

# Red Social con Geo-Localización de lugares basado en tecnología Ruby on Rails

Edmundo Figueroa Herbas

18 de diciembre de 2012

# Índice general

<b>1. Introducción</b>	<b>1</b>
1.1. Antecedentes . . . . .	1
1.2. Descripción del problema . . . . .	2
1.3. Objetivo general . . . . .	2
1.4. Objetivos Específicos . . . . .	2
1.5. Justificación: . . . . .	3
<b>2. Marco Referencial</b>	<b>4</b>
2.1. Ruby on Rails . . . . .	4
2.2. Base de Datos . . . . .	6
2.2.1. PostgreSQL . . . . .	6
2.2.2. PostGis . . . . .	7
<b>3. Ruby on Rails y patrones Web 2.0</b>	<b>9</b>
3.1. Porque usar Ruby on Rails para desarrollar una aplicación web ? . . . . .	9
3.2. Patrones de diseño de la Web 2.0 . . . . .	11
3.2.1. Patrones de diseño . . . . .	11
3.2.2. REpresentational State Transfer (REST) . . . . .	12
3.2.3. MVC . . . . .	14
3.2.4. Mashup . . . . .	17
3.3. Conclusión . . . . .	20
<b>4. Geolocalización</b>	<b>22</b>
4.1. Definiciones . . . . .	23
4.2. Sistema de Coordenadas para datos Geográficos . . . . .	23
4.2.1. Coordenadas geocéntricas (X,Y,Z) . . . . .	24
4.2.2. Coordenadas Geograficas . . . . .	25
4.2.3. Coordenadas Proyectadas . . . . .	26

4.2.4. Que se usó en la Aplicación . . . . .	27
4.3. Implementación . . . . .	27
4.4. Conclusión . . . . .	31
<b>5. Descripción de la implentación</b>	<b>32</b>
5.1. Generación de las rutas . . . . .	32
5.2. Lectura de los archivos Shapefile . . . . .	33
5.3. Los lugares . . . . .	33
5.3.1. La geolocalizacion de los lugares . . . . .	34
5.4. Modelo de base de datos . . . . .	35
5.5. Camino minimo . . . . .	35
5.6. Modelos ? . . . . .	36
5.7. Diagrama de casos de uso ? . . . . .	36

# Capítulo 1

## Introducción

La ciudad cuenta con lugares turísticos o de interés y es fácil perderse al no tener conocimiento del nombre de las calles, movilidades que circulan por la zona o directamente no saber a dónde ir, un turista que viene por primera vez tiene el deseo de conocer un restaurant criollo o el comprador que necesita localizar el cajero automático mas cercano, toda esta información es difícil de conocer o conseguir ya que la gente del lugar no conoce el destino deseado o pueden existir muchas razones por lo cual es fácil perderse y el tiempo es uno de los recursos más preciados en la actualidad.

Los lugares turísticos son de especial interés por parte de gente que viene a conocer la ciudad y es una gran fuente de ingreso por parte de gente del lugar como también de empresas que se dedican al turismo, lo más importante el turista es contar con información confiable del lugar que va a visitar, esta información es un gran aliciente para el crecimiento de esta industria.

### 1.1. Antecedentes

Actualmente existen blogs o redes sociales dedicadas al intercambio de información, pero la gran mayoría se basa solamente en intercambiar opiniones o gustos sobre determinados lugares, o se centran exclusivamente en los lugares de más alta afluencia turística por ejemplo: el salar de Uyuni, el lago Titikaka o por fechas festivas por ejemplo el carnaval de Oruro.

Alguno de estos portales son [exploroo.com](http://exploroo.com), [dopplr.com](http://dopplr.com), [tripwolf.com](http://tripwolf.com), que ofrecen servicios como ser estado del tiempo, búsqueda de hoteles, con

opciones como poder programar una guía turística personalizada, pero con la desventaja que para nuestro país la información es tan escasa como ambigua, estos portales son de gran utilidad para llegar al destino pero no existe la información adecuada para los usuarios que quieran moverse y conocer la ciudad.

## 1.2. Descripción del problema

En una ciudad o región donde existen muchos lugares de interés para propios o extraños siempre falta una guía actualizada que nos permita movernos o tomar decisiones, las guías disponibles son generalmente impresas y solo llevan un registro de los lugares más destacados o de aquellos que pagan el derecho de aparecer en la guía, además que no se tiene la certeza si la información es actual.

En una ciudad los cambios se dan de un día para otro, locales que cierran, calles que cambian de dirección, rutas de movilidades (trufis, taxitrufis) que son modificadas o creadas, donde tener información actual es primordial.

Actualmente la Universidad Mayor de San Simon, no cuenta con un mapa actualizado de las locaciones internas, y para los estudiantes nuevos o personas que necesitan hacer algun tramite, se hace difícil el movilizarse dentro del Campus Universitario.

## 1.3. Objetivo general

Construir un Prototipo de Red Social para localización de lugares con tecnología Ruby on Rails (RoR).

## 1.4. Objetivos Específicos

- Analizar el framework Ruby on Rails para el desarrollo de aplicaciones web2.0
- Proponer 3 patrones web2.0 aplicables al problema.
- Implementar 3 Patrones web2.0 con tecnología RoR.
- Analizar y construir un modulo con características de geo-localización para la Red Social.

**1.5. Justificación:**

La gran acogida de la población en general por las redes sociales, hace hincapié en el desarrollo de un sistema de que sea ágil y eficiente en el manejo de información, ayudándose en las tecnologías presentes en la web2.0 y que tenga una gran cobertura, mediante el cual ayude a conocer una locación o decidir al lugar a donde nos podemos dirigir indicando una ruta optima.

## Capítulo 2

# Marco Referencial

### 2.1. Ruby on Rails

Ruby on Rails es un framework diseñado para desarrollar aplicaciones web, y está construido sobre el lenguaje de programación Ruby, Ruby fue creado alrededor de 1993 por Yukihiro “Matz” Matsumoto de Japón, y liberado al público en 1995, y desde entonces fue ganando popularidad y reputación gracias al aporte de una gran variedad de programadores, que realzan la sintaxis elegante y el código limpio que se genera, Ruby es un lenguaje de programación multiparadigma ya que implementa programación Orientado a Objetos, programación Funcional así como también programación Imperativa.

Ruby on Rails fue creado en el 2004 por David Heinemeier Hansson durante el desarrollo de Basecamp, una aplicación de gestión de proyectos, y una vez que se necesitó para otros proyectos, el equipo de desarrollo extrajo el core de funcionalidad el cual fue presentado al público en julio del 2004 con el nombre de Ruby on Rails, como proyecto Open Source bajo una licencia MIT, desde entonces tuvo un gran crecimiento impulsado por la comunidad de usuarios que continuamente están desarrollando nuevas características, limpiando bugs y creando gemas<sup>1</sup>. La última versión de Rails es la 3.2 publicado en enero del 2012 y actualmente está en su revisión 3.2.8 presentado en agosto del 2012, demostrando que el equipo de desarrollo de Rails esta trabajando constantemente en mejorar este framework que actualmente está entre los mejores en desarrollo web.

---

<sup>1</sup> Los plugins o complementos, en el lenguaje Ruby son llamados **gemas**

El núcleo de funcionalidad de Rails es un conjunto de funciones llamadas *Railties*:

**Active Record** Es una implementación del patrón Object-Relational Mapping(ORM), que mapea las tablas de la Base de datos relacional en clases, filas en objetos y columnas en atributos de los objetos.

**Active Support** Es el componente de Rails responsable de proporcionar extensiones del lenguaje Ruby, utilitarios y funciones primordiales a la hora de realizar cualquier tarea en el desarrollo de la aplicación.

**Action Mailer** Permite enviar correo electrónico (email) desde la aplicación usando un modelo y vistas.

**Action Pack** Es el responsable de manejar y responder los request del navegador web. Provee las herramientas para el **routing**, define los **controladores**, y genera las respuestas renderizando las **vistas**. En resumen, Action Pack maneja las capas de la vista y el controlador del paradigma MVC.

Otra de las características de Rails es la facilidad para escribir Pruebas, en realidad Rails sugiere el modelo de Desarrollo guiado por Pruebas(TDD<sup>2</sup>), que consiste en 3 pasos

1. **Rojo**, la prueba falla
2. **Verde**, la prueba pasa
3. **Refactorizar**, limpiar el código

Para este proceso, Rails ofrece primeramente el módulo Test::Unit, pero también se pueden encontrar variadas herramientas para llevar a cabo esta tarea.

Ruby on Rails es un framework MVC, que implementa los principios REST, No te Repitas<sup>3</sup>, Convención sobre Configuración, estas características de Rails están explicadas con mayor detalle en el capítulo 3.

---

<sup>2</sup>Test-Driven Development, por sus siglas en Inglés

<sup>3</sup>DRY, Don't Repit Yourself



## 2.2. Base de Datos

En una aplicación web es necesario alguna forma de persistencia de datos, en especial si se están usando datos complejos y en gran cantidad, para realizar esta tarea, la base de datos es un factor primordial. Rails maneja la base de datos mediante un ORM, por lo tanto la base de datos que se utilice no es tan excluyente, en este caso se utilizó *PostgreSQL* como base de datos relacional.

### 2.2.1. PostgreSQL

PostgreSQL es un sistema de gestión de bases de datos objeto-relacional, Open Source y distribuido bajo licencia BSD. PostgreSQL utiliza un modelo cliente/servidor y usa multiprocesos en vez de multihilos para garantizar la estabilidad del sistema. Un fallo en uno de los procesos no afectará el resto y el sistema continuará funcionando. La última versión de PostgreSQL es la 9.2, su desarrollo comenzó hace más de 16 años, y cuenta con una gran comunidad que aporta con el desarrollo, testeo de nuevas versiones. PostgreSQL está considerada como una de los mejores *Sistemas de gestión de bases de datos*, es muy completo y está muy bien documentado<sup>4</sup>. Entre sus características se pueden nombrar las siguientes.

- Es una base de datos 100 % ACID<sup>5</sup>
- Integridad referencial
- Replicación asincrónica/sincrónica
- Múltiples métodos de autenticación
- Disponible para Linux y UNIX en todas sus variantes
- Funciones/procedimientos almacenados
- Soporte a la especificación SQL

Personalmente se escogió trabajar con PostgreSQL como DBMS porque cuenta con una extensa documentación, y gracias a su carácter “Open Source”, y su gran flexibilidad en poder definir nuevos tipos de datos, se hace posible que empresas como **Refractions Research** puedan crear recursos como PostGIS, necesario para trabajar con datos geográficos ó espaciales.

---

<sup>4</sup> <http://www.postgresql.org/docs/9.2/static/>

<sup>5</sup> ACID es un acrónimo de Atomicity, Consistency, Isolation and Durability

### 2.2.2. PostGis

PostGIS es un módulo que añade soporte de objetos geográficos al DBMS PostgreSQL, convirtiéndola en una base de datos espacial para su utilización en un Sistema de Información Geográfica (SIG<sup>6</sup>).

El desarrollo de PostGIS está a cargo de **Refractions Research**, está liberada con la *Licencia pública general de GNU*, declarándola como software libre y lo protege de cualquier intento de apropiación.

PostGIS implementa la especificación “SFSQL” (Simple Features for SQL, define los tipos y funciones que necesita implementar cualquier base de datos espacial) de la OGC (Open Geospatial Consortium, es un consorcio internacional, formado por un conjunto de empresas, agencias gubernamentales y universidades, dedicado a desarrollar especificaciones de interfaces para promover y facilitar el uso global de la información espacial).

PostGIS al igual que PostgreSQL tiene una documentación bastante extensa, y cuenta con equipo de desarrollo que continuamente va sacando nuevas versiones, actualmente se encuentra la versión 2.0.1, pero para el desarrollo de la aplicación se hizo uso de la versión 1.5.5.

PostGIS es gratis, pero no por ello es una herramienta de baja calidad, al contrario se la considera una herramienta de nivel empresarial, y muchas instituciones la están usando de manera exitosa<sup>7</sup>, aparte de numerosas aplicaciones

Manejar los datos geográficos con PostGIS es sencillo y muy eficiente, por esta razón se utilizó esta herramienta, pero para conseguir la ruta óptima entre 2 puntos se necesitaba el uso del algoritmo de Dijkstra y para PostGIS existe el módulo **PgRouting**, que tiene implementado este algoritmo.

#### pgRouting

pgRouting es una extensión de PostGIS para proveer funcionalidades de ruteo espacial. pgRouting es un desarrollo posterior de pgDijkstra y actualmente está siendo mantenido por Georepublic, la última versión estable es la 1.05, y es la que fue usada para desarrollar el sistema.

---

<sup>6</sup> Es bastante común utilizar el acrónimo en Inglés, Geographic Information System (GIS), de ahí viene el término de PostGIS = Postgres + GIS

<sup>7</sup> <http://www.postgis.org/documentation/casestudies/>

Las ventajas del ruteo en la base de datos son:

- Los datos y atributos pueden ser modificados desde varios clientes, como Quantum GIS y uDig a través de JDBC, ODBC, o directamente usando Pl/pgSQL. Los clientes pueden ser PCs o dispositivos móviles.
- Los cambios pueden ser reflejados instantáneamente a través del motor de ruteo. No hay necesidad de hacer cálculos previos.
- El parámetro de “costo” puede ser calculado dinámicamente a través de SQL y su valor puede provenir de múltiples campos y tablas.

pgRouting provee funciones para:

- Camino mínimo (Dijkstra): algoritmo de ruteo sin heurística
- Camino mínimo (A-Star): ruteo para conjunto de datos grandes (con heurística)
- Camino mínimo (Shooting-Star): ruteo con restricciones de giro (con heurística)
- El problema del viajante (TSP: Traveling Salesperon Problem)
- Cálculo de ruta (Isolíneas)

## Capítulo 3

# Ruby on Rails y patrones Web 2.0

### 3.1. Porque usar Ruby on Rails para desarrollar una aplicación web ?

La gran propaganda de Ruby on Rails (RoR) o más sencillamente Rails se basa en el rápido desarrollo de aplicaciones web, conocido como agile development. Como parte de su construcción, Rails maneja las filosofías *DRY*<sup>1</sup> y *convención sobre configuración*.

**No te repitas(DRY)** según el creador de RoR, David Heinemeier Hansson, significa que cada pieza de conocimiento en un sistema debe ser declarado en un solo lugar.[1] Esto lo logra gracias al patrón Modelo-Vista-Controlador<sup>2</sup>, y el lenguaje multiparadigma Ruby sobre el cual está construido Ruby on Rails.

**Convención sobre configuración** significa que Rails tiene parámetros por defecto para casi todos los aspectos que mantiene unida una aplicación, ya que se logra observar que la gran mayoría de aplicaciones web compartían la misma configuración inicial, siguiendo las convenciones de Rails se llega a simplificar el código escrito en una aplicación, también se agiliza el desarrollo ya que no es necesario tomar decisiones sobre características básicas que se necesita implementar.

---

<sup>1</sup>Don't Repeat Yourself

<sup>2</sup>MVC

David Heinemeier cita en su libro [1], que *“Rails es Ágil porque simplemente la agilidad es parte de su construcción”*.

Se puede analizar esta afirmación teniendo en cuenta los principios del **manifiesto por el desarrollo ágil de software**<sup>3</sup>:

- **Individuos e interacciones** sobre procesos y herramientas
- **Software funcionando** sobre documentación extensiva
- **Colaboración con el cliente** sobre negociación contractual
- **Respuesta ante el cambio** sobre seguir un plan

Rails se enfoca bastante en conseguir un prototipo funcional en muy poco tiempo y sobre ese prototipo seguir incrementalmente hasta conseguir una aplicación de calidad.

Los más grandes obstáculos que se enfrenta una aplicación en el tiempo es el mantenimiento y escalabilidad, actualmente se estima que existen 230,000 websites[2] desarrolladas sobre RoR entre ellas se puede nombrar a GitHub, Hulu, Yellow Pages. Son sitios con miles de visitas diarias con una alta carga del servidor y son un claro ejemplo de que Rails puede manejar sitios de alto perfil.

Los detractores de Rails sostienen que escalar una aplicación construida sobre RoR es muy difícil pero los defensores argumentan que lo que se tiene que escalar es el código de la aplicación no el framework.

Twitter nació sobre Ruby on Rails y no fue hasta que era un servicio usado a nivel mundial y manejaba millones de request por día que empezaron a surgir problemas debido a que Ruby no estaba optimizado para un trabajo muy pesado, según palabras de Alex Payne, Twitter developer, “Ruby es lento”[3]. Actualmente Twitter migró su backend a Scala, framework basado en Java(que esta mas optimizado que Ruby), pero para su front-end no cambian a Rails.[4]

Se puede agregar que Rails es una muy buena opción a la hora de empezar cualquier proyecto web, ya que implementa las herramientas necesarias para un desarrollo ágil, sólido y de calidad respaldado por un modelo de

---

<sup>3</sup><http://agilemanifesto.org/iso/es/>

desarrollo basado en pruebas(TDD<sup>4</sup>), las filosofías DRY y convención sobre configuración. y cuando la aplicación haya crecido y empiecen a aparecer los problemas es decisión de los programadores el ver si mantener el código actual y parchearlo o cambiar de tecnología para mejorar el rendimiento y la experiencia del usuario

## 3.2. Patrones de diseño de la Web 2.0

Que es la Web2.0 ?, primeramente se debe explicar que este término fue acuñado por 1999 para describir paginas web que usaban tecnologías mas alla de las simples estaticas paginas web.

No fue hasta que en el 2004 en la conferencian sobre la Web2.0 que se popularizo este termino, y asi mismo como la Web que evoluciona, la definicion se actualiza con el tiempo, y Tim O'Reilly trato de definirla en su articulo “*What is Web 2.0*”[5], articulo que se puede considerar como la guia de referencia para cualquier persona que quiera entender que es la Web2.0 y sus origenes, en un articulo posterior “*Web 2.0 Compact Definition: Trying Again*”[6] del que se puede extraer la siguiente definición:

“Web 2.0 is the business revolution in the computer industry caused by the move to the Internet as a platform, and an attempt to understand the rules for success on that new platform. Chief among those rules is this: build applications that harness network effects to get better the more people use them.”

—Tim O'Reilly

En resumen se puede definir que una aplicación web2.0 es aquella que mejora y crece con la participación activa de sus usuarios.

“Software que mejora mientras más gente la usa” [5]

### 3.2.1. Patrones de diseño

Un patrón de diseño es una solución general, reusable y flexible que describe cómo resolver algún problema general en el desarrollo de software, un

---

<sup>4</sup>Test Driven Development

patrón puede ser usado y modificado segun el problema al cual se esta aplicando.

Se pueden observar los siguientes patrones de diseño en la aplicación:

- **REST**
- **MVC**
- **Mashup**

### 3.2.2. REpresentational State Transfer (REST)

REST es un término descrito por Roy Fielding en su tesis doctoral “*Architectural Styles and the design of Network-based Software Architectures*”[7], describe estilos arquitectónicos de sistemas interconectados por red.

REST es un estilo arquitectónico que especifica cómo los recursos van a ser definidos y direccionados, especifica la importancia del protocolo *cliente-servidor-sin estado*, ya que cada request tiene toda la información necesaria para entenderla.

En el contexto de Rails, REST significa que los componentes del sistema por ejemplo los usuarios son modelados como recursos que pueden ser creados, leídos, actualizados y borrados, estas acciones corresponden a las operaciones CRUD (Create, Read, Update, Delete) de las base de datos relacionales y a los cuatro operaciones fundamentales POST, GET, PUT, DELETE definidos en el protocolo HTTP.

En Rails el estilo de desarrollo RESTful<sup>5</sup> ayuda a determinar acerca de qué controlador y cuál será la acción que se ejecutará, solamente procesando el request HTTP hecho al servidor.

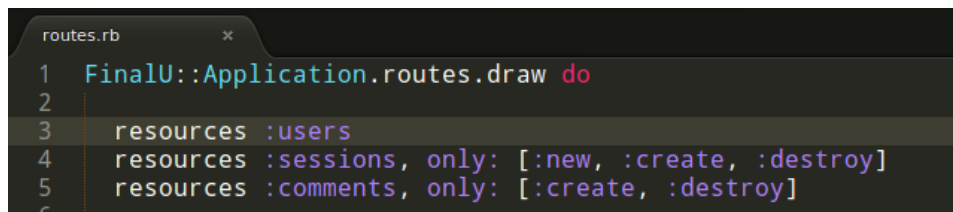
**GET** es la operación HTTP más común, es usado para leer datos en este caso paginas, se puede leer como “get a page”, **POST** es la operación que se usa cuando se ejecuta un formulario, en la convención de Rails **POST** se usa para crear objetos o recursos, **PUT** se usa para actualizar objetos, **DELETE** es usado para borrar objetos. Los browsers actuales no son capaces de manejar las operaciones **PUT** y **DELETE** de forma nativa, por lo que Rails usa un pequeño truco que consiste en declarar el método que se esta

---

<sup>5</sup>se denomina RESTful a los sistemas que siguen los principios REST

enviando en un *hidden field* en el formulario HTML.

Para lograr todo este comportamiento es necesario declarar, en el archivo que controla las rutas dentro de la aplicación, **routes.rb**, que el recurso **user** es *restful*, tal como se muestra en la figura 3.1



```

1 FinalU::Application.routes.draw do
2
3   resources :users
4   resources :sessions, only: [:new, :create, :destroy]
5   resources :comments, only: [:create, :destroy]
6

```

Figura 3.1: config/routes.rb

La figura 3.1 muestra como se declara a **users** con todas las acciones restful los cuales se listan en el cuadro 3.2.2, Rails también permite declarar solamente algunas acciones restful, como se ve el recurso **sessions** solamente tiene las acciones de **new**, **create** y **destroy**.

HTTP request	URL	ACCIÓN	USADO PARA
GET	/users	index	mostrar todos los usuarios
GET	/users/new	new	genera un formulario HTML para crear un nuevo usuario
POST	/users	create	crea un nuevo usuario
GET	/users/1	show	muestra un usuario especifico
GET	/users/1/edit	edit	genera un formulario HTML para editar un usuario
PUT	/users/1	update	actualiza los datos de un usuario especifico
DELETE	/users/1	destroy	destruye un usuario

Cuadro 3.1: las posibles rutas que se generan

Tal como se ve en el cuadro 3.2.2, Rails maneja los request HTTP de acuerdo con el tipo de llamada que se realice, este trabajo lo realiza el **rou-**



**ter**, que reconoce las URLs y los despacha a una **acción** del controlador, todo este proceso ya esta implementado en el nucleo de Rails por lo tanto es automático y el programador no necesita mas configuración que la mostrada en la figura 3.1, obedeciendo al principio de *Convención sobre configuración*

Por ejemplo, si se genera una petición GET hacia la dirección `/usuarios/1` el servidor interpreta la dirección y responde mostrando la información del usuario “1” ejecutando la acción **show** del controlador `usuarios` y en cambio si se genera una petición PUT a la misma dirección `/usuarios/1` el router procesa la información y ejecuta la acción **update** del controlador **usuarios** actualizando la información del usuario “1”.

Esta convención de Rails ayuda a entender de mejor manera el flujo que tiene un recurso, las URL son legibles y únicos para cada recurso. La implementación de los recursos se hace de forma mas limpia y ordenada, situaciones que son claves para el mantenimiento y la extensibilidad del sistema.

### 3.2.3. MVC

MVC (Modelo Vista Controlador) es un patrón arquitectónico que separa los datos de la aplicación en la interfaz del usuario y la lógica del negocio, en tres partes cada uno especializado para su tarea, la vista maneja lo que es la interfaz del usuario, puede ser gráficamente o solo texto, el controlador interpreta las entradas del teclado, mouse, o los cambios de la vista de la mejor forma posible y finalmente el modelo maneja el comportamiento de los datos de la aplicación.[8] Concepto que se desarrolló en 1979 por Trygve Reenskaug el cual da una solución al problema de separar la lógica del negocio de la lógica de la presentación.

Rails esta construido sobre el patrón MVC, esto significa que para cada pieza de código existe un lugar predeterminado y todas las piezas de código de la aplicacion interactuan de forma predeterminada.

**Modelo** Representa la información o los datos y contiene las reglas o métodos para manipular estos datos. En el caso de Rails, los modelos son usados principalmente para manejar la interacción con las tablas de la Base de Datos, en la que cada tabla corresponde a un modelo en la aplicación, para nombrar a estos modelos existe una simple convención que es la de nombrar a la tabla en forma plural, mientras que el

modelo tiene el nombre en singular. El Modelo, en Rails es el principal encargado de manejar la *logica del negocio*.

**Vista** Representa la interfaz de la aplicación. En Rails las vistas son archivos HTML con código Ruby embebido<sup>6</sup> que realiza la tarea de representar los datos. Las Vistas son las que generan los datos que son enviados al navegador web.

**Controlador** Es el encargado de interactuar entre el modelo y la vista, procesando los datos enviados en el request del navegador web, llamando a métodos del modelo para conseguir *información* de la base de datos, posteriormente el controlador envia esta *información* a la vista.

El Controlador y la Vista son los encargados de manejar la *lógica de la presentación*.

Se puede apreciar el comportamiento de este patrón, en la figura 3.2

1. se genera un request “**GET** /users/1” desde el navegador web.
2. este request primeramente es analizado por el **router**, que siguiendo el principio REST lo direcciona a la acción *show* del **controlador** *users*.
3. el controlador extrae el id del usuario de la llamada<sup>7</sup> **GET** /users/1.
4. la acción *show* del controlador se encarga hacer la llamada al **modelo** “User”, mediante el metodo `find_by_id(1)`.
5. el modelo “User”, inicialmente valida o modifica los datos recibidos.
6. el modelo “User”, mediante el modulo ActiveRecord extrae la información de la Base de datos.
7. esta información es almacenada en una *instance variable*<sup>8</sup> `@user`
8. la vista “show.html.erb”<sup>9</sup> incluye la variable `@user` mediante los *ERB output tag* (`<%= %>`)
9. este template es procesado en el servidor, y renderizado como una pagina HTML que es presentada en el navegador web.

---

<sup>6</sup> ERB (Embedded Ruby)

<sup>7</sup> request

<sup>8</sup> los *instance variable* son variables especiales que empiezan con una “@” y están disponibles en la vista

<sup>9</sup> la extensión **erb** indica que la pagina HTML puede tener contener código Ruby

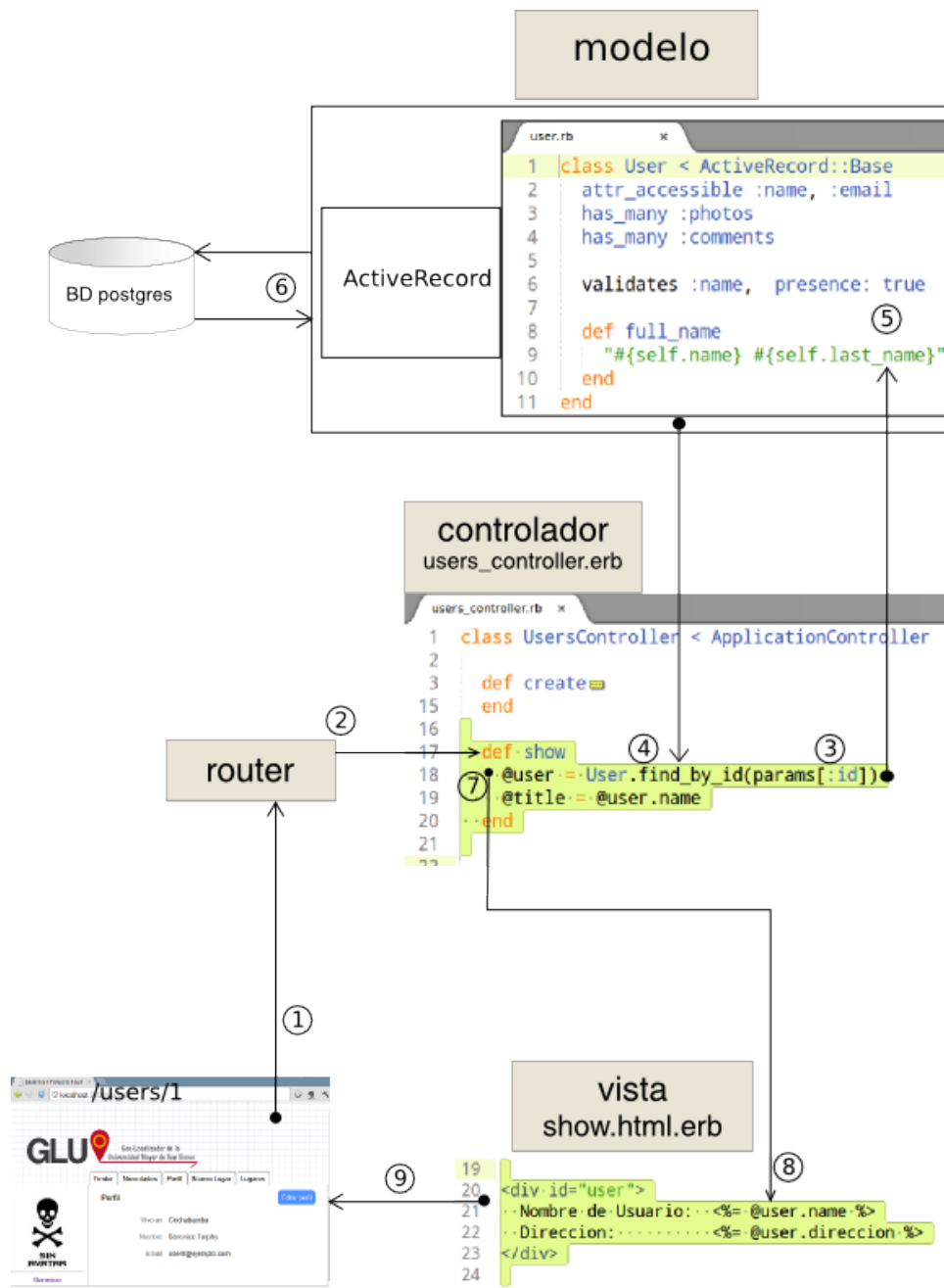


Figura 3.2: Un diagrama detallado del patrón MVC en Rails

La capacidad de declarar una variable en el controlador y que esta variable esté disponible en la vista, así como toda la configuración que envuelve el comportamiento del patrón MVC ya esta implementada y no es necesario escribir archivos de configuración, El patrón MVC es la base de la aproximación RESTful de Rails, así como de las filosofías *DRY* y *Convención sobre Configuración*, en conclusión se pudo apreciar que al usar esta arquitectura el código se vuelve mas ordenado, con el consiguiente resultado de ser un código mucho mas limpio, legible y entendible.

### 3.2.4. Mashup

Mashup es una técnica en el desarrollo de aplicaciones web que consiste en combinar datos de diferentes proveedores. Un mashup usa, uno o más servicios y mezcla algunas características de estos servicios, los servicios generalmente son datos que son distribuidos mediante las APIs<sup>10</sup>.

Con la finalidad de ofrecer estos datos de forma más fácil de entender y usar para el cliente.

Los primeros mashups empezaron como experimentos con los web services que ofrecían las grandes empresas, por ejemplo uno de los primeros mashups mas importantes fue el de housingmaps<sup>11</sup> que combina los datos de Craigslist ( listas de casas, departamentos, etc, para rentar, vender ) y Google Maps ( mapas a nivel global), en una aplicación que ofrece servicios y los posiciona en un mapa. Desde entonces surgieron diferentes tipos y aproximaciones dependiendo del tipo de datos y el servicio que se quiere ofrecer.[9]

Se pueden apreciar diferentes implementaciones de mashups, los mashups orientados al consumo y los mashups orientados a las empresas, básicamente estos mashups de consumo se refieren a aplicaciones en las cuales los datos obtenidos de diferentes fuentes son mezclados y presentados al cliente, y los mashups empresariales o de negocios aparte de las fuentes de datos externas, mezclan sus propios datos en el mashup ofreciendo una mejor experiencia del usuario.

Uno de los componentes básicos de la Web 2.0 son los **datos**,[5] sobre los cuales se puede obtener un producto con valor agregado, generalmen-

---

<sup>10</sup>Application Programming Interface, es el conjunto de funciones y procedimientos que se ofrece para ser utilizado por otro software

<sup>11</sup><http://www.housingmaps.com/>

te los datos están disponibles para que sean usados, pero dependiendo del tipo de “datos”, obtenerlos y almacenarlos puede llegar a costar tiempo o dinero o ambos, es por eso que los datos son ofrecidos con o sin restricción dependiendo de las políticas de uso que tenga la empresa que ofrece los datos.

El mashup implementado se podría catalogar como un mashup de negocio, ya que se esta integrando datos externos(Google Maps API) y datos internos(locaciones dentro del campus de la UMSS), inicialmente se almacenaron varias locaciones pero eventualmente estos datos se incrementaran mediante el uso que se le de por parte de los usuarios.

Para usar el API de Google Maps es necesario importar la librería declarandola en la cabecera del documento HTML, como se ve muestra en la figura 3.3, para visualizar el mapa es necesario declarar un elemento único (se recomienda un div con id de preferencia “map” o similar) dentro del DOM de la página, figura 3.4, de esta forma se logra visualizar el mapa geográfico que ofrece Google, y su manejo es mediante el lenguaje de programación del lado del cliente, *javascript*, figura 3.5.

Para el uso de los datos internos se hizo uso intensivo de la interfaz REST de Rails, ya que se necesitaba desplegar en el mapa los lugares o locaciones de la UMSS, esto se debe a que la interfaz mashup se encuentra en el lado del cliente(navegador Web) y la información recopilada esta almacenada en el servidor(Rails), entonces necesitaba extraer los datos del servidor y presentarlos al cliente, este procedimiento se lo puede demostrar con el uso del comando *curl*, y con la información obtenida se puede desplegar la locación en el mapa.

```
$ curl http://localhost:3000/places/23.json
```

```
{
  "id": 23,
  "type": "caseta",
  "address": "",
  "lng": -66.1473755998898,
  "lat": -17.3938599682795,
  "name": "informaciones",
  "photos": [
    {
```

```
      "id": 28,  
      "desc": "informaciones",  
      "url": "photo/image/28/square_shin.jpg"  
    }  
  ]  
}
```

Como se ve en el ejemplo, el servidor responde mandando los datos en formato **json**<sup>12</sup>, que al ser un metodo comun para enviar datos desde el servidor al cliente, es facil de implementar y de manipular, y se genera atraves de una plantilla en la que se puede manipular los datos y presentarlos de la mejor forma posible. Por ejemplo para mostrar los lugares se escribio la siguiente plantilla.

```
# views/places/show.json.rabl  
  
object @place  
  attributes :id, :name, :desc, :address  
  
  node(:type) { |place| place.type_place.name }  
  node(:lng)  { |place| place.coord_geographic.x }  
  node(:lat)  { |place| place.coord_geographic.y }  
  
  child :photos do  
    attributes :id, :desc  
    node(:url) { |photo| photo.image.url(:square) }  
  end
```

De esta forma se pueden ofrecer los datos internos para que otras aplicaciones la puedan usar. Es lo que se denominaria *REST Web Services*

En la aplicación desarrollada se necesitaba el uso de mapas para que el despliegue de información sea de forma más interactiva y de fácil uso, los datos geográficos públicos generalmente están disponibles para ser usados pero para su uso se necesitaba que esté representado de forma gráfica en la pantalla de la computadora, este trabajo lo llevan a cabo empresas las

---

<sup>12</sup>Notación de Objetos de JavaScript, es un formato ligero de intercambio de datos y está basado en un subconjunto del Lenguaje de Programación JavaScript [10]

cuales generalmente cobran por el uso de los mapas, se puede apreciar que se están obteniendo datos en forma de mapas a través del API de Google Maps, Google permite usar sus datos con restricciones dependiendo de la cantidad de llamadas que se haga a su API por la aplicación que la usa.

El gran beneficio de usar los datos de Google Maps en un mashup es que nos permite desplegar información que de otra forma costaría mucho tiempo y dinero para implementar.

### 3.3. Conclusión

Los patrones no son entidades separadas, como ya se vio todos los patrones web2.0 se integran y trabajan de manera conjunta, no se podría implementar software que solo implemente algun patrón.

Los patrones empiezan como soluciones a algun problema de diseño y descritas en concepto, gracias a que los problemas y experiencias que tuvieron los programas que lograron pasar de la Web Estatica y los que nacieron en la Web2.0 están documentados y analizados, después depende de los desarrolladores el implementarlas de la mejor forma posible.

Rails se diseño, para que trabaje sobre las ultimas tecnologías disponibles, para poder desarrollar software que se pueda catalogar como “*Aplicacion Web2.0*”, pero al final que una aplicación obtenga esta etiqueta esta dada por múltiples factores, tales que pueden ser, la capacidad de entender la web como un algo que evoluciona con el tiempo, e implementar software que pueda utilizar todos los avances tecnológicos en favor de mejorar la experiencia del usuario, siempre apoyados por la interacción que supone el intercambio y flujo de ideas de la comunidad de usuarios que usan la aplicación.

```

3 <head>
4 <title><%= full_title(@title) %></title>
5 <%= include_gon %>
6 <%= stylesheet_link_tag "application", :media => "all" %>
7 <%= javascript_include_tag "http://maps.google.com/maps/api/js?sensor=false" %>
8 <%= javascript_include_tag "application" %>
9 <%= csrf_meta_tags %>
10 </head>

```

Figura 3.3: Se importa el API de Google Maps declarando en <head> del documento

```

10 .....
17 .....<article id="finder_place" class="perfil">
18 .....<div id="map"></div>
19 .....

```

Figura 3.4: El mapa se visualiza dentro de la etiqueta div con id map

```

11 .....
12 .....
13 .....init: function( opt_ ) {
14 .....var opt = opt_ || {};
15 .....var options = {
16 .....zoom: opt.zoom || 17,
17 .....center: new google.maps.LatLng( (opt.lat || -17.3937285), (opt.lng || -)
18 .....mapTypeId: google.maps.MapTypeId.ROADMAP,
19 .....disableDefaultUI: true,
20 .....navigationControl: true
21 .....}
22 .....
23 .....this.map = new google.maps.Map( $(opt.selector || "#map").get(0), options );
24 .....
25 .....},
26 .....
27 .....

```

Figura 3.5: Se captura el div map del documento y se inicializa el mapa mediante el lenguaje javascript



## Capítulo 4

# Geolocalización

La Geolocalización o Georreferenciación es un termino bastante nuevo, de hecho no aparece en el diccionario de la Real Academia Española, no obstante se lo puede definir como:

El posicionamiento en el que se define la localización de un objeto espacial (representado mediante un punto, vector, área, volumen) en un sistema de coordenadas y datum determinado. Este proceso es utilizado frecuentemente en los Sistemas de Información Geográfica.

La Georreferenciación era usada bastante en el ambito científico, y se necesitaba de instrumental y personal bastante cualificado para su manejo, pero en la actualidad la cantidad de dispositivos con capacidad para geolocalizar un objeto sobre la tierra es bastante comun, de hecho todos los smartphones actuales (celulares con Android, Iphones) traen integrados receptores GPS (Global Posicion System), y sumados a la explosión de aplicaciones que integran mapas con localización (mashups), ya que se puede tener una base de datos con información muy importante pero al final los datos son cifras, descripciones, etc. y para tomar decisiones se hace muy difícil el interpretar estos datos, aca viene en nuestra ayuda los SIG (Sistemas de Información Geografica).

Actualmente existe una explosión de aplicaciones, donde empresas, particulares y hasta donde organismos gubernamentales están haciendo uso de estas tecnologías. Y las posibilidades son diversas, por ejemplo, se quisiera planificar la construcción de un colegio se podria integrar los datos del censo con un mapa, identificando los sectores con mayor porcentaje de niños

y localizando los sectores mas propicios para realizar la construcción del inmueble. En el caso de una catástrofe natural, el tener las rutas de evacuación geolocalizadas y disponibles en un mapa de manera eficiente, ayudaria en la evaciación de las personas del lugar.

## 4.1. Definiciones

En la aplicación desarrollada se requería trabajar con datos espaciales, y para ello es necesario entender algunos conceptos envueltos en el manejo de la información geografica.

**Coordenada** Es una secuencia de n-numerós que designa la posición de un punto en un espacio n-dimensional.

**Sistema de coordenadas** Un sistema de coordenadas es un conjunto de reglas matemáticas que especifican como las coordenadas son asignadas a cada punto.

**Punto** Es la representación de una posición, topológicamente 0-dimensional (no tiene volumen, area, longitud o cualquier otra unidad multi-dimensional).

Estas definiciones estan desarrolladas en la especificación **Simple Feature Access**, la cual es mantenida por la OGC (Open Geospatial Consortium). Esta especificación define el conjunto de tipos de datos (puntos, linia, poligono, etc) y las operaciones o metodos necesarios para manejar estos datos.

## 4.2. Sistema de Coordenadas para datos Geográficos

Se podria pensar en un sistema de coordenadas como la forma de dar sentido a un *par de coordenadas*, por ejemplo cuando se ve una locación “POINT(-66.1457475 -17.3937285)”, como se interpretan estos números?. Podria ser la latitud y longitud del campus de la UMSS, o podria ser un sistema de años luz desde alguna estrella en el Universo. El sistema de coordenadas es lo que diferencia estos casos.

Una aplicación que maneja datos geograficos, generalmente trabaja con sistemas de coordenadas relacionadas con la superficie terrestre, conocidas como coordenadas espaciales (coordenadas globales), que permiten representar la tierra en 3-Dimensiones (3D), ya que esta es una Esfera (elipsoide oblato), o en una representacion de la superficie terrestre en 2-Dimensiones (2D), se pueden nombrar los siguientes:

#### 4.2.1. Coordenadas geocéntricas (X,Y,Z)

También conocido como Coordenadas Cartesianas 3D, Este sistema tiene como origen el centro de la Tierra, con el eje X y el eje Y en el plano del ecuador. El eje X pasa a través del meridiano de Greenwich, y el eje Z coincide con el eje de rotación de la Tierra.

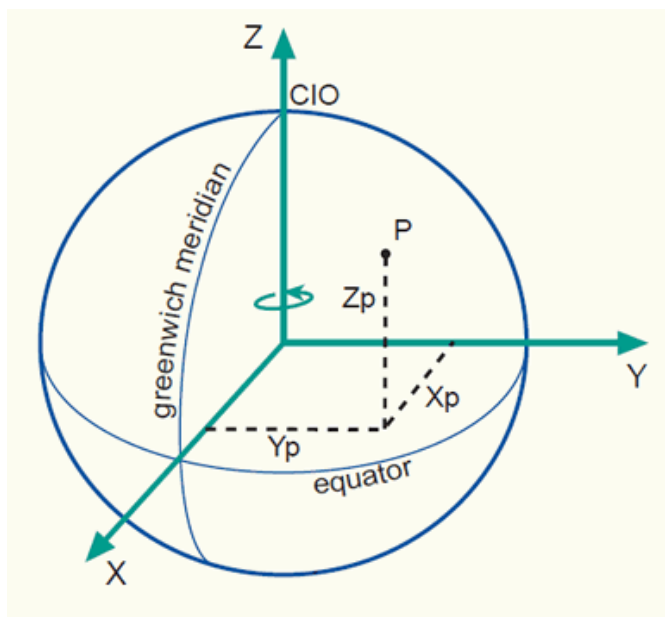


Figura 4.1: Sistema de coordenadas Geocentricas, en la figura se muestra la posición del punto P

Este Sistema de coordenadas no es muy usado en la representacion de datos.

### 4.2.2. Coordenadas Geograficas

Sistema de coordenadas Geográfico, utiliza las coordenadas angulares latitud ( $\phi$  o  $\phi$ ) y longitud ( $\lambda$  o  $\lambda$ ). Este sistema de coordenadas se expresa en grados, se lo puede representar con la forma *grados:minutos:segundos* ( $17^\circ 23' 37.4226''$  S,  $66^\circ 8' 44.691''$  W), o de la forma mas comun *grados decimales* ( $-66.1457475$  S,  $-17.3937285$  W).

El sistema de coordenadas mas amplimente usado, el que usan por defecto los sistemas GPS, es conocido como “WGS 84”, y la mayoría de las aplicaciones que manejan mapas.

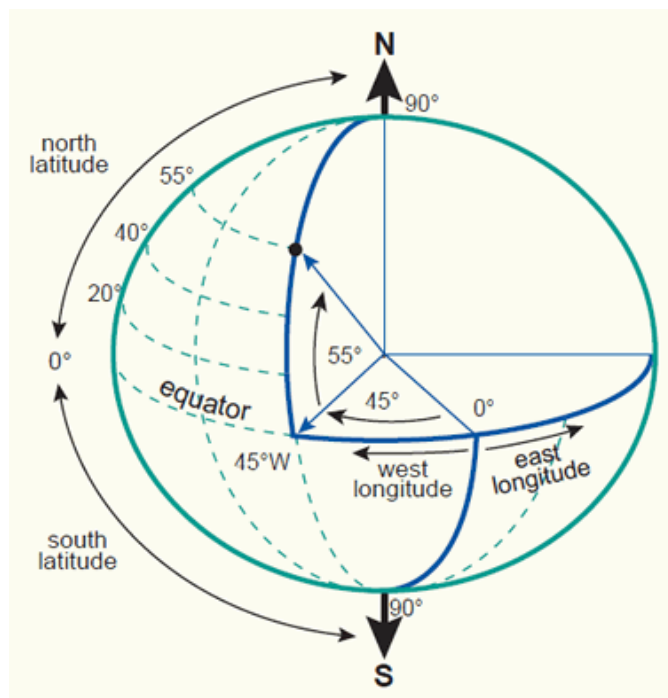


Figura 4.2: Sistema de coordenadas Geográficas, Es el sistema que maneja los mas amplimanete conocidos “latitud y longitud”

### 4.2.3. Coordenadas Proyectadas

Un sistema de coordenadas proyectadas es una representación plana y bidimensional de la tierra. Se basa en un sistema de coordenadas geográficas esféricas, pero utiliza unidades de medida lineales para las coordenadas, de forma que los cálculos de distancia y área se pueden realizar en términos de esas mismas unidades.[13]

Un sistema de coordenadas proyectadas requiere tomar la superficie esférica de la tierra y “aplanarla”, este procedimiento se lo realiza con la finalidad de tener un mapa representable en una hoja de papel así como en la pantalla de la computadora. Sin embargo este procedimiento introduce diversos tipos de distorsión por lo que existen diferentes clases de proyecciones que varían según la región que se quiere representar de la Tierra.

La proyección que usa Google Maps es la **Mercator Projection**, esta proyección está diseñada para preservar los ángulos y las formas de las líneas en forma recta, pero distorciona los tamaños y las distancias mientras más lejos se encuentran de la línea del Ecuador. Esta proyección se puede apreciar en la figura 4.3

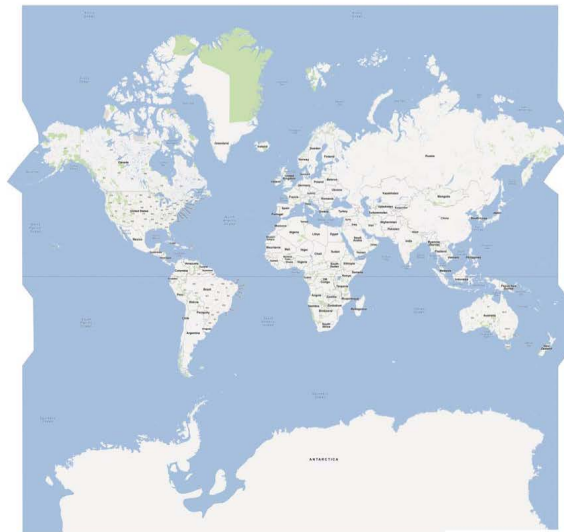


Figura 4.3: Google Maps usa la Proyección de Mercator para mostrar su mapa

Tal como se puede apreciar en la figura 4.3, la distorsión de esta proyección se hace evidente si se observa la zona de Groenlandia ya que parecería tan grande como Africa o America del Sur, cosa que no es, ya que Groenlandia es casi 14 veces mas pequeño que Africa. A pesar de esta distorsión tan marcada, la **Proyección de Mercator** es una de las mas usadas.

#### 4.2.4. Que se usó en la Aplicación

Es importante entender las diferencias entre los distintos tipos de sistemas de coordenadas porque computacionalmente realizar operaciones sobre los sistemas de coordenadas tiene un costo. Si se usara el sistema de coordenadas geográfico (WSG84) este es el más apropiado si se necesitara usar grandes extensiones de la superficie terrestre, que al ser una estructura elipsoidal el costo computacional para realizar las operaciones matemáticas de calcular distancias, intersecciones, etc. es más elevado. En cambio el uso de un sistema de coordenadas proyectado (Mercator Projection) tiene un costo computacional más bajo, ya que se estaría trabajando con un sistema geométrico.

También hay tomar en cuenta la base de datos, ya que será esta la que se encargara de manejar los datos espaciales. Al estar usando PostGIS, se puede ver que en su documentacion<sup>1</sup> que claramente exorta el uso de un sistema geometrico sobre el uso de un sistema geografico si se va trabajar con datos que cubran una pequeña area geografica. Tomando en cuenta esta recomendación y el tamaño del área de estudio (campus UMSS), se procedió a implementar en la base de datos el uso de la proyección Mercator. Se va usar Mercator sobre las otras proyecciones porque aparte de las ventajas que se mencionaron con anterioridad, Google Maps usa esta proyección y ya que se usara este mapa lo más correcto es trabajar con la misma proyección.

### 4.3. Implementación

Al implementar una aplicación con Rails, no se maneja la base de datos con ordenes SQL puro. La base de datos se la maneja atraves de un ORM (ActiveRecord). Para llevar a cabo esta tarea es necesario escribir una *migración*, el cual es un metodo que Rails interpreta y lo traduce a ordenes SQL.

---

<sup>1</sup> <http://postgis.org/documentation/manual-1.5/ch04.html>

```

class CreatePlaces < ActiveRecord::Migration
  def change
    create_table :places do |t|
      t.string :name
      t.point :coord, :srid => 3785
    end
  end
end

```

Esta *migración* creó la tabla **places** con los atributos **name** y **coord**, el atributo **name** es del tipo string que en PostgreSQL se traduciría en una columna de tipo *VARCHAR(255)*, y el atributo **coord** de tipo point con un srid<sup>2</sup> 3785, el SRID es la llave primaria de la tabla *spatial\_ref\_sys* que se crea cuando se inicializa una base de datos espacial, esta tabla provee la información necesaria para interpretar y convertir correctamente todas las coordenadas existentes, además permite definir algún tipo de sistema de coordenadas si no existe en esta tabla, el SRID 3785 esta definida en la tabla *spatial\_ref\_sys* como “Popular Visualisation CRS / Mercator”, Rails interpreta el tipo point y crea una columna de tipo *geometry*, definida por PostGIS.

El anterior código escrito en Rails se traduciría en SQL de la siguiente forma:

```

CREATE TABLE places (
  id      INTEGER      PRIMARY KEY,
  name    VARCHAR(255)
);
SELECT AddGeometryColumn(
  'places', -- nombre_tabla
  'coord',  -- nombre_columna
  3785,     -- srid
  'POINT',  -- tipo
  3         -- dimensi'on
);

```

---

<sup>2</sup> Spatial Reference System Identifier, El SRID corresponde a un sistema de referencia espacial basado en el elipsoide concreto usado para la creación de mapas de tierra plana o de tierra redonda.[14]

Al usar un framework como Rails, se tiene como ventaja que existen herramientas creadas, comprobadas y avaladas por una gran cantidad de usuarios, en este caso se precisa de herramientas que ayuden en el manejo de información geográfica, la gema **RGeo**<sup>3</sup> es la que maneja la correcta manipulación de estos datos.

Una vez configurada/implementada la base de datos se puede obtener la locación de cualquier lugar mediante el ORM.

```
$ rails console
Loading development environment (Rails 3.2.3)
> UMSS = Place.find_by_name("UMSS")
=> #<Place id: 21, name: "UMSS",
      coord: #<RGeo::Geos::PointImpl:0x52e4e60
      "POINT (-7363625.14672284 -1966786.8137609)">, ... >
> UMSS.coord_geographic.to_s
=> "POINT (-66.14857015827988 -17.394421906929086)"
```

Tal como se puede apreciar, la consola de Rails permite manipular los datos cargados en la aplicación desde la línea de comandos de Linux, en la demostración se ve que la variable **UMSS** es el resultado de la búsqueda en la base de datos mediante el modelo *Place* con la opción de buscar por el nombre, el atributo **coord** de la variable **UMSS** es un objeto de tipo *RGeo* y el punto está proyectado, y si fuera necesario también se puede obtener el mismo punto en coordenadas geográficas.

Obtener la coordenada es el primer paso, seguidamente se debe mostrarlo sobre un mapa, en este caso *Google Maps*. El API de Google Maps permite este paso declarando un objeto *Marker*, que es la imagen que muestra Google Maps, este objeto para mostrarse en el mapa requiere que se le declare un objeto tipo *LatLng*, el cual es la coordenada y se por último se necesita declarar el mapa en el cual se va mostrar el marcador. Este objeto es altamente personalizable pero básicamente este es el procedimiento para representar la coordenada en un mapa de Google Maps. Tal como se puede apreciar en la figura 4.3

---

<sup>3</sup> RGeo es una librería de datos geospaciales para el lenguaje Ruby.



```
var maker = new google.maps.Marker({  
  position: new google.maps.LatLng(UMSS.lat, UMSS.lng)  
  map: UMSS.map  
  icon: new google.maps.MarkerImage('img/gota.png');  
});
```



Figura 4.4: Google Maps representa un *punto* con un *marcador*, el cual es altamente personalizable

#### 4.4. Conclusión

Los Mapas son herramientas muy útiles a la hora de desplegar información pero realizar el mapa, crear las fórmulas matemáticas con las cuales se trabajará, determinar cómo se usarán estas fórmulas para una representación adecuada de la superficie terrestre, es una tarea muy compleja. Como programador la tarea más complicada fue determinar el tipo de mapa y el sistema de coordenadas más adecuado al trabajo.

Los términos de longitud y latitud son en un inicio, más fácilmente comprendidos que un sistema proyectado, pero no se puede tomar a la ligera una correcta comprensión del uso de los *sistemas de coordenadas* en una base de datos espacial, un mal uso de estos conceptos puede generar errores a la hora de manejar datos espaciales o en el resultado de operaciones sobre estos datos, llegando a resultados no deseados y que pueden costar mas tiempo y dinero en una posterior corrección.

La geolocalización es actualmente una tecnología y una herramienta usada en gran medida por una gran cantidad de aplicaciones web, añadiendo búsquedas y resultados personalizados a nivel país, ciudad, barrio y calle, resultando en una gran variedad de servicios y que actualmente es de gran ayuda en diferentes escenarios. La geolocalización nos ayuda a movernos por una ciudad, encontrar restaurantes, cines, transporte, etc. actualmente es una de las herramientas más usadas y desarrolladas a nivel de industria, comercio, turismo, etc. y vale la pena estudiarla y entenderla.

## Capítulo 5

# Descripción de la implementación

La aplicación a desarrollar requería de implementar un sistema de geolocalización, para tal motivo primeramente se procedió a generar un mapa de rutas sobre el cual escoger las herramientas más apropiadas para la realización de la aplicación.

### 5.1. Generación de las rutas

Se procedió a trazar un mapa de rutas del campus de la Universidad Mayor de San Simón, para tal efecto se necesitó de un GPS, el único requisito del GPS es la posibilidad de grabar en su memoria la ruta que se camina. El GPS usado fue un Garmin Nuvi 1300, es un GPS básico pero cumple con el requisito, los archivos generados son gpx, es básicamente un archivo XML estándar para compartir datos entre GPS's. Posteriormente el archivo gpx necesita ser editado por un Sistema de Información Geográfica, se usó QGIS que es un SIG open source con licencia GNU GPL multiplataforma. QGIS me permitió editar la ruta generada por el GPS, que al no ser exacto, era necesario este paso, generando un archivo shapefile, el cual es el formato estándar para manejar datos geográficos espaciales.

El mapa generado con las rutas cumple con las características de un grafo ponderado no-dirigido en el que el costo o peso de las aristas es la distancia entre los nodos y los nodos son los puntos de intersección de las rutas. Por lo tanto se puede implementar el algoritmo de Dijkstra para encontrar el camino mínimo.

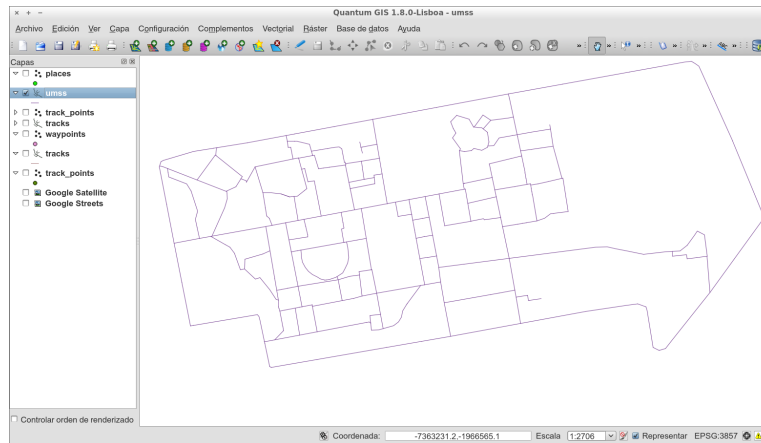


Figura 5.1: Rutas del Campus de la UMSS en QGis

## 5.2. Lectura de los archivos Shapefile

Es necesario cargar la base de datos PostGis con los shapefiles generados por QGis y para eso se uso la gema RGeo y con el siguiente metodo escrito en Ruby se carga los archivos shapefile en la base de datos.

```
def self.load_shapefile(path)
  RGeo::Shapefile::Reader.open(path, factory: FACTORY) do |file|
    file.each do |record|
      way = record.geometry.projection
      ruta = Way.create(name: record['id'],
                        dist: way.length,
                        the_geom: way[0])
    end
  end
end
```

## 5.3. Los lugares

Los lugares o sitios de interes se crean atraves de la aplicacion, ingresando a la seccion de *Nuevo lugar*, en donde es necesario llenar el formulario y hacer un click sobre el mapa en el sitio que se desea registrar el nuevo lugar.

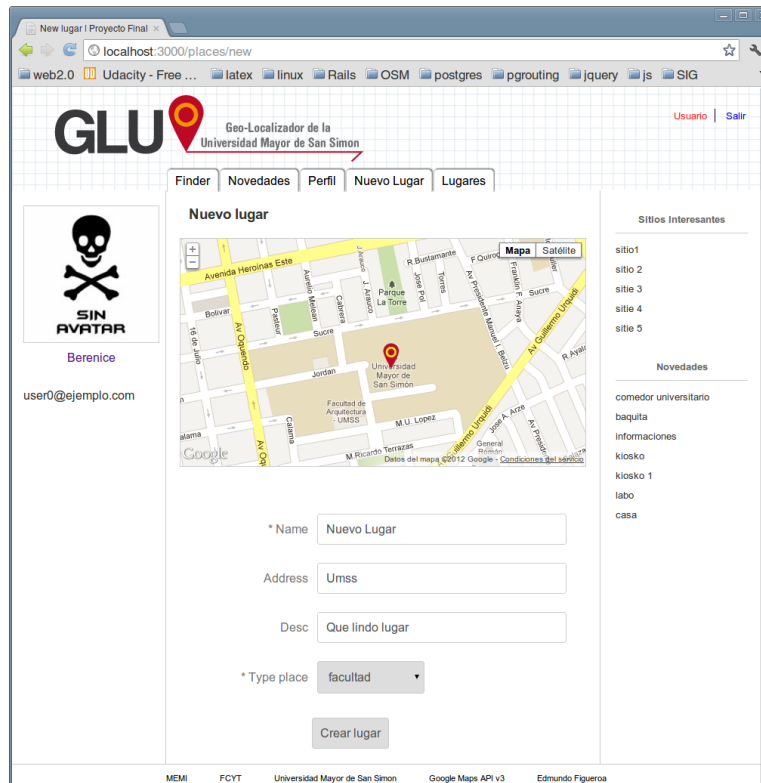


Figura 5.2: Registro de Nuevos lugares

### 5.3.1. La geolocalizacion de los lugares

Para que Rails cree un lugar es necesario los siguientes pasos:

- Declarar un objeto que maneja la proyección
 

```
FACTORY = RGeo::Geographic.simple_mercator_factory
```
- Declarar el atributo encargado de manejar el dato espacial, mediante el metodo `set_rgeo_factory_for_column` especificando la proyección.
 

```
set_rgeo_factory_for_column(:coord, FACTORY.projection_factory)
```
- LLamar a al funcion `create_coord` cuando se crea un nuevo lugar, siempre teniendo en cuenta la proyeccion en la cual se reciben los datos (*lon* y *lat* se reciben del lado del cliente en WS84) y proyectarlo para guardarlo correctamente en la base de datos (*coord* esta en la proyeccion EPSG 3785).

```

def create_coord
  if self.new_record?
    self.coord = FACTORY.point(lon, lat).projection
  else
    p self.coord
  end
end
end

```

## 5.4. Modelo de base de datos

## 5.5. Camino minimo

Una vez se tienen las rutas en la base de datos es necesario realizar algunos pasos previos para encontrar el camino minimo, pgRouting requiere de una tabla en la especifica todos los vertices existentes. Toma como argumentos la tabla de la cual se debe extraer los vertices (ways), la distancia de tolerancia (0.001), el atributo espacial (the\_geom) y su id (gid)

```
SELECT assign_vertex_id('ways', 0.001, 'the_geom', 'gid');
```

Al tener ya lista la base de datos, se procede a llamar a la funcion *shortest\_path* del modulo PgRouting de PostGis, el cual implementa el algoritmo de Dijkstra para encontrar el camino minimo.

En una computadora AMD K6 de 1.8 GHz, de 346 vertices se tiene el siguiente dato

```

-- Executing query:
SELECT * from shortest_path('select gid as id,
                             source::integer,
                             target::integer,
                             dist::double precision as cost from ways',
                             20, 299 ,
                             false, false)

Total query runtime: 11 ms.
31 rows retrieved.

```

La funcion *shortest\_path* toma como variables el id del vertice origen (20), y el vertice destino (299).

**5.6. Modelos ?**

**5.7. Diagrama de casos de uso ?**

# Bibliografía

- [1] Sam Ruby, Dave Thomas, David Heinemeier Hansson, *Agile Web Development with Rails, Fourth Edition*
- [2] <http://trends.builtwith.com/topsites/Ruby-on-Rails>
- [3] [tumblr.yasulab.jp/post/10271634919/5-question-interview-with-twitter-developer-alex-payne](http://tumblr.yasulab.jp/post/10271634919/5-question-interview-with-twitter-developer-alex-payne)
- [4] [http://www.artima.com/scalazine/articles/twitter\\_on\\_scala.html](http://www.artima.com/scalazine/articles/twitter_on_scala.html)
- [5] <http://oreilly.com/web2/archive/what-is-web-20.html>
- [6] <http://radar.oreilly.com/2006/12/web-20-compact-definition-tryi.html>
- [7] <http://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm>
- [8] <http://st-www.cs.illinois.edu/users/smarch/st-docs/mvc.html>
- [9] <http://msdn.microsoft.com/en-us/architecture/bb906060.aspx>
- [10] <http://www.json.org/json-es.html>
- [11] [http://kartoweb.itc.nl/geometrics/Coordinate %20systems/coordsys.html](http://kartoweb.itc.nl/geometrics/Coordinate%20systems/coordsys.html)
- [12] Introduction to Spatial Coordinate Systems: Flat Maps for a Round Planet,  
[http://msdn.microsoft.com/en-us/library/cc749633\(v=sql.100\).aspx](http://msdn.microsoft.com/en-us/library/cc749633(v=sql.100).aspx)
- [13] <http://publib.boulder.ibm.com/infocenter/db2luw/v8/index.jsp?topic=/com.ibm.db2.udb.doc/opt/csb3022b.htm>
- [14] <http://msdn.microsoft.com/es-es/library/bb964707.aspx>
- [15] Introducción a la Teoría De Grafos, Alfredo Caicedo Barrero,  
ISBN: 978-958-99325-7-5