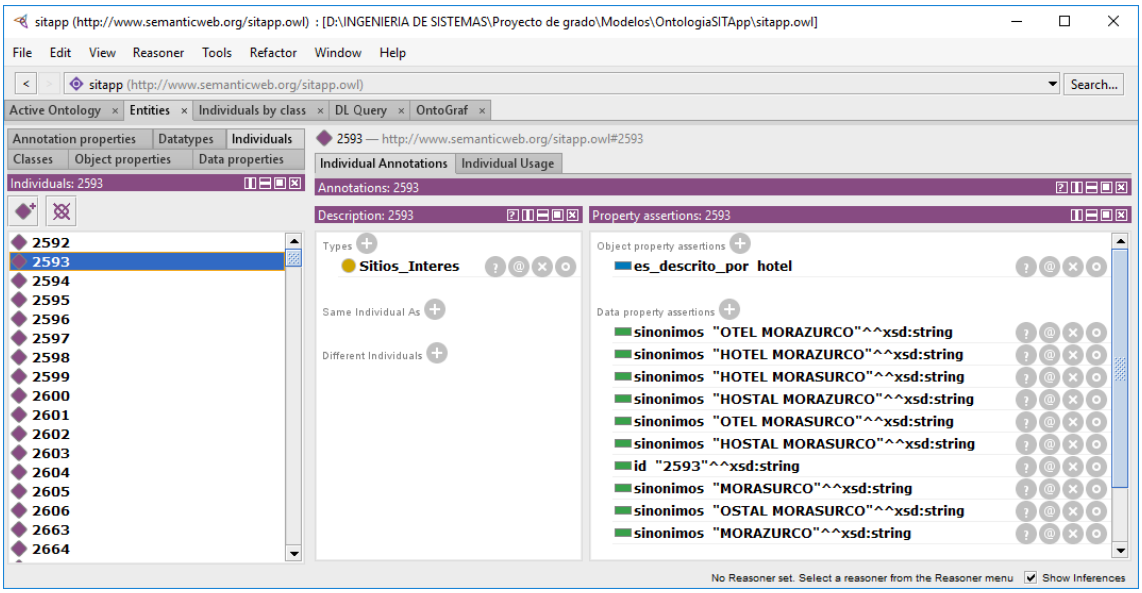
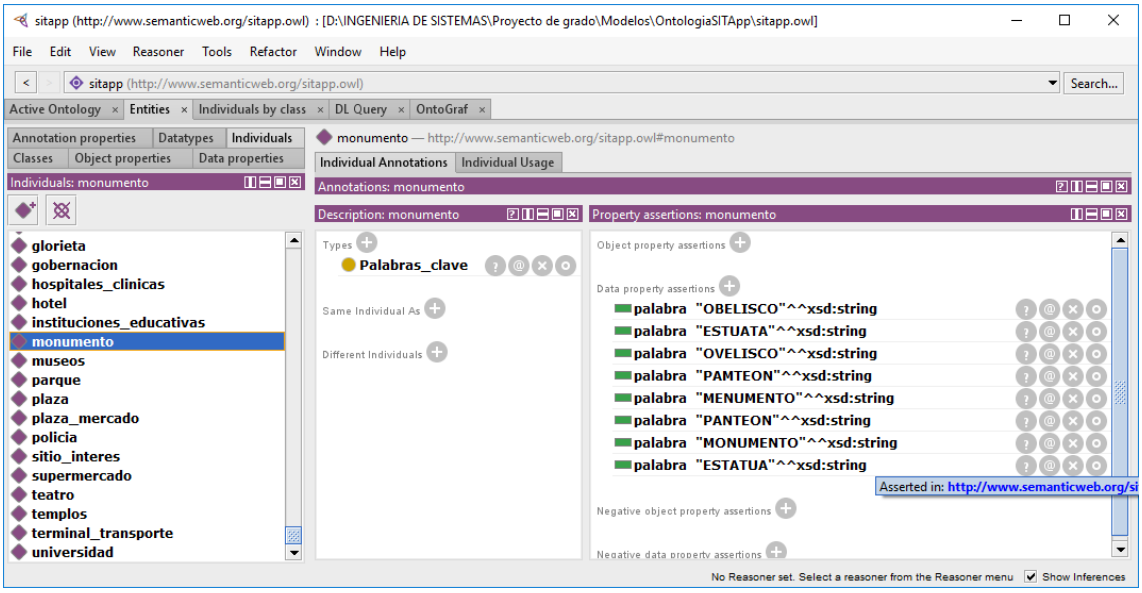


Figura 12. Ejemplo individuo para la clase Palabras_clave.


sitapp (http://www.semanticweb.org/sitapp.owl) : [D:\INGENIERIA DE SISTEMAS\Proyecto de grado\Modelos\OntologiaSITApp\sitapp.owl]
File Edit View Reasoner Tools Refactor Window Help
sitapp (http://www.semanticweb.org/sitapp.owl)
Search...
Active Ontology
Entities
Individuals by class
DL Query
OntoGraf
Annotation properties
Datatypes
Individuals
Classes
Object properties
Data properties
Individuals: monumento
glorieta
gobernacion
hospitales_clinicas
hotel
instituciones_educativas
monumento
museos
parque
plaza
plaza_mercado
policia
sitio_interes
supermercado
teatro
templos
terminal_transporte
universidad
monumento — http://www.semanticweb.org/sitapp.owl#monumento
Individual Annotations
Individual Usage
Annotations: monumento
Description: monumento
Types
Palabras_clave
Same Individual As
Different Individuals
Property assertions: monumento
Object property assertions
Data property assertions
palabra "OBELISCO"^^xsd:string
palabra "ESTUATA"^^xsd:string
palabra "OVELISCO"^^xsd:string
palabra "PAMTEON"^^xsd:string
palabra "MENUMENTO"^^xsd:string
palabra "PANTEON"^^xsd:string
palabra "MONUMENTO"^^xsd:string
palabra "ESTATUA"^^xsd:string
Asserted in: http://www.semanticweb.org/si
Negative object property assertions
Negative data property assertions
No Reasoner set. Select a reasoner from the Reasoner menu
Show Inferences