

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

Edmundo Figueroa Herbas

18 de octubre de 2012

# Capítulo 1

## Ruby on Rails y patrones Web 2.0

### 1.1. 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 logro 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.

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

---

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

<sup>2</sup>MVC

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 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

---

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

<sup>4</sup>Test Driven Development

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

## 1.2. Patrones de diseño de la Web2.0

Que es la Web2.0 ?, primeramente se debe explicar que este término fue acuñado por 1999 para describir paginas web que usaban tecnologias 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]

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

### 1.2.1. 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.

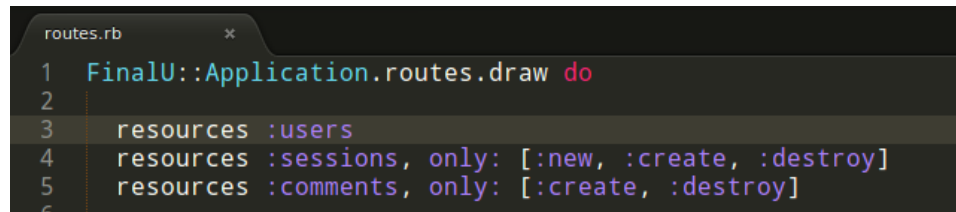
**GET** es la operación HTTP más común, es usado para leer datos en este caso paginas, se puede leer como “get a page”, **POST** es la operación que se usa cuando se ejecuta un formulario, en la convención de Rails **POST** se usa para crear objetos o recursos, PUT se usa para actualizar objetos ,DELETE se usa 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 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 usuarios es *restful*, tal como se muestra en la figura 1.1

La imagen 1.1 muestra como se declara a **users** con todas las acciones restful los cuales se listan en el cuadro 1.2.1, Rails también permite declarar solamente algunas acciones restful, como se ve el recurso **sessions** solamente tiene las acciones de **new**, **create** y **destroy**.

---

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



```
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 1.1: config/routes.rb

HTTP request	URL	ACTION
GET	/users	index
GET	/users/1	show
GET	/users/new	new
POST	/users	create
GET	/users/1/edit	edit
PUT	/users/1	update
DELETE	/users/1	destroy

Cuadro 1.1: las posibles rutas que se generan

Por ejemplo, si generamos 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 servidor procesa la información y ejecuta la acción **update** del controlador `usuarios` actualizando la información del usuario “1”. Estas acciones no son más que métodos dentro del `user_controller.rb` el cual es parte del controlador de la arquitectura MVC.

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 mas limpia y ordenada situaciones que son claves para el mantenimiento y la extensibilidad del sistema.

### 1.2.2. 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 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. Cada acción o concepto se desarrolla en un lugar determinado, los objetivos que se pueden apreciar al usar esta arquitectura es la facilidad para escribir código y de mantenerlo.

Se puede apreciar el comportamiento de este patrón, en la figura 1.2, al visitar la ruta que muestra la información de un usuario “`localhost/users/1`” desde el navegador, este genera una llamada GET y Rails lo direcciona a la acción `show` del **controlador** `users`, la acción `show` se encarga de llamar al **modelo** “User”, el controlador extrae el id del usuario de la llamada<sup>6</sup> **GET** `/users/1`, permitiendo al **modelo** que es el encargado de manejar la lógica del negocio, por ejemplo validando o modificando los datos que se guardan o se extraen de una base de datos, el controlador almacena la

---

<sup>6</sup>request

información devuelta por el modelo en la variable `@user`, que es pasada a la **vista** `show.html.erb` el cual se encarga de generar el template HTML y devuelve al navegador la página solicitada.

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 crear archivos de configuración, todo este trabajo es parte de la filosofía de Rails “convención sobre configuración”.

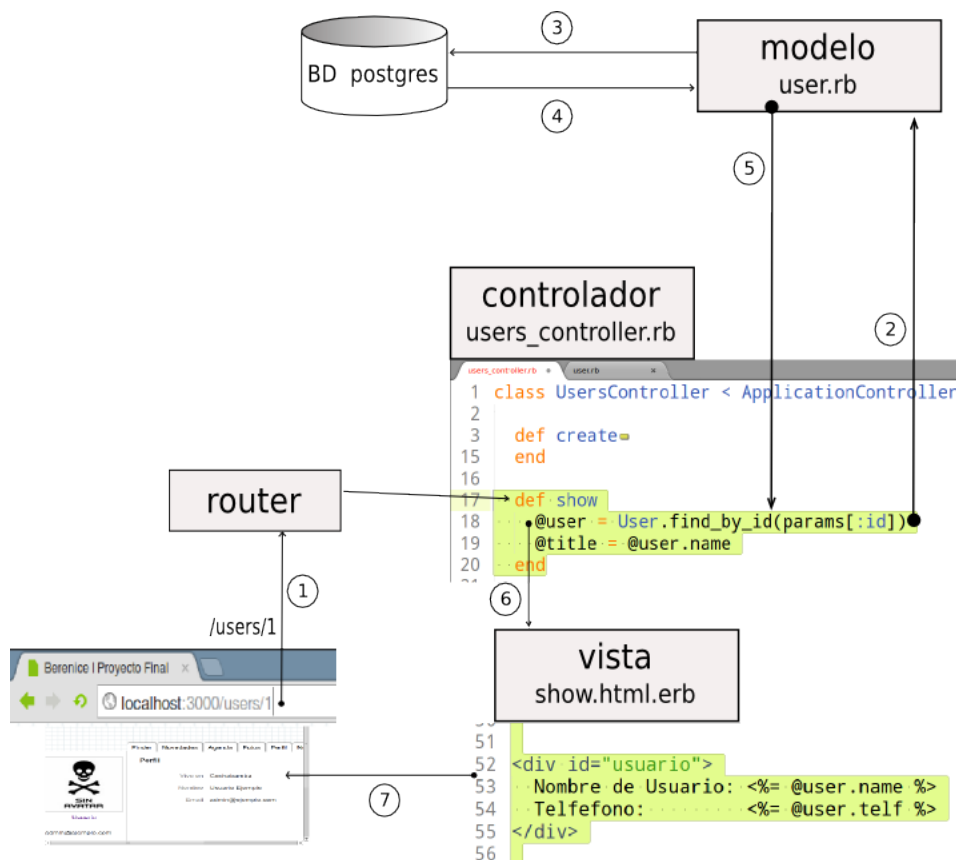


Figura 1.2: Un diagrama detallado del patrón MVC en Rails



### 1.2.3. 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>7</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 importante fue el de [housingmaps.com](http://housingmaps.com) que convina 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 señala 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**, sobre los cuales se puede obtener un producto con valor agregado, los datos están disponibles para que sean usados, pero obtener y almacenar estos “datos” y dependiendo del tipo de datos, 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.

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 cuales generalmente cobran por el uso de los mapas, se puede apreciar que

---

<sup>7</sup>Application Programming Interface, es el conjunto de funciones y procedimientos que se ofrece para ser utilizado por otro software

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.

Para usar el API de Google Maps solo es necesario importar la librería declarandola en la cabecera del documento HTML, para visualizar el mapa es necesario declarar un elemento único (se recomienda un div con id de preferencia “map” o similares) dentro del DOM de la página y se maneja mediante el lenguaje de programación del lado del cliente: javascript.

```
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 1.3: 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 1.4: 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 1.5: Se captura el div map del documento y se inicializa el mapa mediante el lenguaje javascript

# 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>