El maldito libro de los descarrilados.

Revisión

3

25/Oct/2010

Ruby on Rails para programadores que tienen cosas mas importantes que hacer.
Cubre la versión 3.0 de Ruby on Rails.

por R. Herrera

Tabla de contenido

La humanidad es inherentemente amante de la rutina y de la ilusión de control de ahí la resistencia al cambio.

Agradecimientos:

Principalmente al lector por volverme escritor, secundariamente al descarrilado proponente que enriquece este texto día a día, y consecuentemente a mi familia por aguantarme en mis deshoras.

Prólogo:

Un descarrilado en el contexto de este ensayo es un programador intentado entrar al mundo Rails y habiendo fracasado miserablemente con toda esa información en la web que al final no lo llevó a ninguna parte decide que este texto le puede ayudar, a su servidor le sucedió y decidió tomar sus apuntes y armar “el maldito libro que nunca encontró”.

El presente ensayo es solo un intento de iniciar al no iniciado, y tiene como requisitos: Conocimientos técnicos básicos de GNU/BSD/UNIX, HTML, Javascript, SQL y algún lenguaje de 3era. generación orientado a objetos, programación por objetos, análisis de sistemas, diseño de base de datos, comunicaciones TCP/IP y Web 2.0, todo lo demás, aunque no cubriremos nada en profundidad, se verá en las siguientes páginas.

Lamentablemente al lector solo puedo llevarlo de la mano hasta la puerta, pero de ahí en adelante entrará solo, y deberá vencer su propia resistencia al cambio y seguir investigando y aprendiendo ya que Rails está en sus inicios, es imperfecto, es inmaduro y esta lleno de áreas para mejorar, y de hecho lo hace, este mismo ensayo se encuentra en su 5ta. iteración, ya que muchísimas de las acciones que se realizaban en las versiones anteriores ya no son necesarios y han dado paso a otras, en muchos casos, mas simples, pero algunas otras, mas crípticas.

Las razones por las que el lector debe leer este texto son las siguientes: Inducción a Rails, la barrera del idioma, desactualización de otros textos, simplicidad de ejemplos o consulta para realizar tareas comunes.

Las razones por las que el lector NO debe leer este texto son la siguientes: Desconocimiento de alguno de los requisitos, profundizar en algún tema, no tener donde practicar lo aprendido, aprobar un curso.

Todo evoluciona y deberemos, el lector y su servidor, cambiar, sino seguiremos siendo... descarrilados.

Al final del texto se han colocado como capítulos propuestos aquellos temas que los descarrilados me han hecho llegar a través del correo electrónico, agradezco sinceramente las muestras de simpatía y las críticas que son siempre recibidas como constructivas, en el entendido de que estaré cubriendo esos temas en la medida de mis posibilidades, ya sea embebiéndolos en los capítulos ya existentes o escribiendo nuevos. Las actualizaciones estarán en yottabi.com/mld.pdf, y raul.herrera@yottabi.com asunto MLD para las quejas, críticas, comentarios y aportaciones.

Capítulo -1: Sobre logicial privativo

No lea este capítulo, es mas bien una catarsis

Comprendo en toda su cabalidad el impacto que ha causado microsoft y windows al mundo de la tecnología de la información, pasando por el de las comunicaciones y el del entretenimiento, entiendo completamente la inmensurable cantidad de personas que lleva el pan a la mesa de sus hogares día a día gracias a microsoft y también estoy de acuerdo como empresario que el mundo microsoft es una pequeña mina de oro.

Lo que no comprendo es: ¿¡Por que lo permitimos!? ... aparentemente consideramos correcto el mundo de la pantalla azul y el de los viruses, es normal que nuestras computadoras inicien en 10 minutos y consuman 475 watts en vez de 85 por que los discos duros no descansan, también se acepta el formateo como procedimiento normal de recuperación y no es para nada raro que al instalar un dispositivo nuevo la computadora se congele de siempre en siempre y mucho menos raro que cuando sale un nuevo windows nuestro equipo automáticamente es incompatible.

Estamos tan acostumbrados a este sistema no-operativo impuesto que la mayoría no se da cuenta que viene embebido, quiéralo o no tiene que pagarlo, no importa si lo va a usar o no, no importa si va a instalar un BSD o un GNU, usted tiene que pasar saludando al rey ofreciendo lealtad y obediencia con su cuota de US$39.99 a US$699.99 por lo bajo, dependiendo el grado de amor y lujuria que usted esté o no dispuesto a dar, al final eso no importa.

Los técnicos deberían ser millonarios, no conozco a ninguno que tenga un momento de ocio entre formatear y buscar drivers, los programadores deberían ganar la medalla olímpica por terminar el circuito “estar certificados y volverse obsoletos” en 45 segundos, los rockstars de los helpdesk deberían ser poseedores del trofeo a la paciencia por tener que explicar como la empresa mas rica del planeta puso “Formatear Disco” a la par de “Expulsar con seguridad”, los sysadmin deberían ganar el Valium de oro por incrementar las ventas de Diazepán ya que solo así pueden conciliar el sueño y al usuario vil y mortal, la invitación a sembrar un árbol en la calle de filantropía por regalar nuestro dinero ganado con gran esfuerzo, sin obtener nada real a cambio.

La tan popular e inmisericorde interface gráfica es tan inútil para los técnicos y desarrolladores que se ven forzados a repetir una y otra vez las tareas comunes por que simplemente no hay como guardar historia y repetirla, y mucho menos una bitácora para auditoría, y por lo mismo el soporte ssh/telnet es muy limitado o nulo, en realidad hay que utilizar un delomelanicon para invocar algo a la LogMeIn para tener la ilusión de control, y por el amor de Dios no se le vaya a ocurrir darle “Expulsar” a un USB remotamente, si no le dio formatear por error, habrá que reiniciar toda la máquina para montarlo otra vez.

Es alarmante la cantidad de empresas que en búsqueda de su minita de oro “hacen leña del árbol caído”, hay 230 empresas fabricantes de antivirus. ¿A quien le interesará que hayan viruses ... realmente?. Hay más de 2200 empresas que ofrecen entrenamiento para office 2007 al clamor de “Si trabaja con office 2003, necesita actualizarse”, hay en Latinoamérica 28 empresas que otorgan certificaciones para desarrolladores .net y un muy conveniente plan de 6 pagos, curiosamente hay una nueva área de certificación cada 6 meses, curiosamente solo hay 1 latinoamericano con la fatídica certificación Certified Master Directory, un dominicano que es algo así como un diccionario ambulante.

Según el propio microsoft el 88% de los usuarios que pagaron windows vista y office 2007 usan windows xp y office 2003, y el 50% de los programadores que asistieron a los cursos de certificación .net continúan utilizando visual basic 6. La media de “actualización” de vista a 7 es de US$119.99, el usuario promedio a través de 10 años de uso de productos microsoft y sus consecuencias gasta US$5430 por década y una Pyme típica gringa gasta en productos microsoft y su consecuente mantenimiento un promedio de US$985.000, por década.

Un sistema operativo ecológicamente consciente deberá proveer maneras de evitar la impresión de documentos, bajar consumo eléctrico, disminuir el impacto auditivo y visual, maximizar su eficacia para minimizar el tiempo usuario/máquina y prolongar la vida útil del equipo cuanto sea posible ... me he quedado sin palabras.

Si en Latinoamérica hubiera una iniciativa real, venida de nuestros asesores gubernamentales de IT, de no usar logicial privativo nos ahorraríamos unos cuantos millones de dólares al año que actualmente le cedemos a empresas como microsoft, eset u oracle y tendríamos un poquito de dinero adicional para fomentar la inversión extranjera, disminuir la mortalidad infantil, mejorar la educación, combatir la delincuencia y en general ayudar a salir del condenado “tercer mundismo” que al final no se que significa sino es ser pobres babosos.

Por lo anterior, este ensayo no fomenta el uso de logicial privativo en ninguna de sus formas.

Por cierto ... Rails es libre y hace el intento de correr en windows...

La fuente de las estadísticas ha sido una compilación basada en búsquedas a través de www.google.com, www.bing.com, office.microsoft.com, microsoft.com/presspass, eset.com, oracle.com, http://marketshare.hitslink.com, http://news.cnet.com, y por supuesto wiki.com

Capítulo 0: Introducción

Rails es un marco de trabajo basado en Ruby para elaborar aplicaciones web, su nombre oficial es Ruby on Rails, se apega al patrón de diseño MVC, genera aplicaciones para la web 2.0, es agnóstico a la base de datos y reclama que desarrollar en él es 10 veces mas rápido que las herramientas tradicionales. Es tan radical que en muchos casos hay que tomar todo lo que el programador sabe y tirarlo a la basura, es por eso que su penetración en el medio empresarial ha sido lenta al igual que su popularidad, sin embargo otros marcos de trabajo toman a Rails de paradigma y lo siguen a buen paso aunque guardando la distancia, como es el caso de Django para Python o Symfony para PHP5.

Y como todo producto elitista esta basado en filosofías elitistas, es decir que dentro de la utopia de la que busca ser parte Rails existen lineas de pensamiento o principios que de una u otra forma intentan hacer más productivo y feliz al programador, he aquí una recopilación y actualización realizada por el autor:

No te repitas, nunca escribas en dos lugares el mismo cálculo, o el mismo proceso o la misma función, si lo haces entonces nunca estarás seguro de donde viene tu problema, además ... duplicar esfuerzos no es inteligente, esta basado en el principio del origen único de la verdad, el reduccionismo metodológico y KISS.

Convención sobre configuración , la aplicación ya deberá funcionar sin interacción con el usuario, por que se han suplido las necesidades iniciales desde el mismo desarrollo, el usuario deberá poder cambiar estos valores para que se adapten a su necesidad en particular, en el caso del desarrollo provee las siguientes ventajas:

- Permitir a los desarrolladores nuevos aprender un sistema rápidamente.
- Promueve la uniformidad.
- Promueve el dinamismo.

Sin embargo ofrece también algunas desventajas:

- Se requiere familiarizarse con los defaults.
- Aumenta el peso de la aplicación.
- Puede ser difícil modificar un valor de fábrica.

Desarrollo Ágil, Reingeniería constante de tus acciones y tus ideas, es un método reactivo que provee soluciones al momento del problema, sus bases son las siguientes:

- **Las personas y su interacción** deben estar sobre los *procesos y las herramientas*.
- **La aplicación funcionará** y no deberá ser necesaria una *documentación detallada o abundante*.
- **La colaboración del cliente** por sobre la *planificación negociada*.
- **La necesidad** siempre estará sobre la *planificación*.

En el desarrollo ágil los elementos de la **izquierda** son mas importantes que el valor de los elementos de la derecha esto es opuesto al desarrollo tradicional, en busca de:

- La satisfacción del cliente a través de una veloz y continua entrega de aplicaciones útiles.
- Aplicaciones útiles entregadas frecuentemente (semanas en vez de meses).
- La utilidad de las aplicaciones es la medida del progreso del proyecto.
- Todos los cambios son bienvenidos aún aquellos de último momento.
- Cooperación y acercamiento diario entre la gente del negocio y la de desarrollo.

- La conversación cara a cara es la mejor forma de comunicación.
- Los proyectos están rodeados de gente motivada en la que se puede confiar.
- Atención continua a la excelencia técnica y un buen diseño.
- Simplicidad.
- Equipos de organización natural.
- Adaptación regular a circunstancias cambiantes.

Mejores Prácticas, codifica y luego recodifica para que quede legible para ti mismo y tu grupo de trabajo, dentro de 8 semanas no recordarás que hiciste y mucho menos como, comenta y documenta todo lo que puedas y apégate a los standards, tu aplicación debe contener exclusivamente el código que se ejecutará, no te conviene tener focos de confusión, maneja el error de forma elegante y minimiza la repercusión, trabaja simple, por que lo simple es bello por que es simple.

Versionado, utiliza alguna herramienta para manejar tus versiones, nunca sabrás cuando deberás dar algunos pasos hacia atrás, nunca sabrás que porciones de código te seguirán sirviendo o cuales habrá que reemplazar, y por sobre todo nunca sabrás cuando el siniestro acaecerá.

Ensayo, la aplicación fallará, simplemente no sabes cuando, por lo tanto deberás ensayarla en multitud de situaciones, hazlo hasta que la ilusión de control te reconforte, y cuando el momento llegue procura que el código maneje elegantemente la falla y minimice las repercusiones. En lo posible automatiza las pruebas.

Compartir, informa de tus actividades al grupo de trabajo, y procura que ellos también lo hagan, mantengan una clara y eficiente comunicación en función de la planificación o de la necesidad, procúrate una bitácora interactiva para asignar, comentar y validar las actividades, toma en cuenta las holguras y ajusta diariamente el itinerario y hazlo del conocimiento de todos.

Agnosticismo, sé independiente, debes ser capaz de trabajar en cualquier escenario, utiliza solamente herramientas o bibliotecas standard, no debes estar atado a sistemas operativos, editores, bases de datos, equipo o modas, ten claro tu objetivo y cúmplelo.

Conciencia, el mejor producto no necesariamente es el más popular, investiga, innova, experimenta y al final cambia, habrán periodos de cambio constante y otros de una semi eterna estaticidad, piensa en el futuro personal y el de tu empresa, toma en cuenta los impactos económico, fiscal, empresarial, social, ambiental y psicológico de tus decisiones, el mundo cambia, debes cambiar con él.

Honestidad, rodéate de motivadores y elimina los distractores, sé honesto contigo mismo y reconoce cuando estas enfocado y cuando no, haz lo que haces por que lo amas, no por rutina, en este negocio no es posible.

¡Atención! Tenga muy en cuenta que al copiar y pegar los ejemplos de este texto, es probable que no se transfieran correctamente los tabuladores, retornos de carro y líneas largas, causando errores muy ocultos que suelen ser indescifrables al usuario inexperto y por ende se puede perder la continuidad y la paciencia, como este texto propone una actividad didáctica mi recomendación es que el lector los digite.

Capitulo 1: Instalación

Antes de decir cualquier cosa, algunos OS ofrecen a través de comandos de instalación de paquetes nativos, la instalación “Linuxera” de Rails, esto puede ser bueno en un principio, pero los repositorios no se actualizan a la velocidad que lo hacen las gemas, de tal suerte que es posible que el lector se quede anclado a una versión en particular sin poderse actualizar, le suplico lo intente con los métodos que detallo a continuación y juzgue por si mismo cual es el que mejor le conviene.

Nuestra plataforma de ensayos será un Linux Debian, cualquiera de sus múltiples sabores y colores nos es útil, utilizaremos la utilería **apt-get** pero bien podría ser **aptitude** o cualquier otra en la que el lector se sintiera cómodo.

Método a la Linux

Este método lo daré sin mucho miramiento, no lo recomiendo y por lo mismo no lo detallo, en general funciona, a veces no lo hace, y no estoy muy interesado en estudiar el porque.

Para instalar:

```
usuario@host:~$ sudo apt-get install rails
```

Para desinstalar:

```
usuario@host:~$ sudo apt-get remove rails
```

Método Independiente

Este será el método de la salida cobarde, es decir ¨quiero que funcione pero no me interesa como¨, no recomiendo este método, está aquí debido a la petición de los descarrilados que necesitan donde ensayar sin tanta dificultad, así que tiene algún valor didáctico, pero hasta ahí, de tal suerte que será un listado simplista y corto.

Turnkey Linux, http://www.turnkeylinux.org/rails (Agradable, simple, Appliance)
Instant Rails, http://rubyforge.org/frs/?group_id=904 (Lento, Obsoleto, Window$)
RubyStack, http://bitnami.org/stack/rubystack (Actualizado, MultiOS, Standard)
Lomotive, http://sourceforge.net/projects/locomotive/ (Obsoleto, MacOS, Bellísimo)

Dentro de los males el menor, prueben RubyStack, es impresionante.

Método Oficial

Si va a continuar con lo que resto del capítulo asegurése que Rails esta desinstalado con el comando **sudo apt-get remove rails**.

Para efectuar una actividad de desarrollo con Rails son necesarios 4 requisitos mínimos, Ruby, Gems, Rake y Rails, así que como primer paso, obviamente, es verificar si los tenemos instalados y si no lo están, instalarlos.

Es importante ubicar el OS donde trabajaremos, así que se vuelve menester correr el comando **uname** para saberlo, en función del OS el comando de instalación de paquetes podría variar, entiendase un paquete como un programa que se baja de la red que es *ad-hoc* a nuestra configuración y versión de kernel, de esa forma se garantiza la compatibilidad y el desempeño.

```
usuario@host:~$ uname -a
```

```
Linux usuario@host 2.6.35-22-generic #35-Debian SMP Sat 16 20:36:48 UTC 2010 i686 GNU-Linux
```

Por supuesto no debe ser idéntico simplemente similar, en algunos otros OS podrían aparecer mas datos o menos, eso no es lo importante, sino la ubicación de la versión del kernel y la base de la distribución que en este caso es Debian. Si obtuvieramos algo como esto:

```
Darwin MyHost.local 10.4.0 Darwin Kernel Version 10.4.0: Fri Apr 23 18:28:53 PDT 2010;
```

Entonces tendríamos un Darwin 10.4.0 y la utilería de instalación de paquetes sería **ports**, por supuesto siempre podemos ir a la página de cada uno de los requisitos y seguir las instrucción precisas que ahí nos dan para la instalación en nuestro OS en particular.

Ruby

La página oficial de Ruby es http://www.ruby-lang.org/es/, este lenguaje por si mismo es quizás, en mi humilde opinión, lo mejor que le pudo pasar a la programación por objetos y existe una enorme cantidad de información acerca de él, y es por mucho una de mis mas frecuentes recomendaciones de lectura técnica. Para verificar su instalación y versión corremos:

```
usuario@host:~$ ruby -v
```

```
ruby 1.8.7 (2010-06-223 patchlevel 299) [i686-Linux]
```

Si no ve algo como esto, entonces es menester que instale Ruby, en muchos casos el mismo OS le sugerirá el comando, hágale caso, pero tambien deberá instalar las bibliotecas de desarrollo para que Gem pueda efectuar los ajustes en los manejadores de base de datos y cualquier otra cosa que se presente. Por favor digite

```
usuario@host:~$ sudo apt-get install ruby
usuario@host:~$ sudo apt-get install ruby1.8-dev
```

Las lineas son comandos distintos, por favor presione <Enter> después de cada una.

El comando **sudo** permite la ejecución del comando que le sigue con privilegios de superusuario así que le pedirá su clave, si por alguna razón le da algún tipo de error tendrá que pedirle al administrador que le permita instalar paquetes en su máquina o que lo añada al grupo de usuarios Sudoers para poder ejecutar el comando **sudo**. Y por supuesto deberá estar conectado a la red para poder bajar el paquete correspondiente. Una vez el comando anterior fue exitoso reintente verificar la versión de Ruby.

Es muy importante resaltar que apt-get al igual que la mayoría de comandos en los nuevos ¨Unixes¨ poseen la característica de escritura rápida predictiva, es decir que bastará con tipear solo las primeras letras y presionar <Tabulador> para que el OS intente descifrar cual es el siguiente comando y si no puede, sugerir las alternativas, incluyendo los paquetes a instalar, es decir que solo basta con colocar ¨ru¨y presionar tabulador para que el comando sea sugerido con textos como ruby, ruby1.8, runit, rubygems, rumor, rubrica, solo por mencionar algunos.

Rake -- Ruby Make

Rake es un programa armador exclusivamente para Ruby, es decir que toma código de aquí y de allá para armar una aplicación funcional, para nuestro propósitos será útil para realizar las migraciones, las pruebas y la documentación pero tiene muchos usos más, se caracteriza por ser simple y de fácil entendimiento como todo lo que Ruby representa. la página oficial es http://rake.rubyforge.org/ para verificar su instalación y versión corremos.

```
usuario@host:~$ rake --version
```

```
rake, version 0.8.7
```

Si no está instalado entonces corra:

```
usuario@host:~$ sudo apt-get install rake
```

y por favor vuelva a verificar la instalación y versión de Rake, como aclaración, he notado que en algunas instalaciones en el momento de instalar Rails también se instala Rake, pero no quiero que el lector se la juegue, de esta forma funcionará perfectamente.

Gems o Ruby Gems

Los lenguajes de alto nivel se caracterizan hoy por hoy por tener repositorios, es decir que las aplicaciones que se realizan con ellos ,que son de dominio público, son almacenadas en contenedores de donde cualquiera los puede bajar, esto es una cosa maravillosa por que al igual que los paquetes verifican los prerequisitos para garantizar estabilidad y desempeño, pero también cada lenguaje tiene su propio manejador de paquetes, en el caso de Ruby son las Gemas o Gems que son aplicaciones armadas y

listas para producción que simplemente se corren, Rails es una gema y por lo tanto deberemos instalar el instalador, la página oficial es http://rubygems.org

Primero verificamos la instalación y versión de Ruby Gems, sin embargo Ruby Gems es muy distinto a lo que hemos instalado antes, tiene un frente y una aplicación de soporte, **rubygems** es la aplicación, pero **gems** es el frente, así que verificamos la versión e instalación con:

```
usuario@host:~$ gem -v
```

```
1.3.7
```

Para cuestiones exclusivamente de este texto la versión 1.3.7 es la mínima aceptable, así que cualquier versión anterior deberá ser actualizada, pero primero veamos el caso que simplemente no tengamos instalado Ruby Gems, entonces lo instalamos con:

```
usuario@host:~$ sudo apt-get install rubygems
```

Verificamos la versión una vez mas, si no es satisfactoria entonces actualizaremos las gemas con:

```
usuario@host:~$ sudo gem update --system
```

Verificamos la versión una vez mas, si hubo algún tipo de error entonces procedemos a la instalación manual basta con bajar un archivo de http://rubygems.org y ejecutarlo como ahí se indica.

En este momento podemos verificar si todo esta funcionando correctamente al invocar el servidor de gemas

```
usuario@host:~$ gem server
```

```
Server started at http://[::ffff:0.0.0.0]:8808
Server started at http://0.0.0.0:8808
```

Ahora podemos ir a cualquier navegador de internet e ingresar la dirección en la casilla del URL http://localhost:8808 ahí podremos tener un listado de las gemas instaladas y siempre podemos revisar que nuestras gemas esten correctas aqui mismo.

Rails

Rails es el tema de este texto, así que no tiene mucho sentido definirlo una vez mas, la página oficial es http://rubyonrails.org/, si llegó aquí es porque eligió el camino largo y difícil y asumo que no tiene instalada la versión de Rails por repositorios de OS, , verifiquelo de la siguiente forma:

```
usuario@host:~$ rails -v
```

```
Rails 2.3.8
```

Si no ve el mensaje anterior sino algún tipo de error de no instalación, entonces estamos bien, si ve alguna versión de Rails entonces vuelva al principio de este capítulo para desinstalarlo.

Ya que estamos claros y dispuestos a instalar, ahora todo es cuesta abajo, instalaremos la gema Rails y un comparsa llamado Bundler

```
usuario@host:~$ sudo gem install rails
usuario@host:~$ sudo gem install bundler
```

Después de unos minutos de mensajes de instalación, crear documentación y configuración tendrá la versión mas actualizada de Rails, verifique la versión con:

```
usuario@host:~$ rails -v
```

```
Rails 3.0.1
```

Si parece como si Rails no estuviera instalado, no se preocupe, en algunos Debian, léase Ubuntu, no se creará un link automático, de tal suerte que será necesario correr el comando siguiente y por favor utilice la escritura rápida predictiva para asegurarse que los directorios son los correctos:

```
usuario@host:~$ sudo ln -s /var/lib/gems/1.8/bin/rails /usr/bin/rails
usuario@host:~$ sudo ln -s /var/lib/gems/1.8/bin/bundle /usr/bin/bundle
```

Con lo anterior simplemente creamos encadenamientos, digamos atajos, hacia donde están realmente instaladas las gemas para que sea facil invocarlas, ahora verifique la versión, si todo es exitoso entonces estamos listos. Bundler es un empacador de aplicaciones toma las gemas del sistema las empaca y las embebe dentro de la aplicación así que si queremos correr la aplicación en otra computadora no habrá necesidad de instalarlas manualmente.

Aunque Rails esta listo para funcionar, aun nos falta una cosa mas, la base de datos, es decir que Rails puede funcionar sin base de datos pero no es lo común y por lo tanto debemos tener al menos una instalada, para motivos de nuestro texto usaremos SQLite 3, pero también instalaremos MySQL a manera de ilustración.

Como ahora estaremos trabajando con la compilación de paquetes a multiples niveles siempre es conveniente tener ya instalado los compiladores esenciales, esto lo hacemos con:

```
usuario@host:~$ sudo apt-get install build-essential
```

SQLite 3

SQLite 3 es una excelente base de datos relacional, es muy rápida, muy confiable, tiene muy poco peso, y de alguna forma es ubicua, ya que la instalación completa ocupa menos de 300k y es común que se utilice en handhelds, teléfonos celulares, sistemas empotrados y por supuesto computadoras, pero quizás dentro de sus mayores atractivos esta su licencia que dice algo así:

HARÁS EL BIEN Y NO EL MAL.
ENCONTRARÁS PERDÓN PARA TI Y PARA LOS OTROS.
COMPARTIRÁS LIBREMENTE, NUNCA TOMANDO MÁS DE LO HAS DADO.

Rails utiliza esta base de datos como default, pudiendola cambiar en el momento de la creación o en cualquier momento en la etapa de desarrollo, la página oficial es http://www.sqlite.org/, pero como nada es perfecto tiene 2 defectos que no la hacen viable para producción a nivel teórico ... no es segura y tampoco maneja muy bien la concurrencia, es decir que cualquiera puede ver lo que en la base de datos hay y no funciona muy bien cuando hay muchos usuarios, sin embargo esto en la práctica no es ninguna limitante por que al fin y al cabo la BD está en el servidor que es inherentemente seguro y con lo de la concurrencia siempre que no sean 1000 búsquedas por segundo todo ira bien.

Verificamos instalación y versión con:

```
usuario@host:~$ sqlite3 -version
```

```
3.7.2
```

Si no ve algo como lo anterior entonces deberá instalarla con el siguiente comando y deberá volver a verificar la instalación y la versión, pero como estamos destinando SQLite para trabajar con Rails de una vez instalamos la biblioteca de conexión y la de desarrollo para que la gema pueda encontrar donde conectarse.

```
usuario@host:~$ sudo apt-get install sqlite3
usuario@host:~$ sudo apt-get install libsqlite3-ruby
usuario@host:~$ sudo apt-get install libsqlite3-dev
```

Y por último la gema para que Rails pueda acceder a SQLite 3

```
usuario@host:~$ sudo gem install sqlite3-ruby
```

MySQL

MySQL es por llamarle de alguna forma el estándar de las bases de datos para web, sin embargo esa definición se queda muy corta por que en realidad MySQL se utiliza en cada ámbito donde cualquier otra base de datos ha también estado, es muy confiable, muy rápida y muy segura, esta diseñada para producción a nivel empresarial y tiene una licencia muy económica para producción y gratuita para aprendizaje o para aplicaciones sin fines de lucro, la página oficial es http://www.mysql.com/. Cabe mencionar que recientemente la adquirió Oracle, eso significa que no es una cosa que hay pasar a la ligera.

Para verificar su instalación y versión corremos

```
usuario@host:~$ mysql --version
```

```
mysql  Ver 14.14 Ditrib 5.1.49, for debian-linux-gnu (i686) using readline 6.1
```

Si no vió algo como lo anterior por favor instalelo de la siguiente forma:

```
usuario@host:~$ sudo apt-get install mysql-server
```

Tome en cuenta que le pedirá en medio de la instalación la clave de superusuario (root) y la deberá ingresar 2 veces, como cualquier clave no la vaya a olvidar y contrario a las mejores prácticas apúntela en algún lugar visible, al fin y al cabo es solo para los ensayos. El comando anterior instalara tanto el servidor como el cliente, así que no es necesaria una instalación posterior para poder acceder a la base de datos.

Ahora instalamos las bibliotecas de soporte de Ruby y también las de desarrollo para que la gema sepa donde conectarse.

```
usuario@host:~$ sudo apt-get install libmysql-ruby
usuario@host:~$ sudo apt-get install libmysqld-dev
usuario@host:~$ sudo apt-get install libmysqlclient-dev
```

Una vez instalados todos los paquetes sin errores entonces podemos instalar la gema, sin embargo es posible que su versión de MySQL no corra con la gema mas actual, así que instalaremos ambas.

```
usuario@host:~$ sudo gem install mysql
usuario@host:~$ sudo gem install mysql2
```

Como MySql es una base de datos multiesquema entonces hay que crear un esquema para almacenar nuestros datos, lo hacemos de la siguiente forma, asumiendo siempre que la clave es 1234:

```
usuario@host:~$ mysql -u root -p
```

```
Enter password: 1234
Welcome to the Mysql monitor.  Commands end with ; or \g
Your MySQL connection id is 51.
Server version: 5.1.49-1debian11 (Debian)

Copyright (c) 2000, 2010, Oracle and/or its affiliates. All rights reserved.
This software comes with ABSOLUTELY NO WARRANTY. This is free software,
and you are welcome to modify and redistribute it under the GPL v2 license

Type 'help' or '\h' for help. Type '\c' to clear the current input statement.

mysql> create database misdatos;
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

Esencialmente todo se resume en **create database misdatos;** así nos aseguramos que la base de datos está levantada y activa. **misdatos** es ahora el nombre de la base de datos a conectarse y se deberá configurar en el **database.yml**, también se ingresará ahí mismo la clave, por default Rails 3 se configura con el manejador **mysql2** pero podría ser necesario cambiarlo a **mysql** por motivos de compatibilidad, para eso ya estamos preparados, lamentablemente el tema de MySQL se sale del ámbito de este texto así que ya no lo tocaré mas.

La prueba

Ya tenemos todo listo, ahora veamos si realmente esta cosa funciona como debe, crearemos un directorio de trabajo para poder alojar nuestras aplicaciones, no importa donde se coloque solo es necesario tener permisos de lectura-escritura.

```
usuario@host:~$ mkdir rails
usuario@host:~/rails$ cd rails
```

Ahora invocamos el generador de aplicaciones:

```
usuario@host:~/rails$ rails new prueba1
```

```
create
      create
      create  README
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      create  app
      (se omite el resto)
```

Si lo anterior fue exitoso después de unos segundos, veremos texto fluyendo por la pantalla que nos indica que algo se ha ejecutado, podrá ver muchos "create", en este momento ya tenemos media aplicación ya programada, para verificar:

```
$ cd prueba1
```

En este momento ya tenemos media aplicación armada, todo lo que haremos estará alojado en el directorio **~/rails/prueba1**, para motivos de trasiego todo el directorio podrá ser movido a otra posición, la aplicación no tiene ningún componente fuera de este directorio.

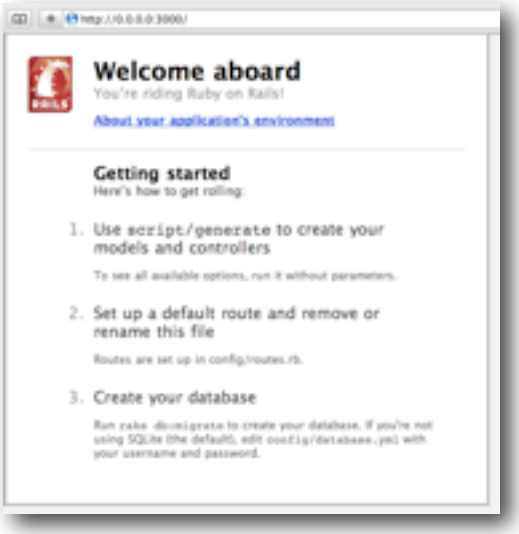
Ahora hay que darle fuego y ver que pasa.

```
usuario@host:~/rails/prueba1$ rails server
```

```
=> Booting WEBrick
=> Rails 3.0.1 application starting in development on http://0.0.0.0:3000
=> Call with -d to detach
=> Ctrl-C to shutdown server
```

Esto ejecutará el servidor en la maquina local publicando una aplicación en el IP 0.0.0.0 puerto 3000, es decir que en nuestro navegador introduciremos el URL, y deberemos ver una página similar a la ilustración.

http://0.0.0.0:3000



En muchos casos querremos publicar nuestra aplicación en la LAN para pruebas o demostraciones, el ip 0.0.0.0 es exclusivo de nuestra máquina, entonces utilizaremos el modificador -b

```
$ rails s -b 192.168.39.122 -p 3008
```

Donde 192.168.39.122 es el ip de nuestra maquina y 3008 el puerto, de esa forma los clientes de la LAN podrán acceder a nuestra aplicación con el URL:

http://192.168.39.122:3008

La ayuda se invoca con:

```
$ rails s -help
```

```
Usage: rails server [mongrel, thin, etc] [options]
    -p, --port=port                  Runs Rails on the specified port.
                                     Default: 3000
    -b, --binding=ip                 Binds Rails to the specified ip.
                                     Default: 0.0.0.0
    -c, --config=file                Use custom rackup configuration
file
    -d, --daemon                     Make server run as a Daemon.
    -u, --debugger                   Enable ruby-debugging for the
server.
    -e, --environment=name           Specifies the environment to run
this server under (test/development/production).
                                     Default: development
    -P, --pid=pid                    Specifies the PID file.
                                     Default: tmp/pids/server.pid

    -h, --help                       Show this help message.
```

Los ambientes de ejecución son Desarrollo, Prueba y Producción, siendo el de desarrollo el estándar, cada uno de ellos accede a una base de datos distinta y varía la modalidad de presentación de advertencias y errores.

Y para detenerlo presionamos <Control-C>

Capítulo 2: Apache 2

Es una realidad que muchos servidores web deben correr aplicaciones PHP, incluyendo los que correrán Rails, de ahí que resulta mas fácil hacer que Apache 2 corra Ruby a que Mongrel o Webrick corran phpmyadmin, añadido a que, cualquier hosting que se respete, manejará las aplicaciones Rails de forma implícita y no deberemos invocar explícitamente el servicio.

Iniciamos instalando Apache 2, esto es muy sencillo, deberemos correr:

```
$ sudo apt-get install apache2 libapache2-mod-fcgid libfcgi-ruby1.8
```

Inmediatamente después activaremos algunos módulos del Apache 2 y lo reiniciaremos:

```
$ sudo a2enmod ssl
$ sudo a2enmod rewrite
$ sudo a2enmod suexec
$ sudo a2enmod include
$ sudo service apache2 force-reload
```

Ahora modificaremos el archivo de sitios disponibles, en nuestro caso reemplazaremos el servicio en el puerto 80 y todo lo que eso implica, por favor saque una copia de respaldo, el archivo es */etc/apache2/sites-available/default*

Para el ejemplo nuestra aplicación estará alojada en el directorio /var/rails/prueba1, se espera que miap.misitio.com apunte hacia el servidor.

```
/etc/apache2/sites-available/default
<Virtualhost *:80>
   ServerName miap.misitio.com
   DocumentRoot /var/rails/prueba1/public/

   <Directory /var/rails/prueba1/public>
      Options ExecCGI FollowSymLinks
      AllowOverride all
      Order allow,deny
      Allow from all
   </Directory>
</Virtualhost>
```

Sin embargo aún no tenemos /var/rails, entonces lo deberemos crear, pero también habrá que darle permisos para que Apache 2 puedo acceder a él, de ahí que usaremos el usuario www-data y el grupo www-data que son propios de Apache 2, pero antes reiniciaremos el servidor para que tome los sitios disponibles, todo eso se hace con los siguientes comandos:

```
usuario@host:~$ sudo service apache2 restart
usuario@host:~$ sudo mkdir /var/rails
usuario@host:~$ sudo chown -R www-data:www-data /var/rails
usuario@host:~$ cd /var/rails
usuario@host:/var/rails$ sudo su -m www-data
www-data@host:/var/rails$ rails prueba1
```

Ahora creamos los archivos .htaccess en /var/rails/prueba1/public, y dispatch.fcgi con el cuidado de darle permisos 755 a este último.

```
/public/.htaccess
AddHandler fcgid-script .fcgi
Options +FollowSymLinks +ExecCGI

RewriteEngine On

RewriteRule ^$ index.html [QSA]
RewriteRule ^([^.]+)$ $1.html [QSA]
RewriteCond %{REQUEST_FILENAME} !-f
#RewriteRule ^(.*)$ dispatch.cgi [QSA,L]
RewriteRule ^(.*)$ dispatch.fcgi [QSA,L]

ErrorDocument 500 "<h2>Aplicación con problemas</
h2>Sufrimos de dificultades, intente mas tarde"
```

```
/public/dispatch.fcgi (con permisos 755)
#!/usr/bin/ruby
require File.dirname(__FILE__) + "/../
config/environment"
require 'fcgi_handler'
RailsFCGIHandler.process!
```

Para nuestro ejemplo deberemos retornar a nuestro usuario para dar los permisos ya que **www-data** no es un **sudoer**.

```
www-data@host:/var/rails$ exit
usuario@host:/var/rails$ sudo chmod 755 prueba1/public/dispatch.fcgi
```

Ahora podemos probar en nuestro navegador con un URL del tipo http://miap.misitio.com o algo como http://192.168.0.100 y deberemos ver la pantalla de bienvenida de Rails, de ahora en adelante deberemos trabajar exclusivamente con el usuario **www-data** para poder desarrollar y dar mantenimiento a nuestra aplicación, el ambiente de producción se establecerá en el archivo **./config/environment.rb** agregando ENV['RAILS_ENV'] ||= 'production' curiosamente no será necesario reiniciar nada, Rails hace esto automágicamente.

Capítulo 3: Análisis

Ahora crearemos nuestra aplicación de introducción, en este caso será un pequeño foro, para que estemos claros, un foro tiene usuarios que crean hilos y cada hilo tiene comentarios, por lo tanto al llevarlo a 3era. forma normal los datos nos quedan de la siguiente forma:

Descripción de los datos para la aplicación "Foro"

Hilos		Comentarios	
texto varchar(100)	Texto del encabezado del hilo.	usuario_id integer	Llave foránea a la tabla usuarios.
usuario_id integer	Llave foránea a la tabla usuario	hilo_id integer	Llave foránea a la tabla Hilos.
estado integer	Activo, Inactivo, Invisible, de forma que inactivo solo permite leer, pero no agregar comentarios.	texto varchar(400)	Texto del comentario.
		estado integer	Modificable o lectura solamente.
Usuarios			
nombre varchar (12)	nombre único del usuario, llave candidata		
clave varchar(20)	Clave encriptada del usuario (por favor omítase en este momento, nos servirá para ejemplo posteriormente)		

La convención sobre configuración difiere del análisis tradicional ya que al ser una herramienta de ultima generación no es necesario definir llaves primarias, ni fechas de creación o fechas de actualización, esos datos son automáticos, adicionalmente tome nota que los campos que terminen con _id se asumirán llaves foráneas a las tablas que tengan como prefijos

Añadido a lo anterior, si la entidad es Usuario, entonces la tabla se creará automáticamente con el nombre Usuarios, como es una pluralización en sajón, no necesariamente es la más adecuada, y por lo tanto habra que cambiar el archivo de locales para que se la correcta en castellano.

En esta fase, no espere ver DDL, tampoco espere ver relaciones o tablas, no habrá esquemas, ni invocaciones al cliente de la base de datos, Rails se encargará de tomar lo que sabemos y convertirlo en lo que queremos.

Capítulo 4: Elaboración

Aunque en la fase de desarrollo formal no se espera que se utilicen generadores, estos están aquí por una y solo una razón, “eliminar un montón de trabajo”, y por lo tanto una vez se comprende lo que en realidad hacen, son una excelente herramienta para iniciar, los siguientes comandos corren igualmente en ~/rails/prueba1

```
$ rails g scaffold hilo texto:text usuario_id:integer estado:integer

$ rails g scaffold comentario usuario_id:integer hilo_id:integer texto:text estado:integer

$ rails g scaffold usuario nombre:string
```

Como podrá darse cuenta no tienen un orden especifico, los usuarios son necesarios para los comentarios y los hilos, sin embargo fueron creados al último. Ahora la instrucción para asentar la migración, que es simplemente crear las tablas en la base de datos:

```
$ rake db:migrate
```

Si obtiene un mensaje del tipo **rake aborted! no such file to load -- sqlite3** por favor corra **sudo apt-get install libsqlite3-ruby1.8**

Y listo, tenemos una aplicación perfectamente funcional aunque aun dista un poco de ser útil.

Iniciamos el servidor.

```
$ rails s
```

Y coloquemos en el URL del visualizador

http://0.0.0.0:3000/usuarios

Podremos ver un ABCD de la tabla usuario, simplista pero al menos completo, en realidad tenemos 3 módulos funcionales, de ahora en adelante les llamaré **controladores**:

http://0.0.0.0:3000/usuarios
http://0.0.0.0:3000/comentarios
http://0.0.0.0:3000/hilos

Si cometemos un error en un scaffold basta con invocar el script destroy, es decir:

```
$ rails destroy scaffold usuario
```

Y tendremos una reversión muy buena, pero habrá que ir a la BD y administrarla ya que la tabla seguirá existiendo. Vea el capítulo 14 para invocar la consola de la base de datos.

Capítulo 5: MVC

El patrón de diseño MVC, significa que todo se distribuirá en Modelos, Vistas y Controladores, cada elemento tiene una muy bien definida función.

En los modelos está el código que interactúa con la base de datos, se encarga de las reglas del negocio, relaciones entre tablas, métodos y atributos virtuales entre otras cosas.

La vista es un generador de código visual aunque en Rails el único código visual que genera es HTML y Javascript, y consecuentemente AJAX. La vista puede acceder al modelo y presentar cualquier tipo de información pero no debe, en general, hacer cálculos.

El controlador se encarga de la interacción entre el modelo y la vista, en general recopila los datos, los procesa e invoca la siguiente vista, no debe tener reglas de negocio.

El flujo normal inicia en el controlador inicial, que extrae datos del modelo, y los presenta en una vista, una vez esta vista retorna la petición el controlador redirecciona esos datos al modelo donde son aceptados o no, el controlador actúa en función de esa aceptación o rechazo para volver a hacer otra petición al modelo y redireccionar el flujo a la vista correspondiente, este ciclo se repetirá indefinidamente.

El controlador inicial será el que aparece en el archivo de rutas, por el momento nos tendremos que conformar con saber que el controlador con el que interactuamos se extrae del URL, en la forma:

El controlador recibe en este caso como parámetros el Id y la Acción para que pueda saber como actuar y que vista presentar, dentro de sí el controlador tiene un método con el nombre de la acción y cada acción tiene relacionada una vista con el mismo nombre. El capítulo 12 habla de las rutas, y el 7 de los controladores y las vistas.

Capítulo 6: El esqueleto

Cuando se invoca el comando *rails*, este genera un conjunto de directorios y archivos, a esto se le denomina el esqueleto de la aplicación:

```
Gemfile
app
doc
script
Gemfile.lock
config
lib
test
README
config.ru
log
tmp
Rakefile
db
public
vendor
```

El archivo *README* contiene una muy buena descripción de lo que contienen y cual es su función, en general toda la acción estará en el directorio “app” ya que dentro de sí mismo están los controladores, modelos y vistas, para cada tabla se ha creado un controlador y un modelo, y un directorio con 4 vistas, cada una de ellas para los eventos *index, edit, new, show*. Se usará *usuario* como ejemplo:

- Vista *index*, presenta una matriz con el contenido de la tabla.
- Vista *edit*, presenta una forma para modificar los datos de la tupla.
- Vista *new*, presenta una forma para ingresar una tupla nueva.
- Vista *show*, presenta los datos de una sola tupla.

Como ejemplo, vea el directorio **./app/views/usuarios/**

El controlador contiene 7 acciones, *index, show, new, edit, create, update, y destroy.*

- Acción *index*, extrae todas las tuplas y las envía a la vista *index*
- Acción *show*, extrae una tupla y la envía a la vista *show*
- Acción *new*, genera una tupla en blanco y la envía a la vista *new*
- Acción *edit*, extrae una tupla y la envía a la vista *edit*
- Acción *create*, toma la respuesta de la vista *new*, graba, y presenta la vista *show*, si hay error invoca la vista *new* otra vez.
- Acción *update*, toma la respuesta de la vista *edit*, graba y presenta la vista show. si hay error invoca la vista *edit* otra vez.
- Acción *destroy*, toma la respuesta la vista que lo invoca, borra la tupla e invoca la vista *index*.

Como ejemplo vea el archivo **./app/controllers/usuarios_controller.rb**

Curiosamente el modelo no contiene nada útil, mas que la definición de él mismo, sin embargo será en el donde la magia se creará. Vea el archivo **./app/models/usuario.rb**

El directorio “config”, contiene 3 archivos importantes, el environment.rb, routes.rb y database.yml, como hemos hecho nuestro mejor esfuerzo por configurar lo menos posible estos archivos permanecerán en la incógnita por el momento, sin embargo, presentan los puntos de configuración de Rails, cada uno tendrá su propio capítulo mas adelante.

El directorio “db” contiene en nuestro caso la base de datos, es decir que en la configuración por defecto, el archivo development.sqlite3 **¡es nuestra base de datos!**, Sqlite3 es un excelente gestor de base de datos y tiene el particularidad que todo se guarda en un solo archivo, también contiene el directorio “migrate” que contiene los cambios realizados a nivel de DDL a la base de datos, también merece un capitulo aparte.

El directorio “public” es donde se almacenan los archivos expuestos al usuario final, es la raíz de nuestra aplicación y el archivo **index.html** es la página de bienvenidos de Ruby on Rails, también requerirá de un capitulo aparte.

Capítulo 7: La vista index

Al usar “usuario” como ejemplo **./app/views/usuarios/index.html.erb**, primero debemos revisar **./app/controllers/usuarios_controller.rb**, que es su controlador:

Lo que vemos aquí, en el método index, no es mas que una petición al modelo Usuario, para que exponga todas sus tuplas, estas se almacenan en @usuarios que es una variable de instancia, es decir que transcenderá hasta la vista, la cláusula de respuesta está en función si lo que pedimos es una página html, o un archivo XML.

Index es siempre el método por defecto, es decir que se puede hacer una llamada del tipo http://0.0.0.0:3000/usuarios en donde se asumirá que se busca la acción index, se ejecutará el método y presentará la vista index.html.erb. En contraparte, una llamada del tipo http://0.0.0.0:3000/usuarios.xml instruirá a la cláusula de respuesta que no queremos un HTML sino un XML y generará el archivo pertinente.

Aquí tenemos HTML simple, donde se insertan instrucciones en Ruby a través de <% %>. el = significa print, y la h filtrar HTML.

@usuarios es la variable de instancia que creamos en el método index, el método .each significa que tome de a uno por uno los usuarios, y lo nombre como usuario, el método nombre del objeto usuario se refiere al atributo nombre de la tupla usuario.

```
./app/views/usuarios/index.html.erb
<h1>Listing usuarios</h1>

<table>
  <tr>
    <th>Nombre</th>
    <th>Hilo</th>
    <th>Texto</th>
    <th>Estado</th>
  </tr>

<% @usuarios.each do |usuario| %>
  <tr>
    <td><%=h usuario.nombre %></td>
    <td><%=h usuario.hilo_id %></td>
    <td><%=h usuario.texto %></td>
    <td><%=h usuario.estado %></td>
    <td><%= link_to 'Show', usuario %></td>
    <td><%= link_to 'Edit', edit_usuario_path
(usuario) %></td>
    <td><%= link_to 'Destroy', usuario, :confirm =>
'Are you sure?', :method => :delete %></td>
  </tr>
<% end %>
</table>

<br />

<%= link_to 'New usuario', new_usuario_path %>
```

La instrucción link_to maneja rutas simplificadas: usuario a la acción show, edit_usuario_path(usuario) ira a la accion edit con el id del usuario y new_usuario_path irá a la accion new del controlador usuario.

Destroy devolverá un flujo REST tipo DELETE de ahí que es distinta.

Capítulo 8: La vista new y edit

Capítulo obsoleto las vistas generadas por Rails 3 utilizan parciales. Las vistas new y edit son idénticas en función a excepción de algunos cambios de formason llamadas con http://0.0.0.0:3000/usuarios/new y http://0.0.0.0:3000/usuarios/1/edit, donde obedecen completamente a la lógica del URL.

Me resulta muy interesante ver que existe un generador de XML en blanco que solo contiene la estructura pero no los datos.

En ambos casos se genera la variable @usuario, en singular por que solo es uno, el modelo Usuario puede reservar un espacio para una nueva tupla con el método new, en contra parte el modelo Usuario y el método find pueden extraer de los parámetros enviados de la vista el Id para extraer esa tupla en particular.

./app/controllers/usuarios_controller.rb

```
  def new
    @usuario = Usuario.new

    respond_to do |format|
      format.html # new.html.erb
      format.xml  { render :xml => @usuario }
    end
  end

  def edit
    @usuario = Usuario.find(params[:id])
  end
```

Por simplicidad solo mostrare new.html.erb, edit.htm.erb es completamente equivalente.

La instrucción form_for genera una forma que presenta y recauda los datos del objecto f que fue inicializado con los datos de @usuario, lo interesante no sucede aquí, sino en el retorno, donde si la llamada fue generada por new, entonces retornara a la acción create, pero si fue generada por edit, entonces retornara a la acción update, eso lo sabe el enrutador que se configura en ./config/routes.rb, gracias al estandard REST.

El enrutador evalúa si el REST es un PUT entonces efectúa update, si es un POST entonces efectúa new, POST y PUT son flujo ocultos y a veces encriptados de datos que van de la vista al controlador para grabaciones, en contraparte de GET que es solo de consulta.

./app/views/usuarios/new.html.erb

```
<h1>New usuario</h1>

<% form_for(@usuario) do |f| %>
  <%= f.error_messages %>

  <p>
    <%= f.label :nombre %><br />
    <%= f.text_field :nombre %>
  </p>
  <p>
    <%= f.label :hilo_id %><br />
    <%= f.text_field :hilo_id %>
  </p>
  <p>
    <%= f.label :texto %><br />
    <%= f.text_area :texto %>
  </p>
  <p>
    <%= f.label :estado %><br />
    <%= f.text_field :estado %>
  </p>
  <p>
    <%= f.submit 'Create' %>
  </p>
<% end %>
<%= link_to 'Back', usuarios_path %>
```

Capítulo 9: Las acciones create, update y destroy

Existen 4 flujos REST que genera el visualizador, GET, PUT, POST y DELETE, en el momento en que se recibe una página, también se recibe el método con que se devolverá el resultado, típicamente solo se utilizan 2, POST y GET, pero el estándar Web 2.0 requiere de los 4.

La acción create se ejecutara cuando se devuelva un REST POST, la acción update cuando se devuelva un REST PUT y la destroy cuando se devuelva un REST DELETE. Es por esa razón que el link_to de la forma index para destroy es distinta, REST GET es inofensivo así que se utiliza para index y show.

Atención a lo que hace create, extrae de lo que ha enviado lo la vista la tupla usuario, y en la misma instrucción genera una nueva tupla, despues la graba y si el resultado es satisfactorio se va a la acción show con la instruccion redirect_to (@usuario)

Update es muy similar, busca los datos temporales en el modelo con el id que le envía la vista e intenta hacer una actualización de atributos.

flash[:notice] es un mensaje que se envía al template de la aplicación.

./app/controllers/usuarios_controller.rb

```
  def create
    @usuario = Usuario.new(params[:usuario])

    respond_to do |format|
      if @usuario.save
        flash[:notice] = 'Usuario was successfully created.'
        format.html { redirect_to(@usuario) }
        format.xml  { render :xml => @usuario, :status
=> :created, :location => @usuario }
      else
        format.html { render :action => "new" }
        format.xml  { render :xml =>
@usuario.errors, :status => :unprocessable_entity }
      end
    end
end

  def update
    @usuario = Usuario.find(params[:id])

    respond_to do |format|
      if @usuario.update_attributes(params[:usuario])
        flash[:notice] = 'Usuario was successfully updated.'
        format.html { redirect_to(@usuario) }
        format.xml  { head :ok }
      else
        format.html { render :action => "edit" }
        format.xml  { render :xml =>
@usuario.errors, :status => :unprocessable_entity }
      end
    end
end

  def destroy
    @usuario = Usuario.find(params[:id])
    @usuario.destroy

    respond_to do |format|
      format.html { redirect_to(usuarios_url) }
      format.xml  { head :ok }
    end
end
```

Destroy es muy simple de entender y no tiene control de errores, busca el Id del usuario y lo borra. Todas las cláusulas respond_to contiene código para evaluar si se devolverá una vista HTML o XML no las comentaré por que no estamos interesados en este momento en el XML.

Capitulo 10: El modelo

El modelo contiene las reglas del negocio y las relaciones entre modelos, por reglas de negocio debemos entender todas esas funciones de verificación que permiten a los datos ser o no legítimos, y por relaciones la interacción lógica de las tablas que a su vez están representadas por modelos. Para nuestra aplicación “Foro” que por cierto ya se me estaba olvidando, tenemos varias relaciones.

La tabla usuarios tiene una relación de uno a muchos con respecto a las tablas hilos y comentarios, por lo tanto debemos denotar eso en nuestro modelo agregando las cláusulas *has_many*, adicionalmente no es una buena idea que un usuario no tenga nombre, entonces validamos la presencia de ambos atributos.

```
./app/models/usuario.rb
class Usuario < ActiveRecord::Base
      has_many :hilos
      has_many :comentarios

      validates_presence_of :nombre
end
```

La tabla hilo tiene una relación de uno a muchos con la tabla usuarios, así que agregamos esa relación con un *belongs_to*, véase la singularización de usuario.

Y como un hilo puede tener muchos comentarios agregamos las instrucciones pertinentes, también no es una buena idea que alguien deje un hilo sin texto, tampoco nos conviene que hayan 2 hilos repetidos, y pensandolo bien el comentario también se debe validar para evitar que tenga textos en blanco

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
      belongs_to :usuario
      has_many :comentarios

      validates_presence_of :texto
      validates_uniqueness_of :texto
end
```

El comentario debe hacer referencia a un usuario y a un hilo y por supuesto debe tener texto, de ahí los 2 *belongs_to* y el *validates*. Por cada *has_many* debe haber una contraparte *belongs_to* en el otro modelo.

```
./app/models/comentario.rb
class Comentario < ActiveRecord::Base
      belongs_to :usuario
      belongs_to :hilo

      validates_presence_of :texto
end
```

Por favor haga los cambios en sus archivos y **reinicie el servidor**, los modelos son recreados en cada inicio, este paso no es necesario cuando se hacen cambios en los controladores o las vistas.

Ahora intente crear un usuario sin nombre y se dará una grata sorpresa, el manejo de errores es virtualmente mágico en Rails.

En el capítulo 13 se verá la importancia de has_many y belongs_to.

Tabla diferente a la esperada

Si iniciamos con Rails, no tendremos que hacer esto, pero que hay si ya existe la base de datos y los nombres de las tablas son cualquier cosa menos nuestro estandard, es por eso que tenemos que especificar el mismo al igual que las llaves foráneas, es decir que si la tabla fuera otra entonces utilizamos el comando set_table_name, similarmente tenemos las llaves foráneas que las podemos definir con el símbolo foreign_key, por funcionalidad las dejamos idénticas al default.

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
   set_table_name "hilo"

   belongs_to :usuario, :foreign_key=>"usuario_id"
   has_many :comentarios, :foreign_key=>"comentario_id"

   validates_presence_of :texto
   validates_uniqueness_of :texto

   def initialize(attribs={})
      super(attribs)
      if !self.texto.present?
         self.texto = "Escriba aquí ..."
      end
      self
   end
end
```

Valores por defecto

Esta es una de las acciones típicas dentro de una aplicación pero hay que tomarlo con calma. es decir que no es correcto inicializar una atributo ya inicializado, sino hay que saber controlar cuando esta variable ya contiene algo útil, esto se hace con el método present?, este método evalúa si tiene nulo y así sabemos que no ha sido asignada previamente, el signo de admiración es una negación del método. es decir si "no ha sido asignada previamente" entonces asígnele el texto "Escriba aquí ..."

self es el envío del objeto mismo para que los atributos sean modificados y super (attrib) es el que permite que los atributos de la clase sean accesibles a este método para poderlos modificar, attribs={} es el conjunto de atributos que la clase le envía al constructor initialize.

Otras validaciones

No existe solamente *validates_presence_of* o *validates_uniqueness_of* es posible validar si un campo contiene un número, o tiene un formato en lo específico, etc. Todas estas validaciones están documentadas en http://api.rubyonrails.org/

validates_acceptance_of	validates_inclusion_of
validates_associated	validates_length_of
validates_confirmation_of	validates_numericality_of
validates_each	validates_presence_of
validates_exclusion_of	validates_size_of
validates_format_of	validates_uniqueness_of

Integridad Referencial

Rails tiene una forma particular de manejar este tema, al parecer como que no es amante de la integridad referencial o quiere dejar que la base de datos la maneje, en fin, Rails es totalmente destructivo o mas bien no constructivo.

En general una integridad referencial decente debería evitar que un registro padre fuera eliminado si tiene hijos, o crear un registro hijo sin padre, la solución que Rails propone es la siguiente:

En un *has_many* puede haber un símbolo *:dependent* que puede apuntar a *:destroy, :delete_all* o *:nullify*, si es *:destroy* entonces todos los objetos asociados son destruidos cuando este objeto llame al método *destroy*, es decir cuando sea borrada la tupla padre se destruirán los hijos uno por uno, si es *:delete_all* similar a *:destroy* solo que se borrarán en grupo así que si en el modelo del hijo existen métodos especiales para cuando sean borrados, estos no se ejecutarán, y *:nullify* que me parece la más estúpida de las 3, dice que las llaves foráneas de los hijos tendrán nulos.

En un *belongs_to*, el simbolo *:dependent* solo puede apuntar a *:destroy y :delete* donde *:destroy* quiere decir que esta tupla se borrará si el objeto padre es borrado, y *:delete* idéntico a *:destroy* pero con la diferencia que si hay métodos asociados a la destrucción de esta tupla, no se ejecutarán.

```
./app/models/hilo.rb
class Hilo < ActiveRecord::Base
   set_table_name "hilo"

   belongs_to :usuario, :foreign_key=>”usuario_id”
   has_many :comentarios, :foreign_key=>”comentario_id”, :dependent=>:destroy

   validates_presence_of :texto
   validates_uniqueness_of :texto

   def initialize(attribs={})
      super(attribs)
      if !self.texto.present?
         self.texto = “Escriba aquí ...”
      end
      self
   end
end
```

Para nuestro ejemplo agregaremos un *:dependent=>:destroy* a comentarios, es decir que si este hilo es borrado destruiremos automáticamente sus comentarios.

Tome en cuenta que para cada asociación del tipo *hilo.usuario.nombre* deberá haber un *rescue,* especialmente si estará utilizando *:nullify* en la tabla usuario.

Capítulo 11: La plantilla

En ninguna parte de la aplicación se han declarado elementos visuales tales como distribución, coloración, tipos de letra o íconos, esta omisión es totalmente deliberada, pero al fin y al cabo es una aplicación web y se supone debe lucir agradable.

La parte visual no debe interferir con la funcional en ningún momento, de esa forma el programador esta perfectamente enfocado en su trabajo y el maquillaje se realizará muy a gusto del cliente sin necesidad de tocar una linea del código ya estable.

- Copiaremos ./app/views/layouts/usuario.html.erb a ./app/views/layouts/application.html.erb
- Eliminaremos usuarios.html.erb, hilos.html.erb y comentarios.html.erb

El propósito de la eliminación es evitar que los controladores busquen sus propias plantillas y se apeguen a la de aplicación, que es una plantilla pública en donde colocaremos un menú simple de navegación.

Habrá que cambiar el código original por este o alguno parecido, este intenta ser un ejemplo muy simplista de lo que se puede hacer.

./app/views/layouts/application.html.erb

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
  <title>El foro mas grande del mundo</title>
  <%= stylesheet_link_tag 'scaffold' %>
</head>
<body>
<a href="/usuarios">Usuarios</a>  
<a href="/hilos">Hilos</a>  
<p style="color: green"><%= flash[:notice] %></p>
<%= yield %>
</body>
</html>
```

yield es donde se insertará el código de la vista, es decir que toda la vista estará embebida en ese espacio, así que tendremos un encabezado y un pie de pagina común para todas las vistas, *stylesheet_link_tag* apunta a **./public/stylesheets/scaffold.css** y es ahí donde modificaremos el css.

El capitulo sobre el maquillaje muestra un ejemplo, sin embargo la plantilla debe ser de índole concretamente funcional, es decir que colocaremos menues, avisos, alguna distribución necesaria, etc. pero no elementos de mejora visual, ya que eso es tarea exclusiva del CSS.

Siempre deje al último la parte de mejora visual, en muchos casos se necesitará de un profesional en el área o de encontrar algún template pre-armado que nos sirva, recuerde que el cliente puede elegir las mejoras visuales arbitrariamente y por lo mismo será un esfuerzo en vano hacerlo en este momento.

Capitulo 12: Las rutas

El URL identifica al controlador, la acción, el identificador y en algunos casos, parámetros adicionales, sin embargo no estamos restringidos a esa distribución, ya que podemos modificar las rutas en el archivo **./config/routes.rb**

```
./config/routes.rb
ActionController::Routing::Routes.draw do |map|
  map.resources :usuarios

  map.resources :comentarios

  map.resources :hilos
  map.connect 'mishilos', :controller=>'hilos',:action=>'mios'

  map.root :controller => "hilos"

  map.connect ':controller/:action/:id'
  map.connect ':controller/:action/:id.:format'
end
```

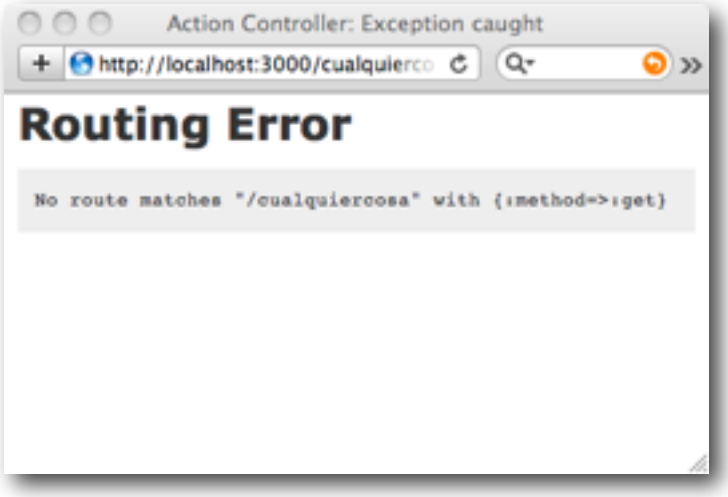
Las líneas con # han sido eliminadas por razones de espacio, estas líneas son comentarios y sirven de ejemplo para las distintas formas que puede tomar una ruta, en nuestro caso haremos 2 cambios, primero cambiaremos el controlador default con map.root apuntandolo a hilos, e inmediatamente después borrar el archivo **./public/index.html** con esto logramos que Rails busque en las rutas el controlador hilos en vez de la página de bienvenida.

Las declaraciones *map.resources*, crean 7 rutas para *index, show, new, edit, create, update y delete*, para cada controlador, sin embargo si es necesario crear alguna ruta adicional, por ejemplo http://0.0.0.0:3000/mishilos, entonces simplemente declaramos un *map.connect*, y elegantemente aislamos al usuario del controlador y la acción a utilizar.

Es posible enmascarar cualquier tipo de acción, pero también es menester crearla en el archivo de rutas si no es una de las 7 mencionadas arriba.

En el caso de *map.connect ':controller/:action/:id.:format'* esto indica a Rails de que forma manejará los controladores, las acciones, el identificador y el formato, los símbolos antecedidos de 2 puntos indican la posición, ya que las rutas funcionan con prioridad descendente si se declarara otro orden arriba de esta instrucción entonces primeramente se intentaría con la ruta agregada y a posteriori con esta que es la default.

Error de rutas



El error en las rutas es algo normal, a lo mejor un usuario queriendose pasar de listo nos coloca una dirección que no está contemplada en el ruteo, causando el *error de rutas*, al no ser un 404 ni nada que el usuario está acostumbrado da la sensación de descontrol y de mala calidad en la aplicación.

La solución es simple, en el archivo **./app/controllers/applicaction_controller.rb** donde insertamos el método *rescue_action* donde exception es el error, así que redireccionaremos hacia la vista *error* del controlador *application*

```
./app/controllers/applicaction_controller.rb
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See
ActionController::RequestForgeryProtection for details

  # Scrub sensitive parameters from your log
  # filter_parameter_logging :password

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
       @exception=exception
       render :controller => 'application', :action=>"error"
    else
       super
    end
  end
end
```

El controlador application ya existe, pero curiosamente no tiene vistas, de hecho no tiene ni el direcctorio para alojarlas, aunque es posible direccionar a cualquier vista, en general podemos armar nuestros mensajes de error personalizados para cualquier cosa, asi que creamos el directorio **./app/views/application** y dentro de él agregamos el archivo **./app/views/application/error.html.erb** donde colocaremos un texto simple para motivos ilustrativos.

```
./app/views/applicaction/error.html.erb

Error de rutas
```

Con esto tenemos un error personalizado, por favor vease que se hace referencia a *::ActionController::RoutingError* y se compara con *exception* para poder determinar el tipo de error y poder actuar de forma afín, por favor vea el capítulo 17 donde se habla de otros tipos de errores.

Capitulo 13: La consola y el ORM

Rails tiene una consola de pruebas muy efectiva para probar algoritmos, administrar datos, ejecutar metodos y dar seguimiento a variables u objetos, en realidad no es propio de Rails sino de Ruby y es la IRB que es la consola interactiva de Ruby, sin embargo al invocarla dentro de Rails, todos los modelos, helpers, bibliotecas y controladores estarán disponibles.

```
usuario@host:~/rails/prueba1$ ./script/console
```

```
Loading development environment (Rails 2.3.8)
>>
```

El ORM es el mapeo de objetos relacionales, y se encarga de ser la cara de los modelos y la relación con los otros modelos. En este momento vamos a ingresar datos por la consola, los cambios podrán ser vistos en el visualizador cuando se invoque el servidor. El >> indica lo que nosotros escribimos y => la respuesta.

```
>> Usuario
=> Usuario(id: integer, nombre: string, created_at: datetime, updated_at: datetime)
>> a=Usuario.new
>> a.nombre="Perico"
>> a.save
=> true
```

Como tenemos reglas en los modelos estas evitarán que se grabe un hilo sin texto.

```
>> Hilo
=> Hilo(id: integer, texto: text, usuario_id: integer, estado: integer, created_at:
datetime, updated_at: datetime)
>> h=Hilo.new
>> Usuario.all
=> [#<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">]
>> h.usuario_id=1
>> h.estado=1
>> Hilo.all
=> []
>> h.save
=> false
>> h.texto="Que le parece Ruby on Rails??"
>> h.save
=> true
```

Otro hilo más para ejemplo, y revisamos que todo esté en orden.

```
>> h=Hilo.new
>> h.usuario_id=1
>> h.estado=1
>> h.texto="El ORM es maravilloso!!"
>> h.save
=> true
>> Hilo.all.count
=> 2
```

Ahora un poco de la magia del ORM

```
>> h=Hilo.find(:first,:conditions=>"texto like '%ORM%'")
=> #<Hilo id: 2, texto: "El ORM es maravilloso!!", usuario_id: 1, estado: 1, created_at:
"2010-07-31 03:09:37", updated_at: "2010-07-31 03:09:37">
>> h.usuario
=> #<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">
>> h.usuario.nombre
=> "Perico"
>> u=h.usuario
=> #<Usuario id: 1, nombre: "Perico", created_at: "2010-07-31 03:05:22", updated_at:
"2010-07-31 03:05:22">
>> u.hilos.count
=> 2
>> u.hilos.collect(&:texto)
=> ["Que le parece Ruby on Rails??", "El ORM es maravilloso!!"]
```

Gracias a las instrucciones *has_many* y *belongs_to* tenemos que los modelos adoptan a otros modelos como propios, en una estructura n-dimensional donde las herencias pierden el sentido de antecesores y sucesores. De ahí que podemos hacer cosas como esta:

```
>> Hilo.first.usuario.hilos[1].texto
=> "El ORM es maravilloso!!"
>> Usuario.all[0].hilos[1].usuario.nombre
=> "Perico"
```

El ORM se encarga de manejar la lógica relacional y no tenemos nunca que preocuparnos de recuperar datos aisladamente, si el modelo esta relacionado con otro modelo, entonces sus datos puede ser recuperados por relación sin importar cuantos modelos pueda tener en el intermedio, ingresemos un comentario para el ejemplo.

```
>> c=Comentario.new
>> c.usuario_id=1
>> c.hilo_id=2
>> c.texto="Es increible lo bien que funciona"
>> c.estado=1
>> c.save
=> true
```

Ahora podemos involucrar una tabla más

```
>> comentario=Comentario.first
=> #<Comentario id: 1, usuario_id: 1, hilo_id: 2, texto: "Es increible lo bien que
funciona", estado: 1, created_at: "2010-07-31 04:02:50", updated_at: "2010-07-31
04:02:50">
>> comentario.hilo.texto
=> "El ORM es maravilloso!!"
>> comentario.hilo.usuario.nombre
=> "Perico"
```

El ORM de Rails se llama ActiveRecord y puede utilizarse por separado, pero no veo por que razón alguien quisiera hacer eso.

El ORM es una de las piezas mas valiosas y sólidas dentro del abanico de herramientas que ofrece Rails, sin embargo también se puede abusar de él con instrucciones del tipo: Hilo.find_by_sql("select * from usuarios u, hilos h where h.usuario_id=u.id")

Capítulo 14: La consola de la base de datos y la base de datos

La consola de la base de datos es siempre la de la base de datos conectada y se invoca:

```
usuario@host:~/rails/prueba1$ rails db
```

En función de como fue creada la base de datos así es como deberemos seguir su mantenimiento, es decir que si nosotros la creamos a nivel de migraciones entonces se vuelve menester hacerlo de esa manera. Sin embargo en no pocos casos tendremos que trabajar con bases de datos ya establecidas y probablemente en producción, por lo mismo, la opción de la migración se vuelve inadecuada y queda en total potestad del DBA y su DDL el hacer los cambios al esquema.

Hay 3 ambientes: Producción, prueba y desarrollo, el ambiente de desarrollo, es el default y los otros 2 se invocarán en el momento de iniciar el servidor o modificando el archivo **./config/environment.rb** agregando **ENV['RAILS_ENV'] ||= 'production'**

El archivo **./config/database.yml** contiene la información pertinente de la base de datos de cada ambiente, de ahí que la consola de la base de datos puede variar.

Quizás la forma mas simple de crear una aplicación para una base de datos puntual es desde el inicio con el comando rails -d donde las opciones son mysql, oracle, postgresql, sqlite2, sqlite3, frontbase, ibm_db. Por default trabajará con sqlite3.

```
 ./config/database.yml
# SQLite version 3.x
#   gem install sqlite3-ruby (not necessary on OS X Leopard)
development:
  adapter: sqlite3
  database: db/development.sqlite3
  pool: 5
  timeout: 5000

# Warning: The database defined as "test" will be erased and
# re-generated from your development database when you run "rake".
# Do not set this db to the same as development or production.
test:
  adapter: sqlite3
  database: db/test.sqlite3
  pool: 5
  timeout: 5000

production:
  adapter: sqlite3
  database: db/production.sqlite3
  pool: 5
  timeout: 5000
```

Al ser agnóstico a la base de datos Rails no necesariamente promueve acciones propias de la misma como la integridad referencial, procedimientos embebidos o índices compuestos, estas acciones vienen en código dentro de Rails y pueden impactar negativamente en el desempeño y si la base de datos tiene reglas que riñen con las de la aplicación entonces se puede generar un error fatal, así que es necesario que ambos, aplicación y base de datos estén muy de acuerdo.

Al parecer el desempeño es el punto débil de Rails ya que los puristas lo han considerado lento, pero hay mejoras constantes en esta área y estoy convencido que la versión actual dará resultados mas que satisfactorios a la mayoría.

Capítulo 15: La aplicación

Le daremos funcionalidad a los hilos y lo castellanizaremos, en primer lugar como ya se vió, la vista new y la vista edit son virtualmente idénticas, por lo tanto podemos aplicar una vista parcial, donde tomamos el código común de ambas y lo pegamos en **./app/views/hilos/_edit.html.erb**

```
./app/views/hilos/edit.html.erb
<h1>Modificar Hilo</h1>

<% form_for(@hilo) do |f| %>
  <%= render :partial=>"edit", :locals=>{:f=>f} %>
    <%= f.submit 'Actualizar' %>
  </p>
<% end %>

<%= link_to 'Mostrar', @hilo %> |
<%= link_to 'Retornar', hilos_path %>
```

```
./app/views/hilos/new.html.erb
<h1>Nuevo Hilo</h1>

<% form_for(@hilo) do |f| %>
  <%= render :partial=>"edit", :locals=>{:f=>f} %>
    <%= f.submit 'Crear' %>
  </p>
<% end %>

<%= link_to 'Retornar', hilos_path %>
```

Y creamos el archivo parcial que tiene como prefijo un guión mayor _

La belleza de esto radica en que un cambio en el archivo **_edit.html.erb** afecta tanto a new como a edit.

La instrucción *render* visualiza la vista parcial y el símbolo *locals* envía a la vista parcial cualquier objeto o colección necesaria.

En el caso de *f.text.area* tiene un área de texto enorme que quizás deba ser modificada para que sea mas ancha y menos alta así que simplemente enviamos los parámetros adecuados.

```
./app/views/hilos/_edit.html.erb
<%= f.error_messages %>
<p>
  <%= f.label :texto %><br />
  <%= f.text_area :texto, :rows=>2, :cols=>40 %>
</p>
<p>
  <%= f.label :usuario_id %><br />
  <%= f.collection_select :usuario_id,Usuario.all,:id,:nombre
%>
</p>
<p>
  <%= f.label :estado %><br />
  <%= f.radio_button :estado,"1" %> Activo
  <%= f.radio_button :estado,"2" %> Inactivo
```

En el caso del *usuario_id* colocaremos una muy conveniente selección para apoyar la integridad referencial además que luce mucho mas profesional.

También habrá que colocar estados mas legibles, y declararemos 2, activo e inactivo con un par de radio button.

Utilice siempre que le sea posible vistas parciales esto modularizará su código y evitara repetirlo, lo cual siempre le dará la certeza de donde modificar en cuanto esto se vuelva necesario.

Así como hay *radio_button*, existe *check_box*, *date_select*, *text_area*, *text_field*, *hidden_field* e inclusive *file_field* entre otros, estos objetos son de la clase *form_for*. La documentación completa del API de Rails está en http://api.rubyonrails.org/

En el caso de *index* se requerirá de mucho maquillaje ya que debe ser coherente con la aplicación:

```
./app/views/hilos/index.html.erb
<h1>Listado de Hilos</h1>
<table>
  <tr>
    <th>Texto</th>
    <th>Usuario</th>
    <th>Estado</th>
  </tr>
<% @hilos.each do |hilo| %>
  <tr>
    <td><%= link_to hilo.texto, hilo %></td>
    <td><%=h hilo.usuario.nombre rescue “Usuario no asignado” %></td>
    <td><%= case hilo.estado
                when 1
                        "Activo"
                when 2
                        "Inactivo"
            end
        %></td>
    <td><%= link_to image_tag("edit.png"), edit_hilo_path(hilo) %></td>
    <td><%= link_to image_tag('delete.png'), hilo, :confirm => '¿Está seguro?', :method => :delete %></td>
  </tr>
<% end %>
</table>
<h2><%= @hilos.count %> Comentarios</h2>
<br />
<%= link_to 'Crear un nuevo Hilo', new_hilo_path %>
```

En vez de tener una opción show a la derecha se modificó la instrucción para que el link sea ahora el texto del hilo, por otro lado usando el ORM se ha extraído el nombre del usuario en vez de solamente usuario_id con la instrucción *hilo.usuario.nombre,* la cláusula *rescue* es sumamente importante, ya que controla y minimiza el error colocando el texto cuando se suceda un error.

Control de errores

rescue después de cualquier instrucción evalúa la expresión de la derecha si se encuentra con algún tipo de error, en el caso de *hilo.usuario.nombre*, si *hilo.usuario_id* es nulo entonces el objeto *hilo.usuario* no es viable de ahí que Rails nos envíe un error de método no encontrado “nombre”, es una forma muy elegante de enviar un mensaje de error y no lo parezca, añadido a que la aplicación podrá continuar sin mayores percances.

Por favor vea la elegancia de Ruby en la condición del estado, el valor de retorno será el string del condición verdadera.

Los *link_to* pueden recibir como parámetros *image_tag* que por default buscará las imágenes en el directorio **./public/images**, estas imágenes fueron colocadas ahí manualmente. Los paths *hilo* y *edit_hilo_path* son generados automáticamente por las rutas así que no hay que preocuparse por crearlos en algún lugar.

El *link_to* del *delete*, funciona enviando los datos de hilo pero con un método REST de tipo DELETE de ahí que el controlador sabe que debe borrar el hilo que le envían.

Se colocó el número de hilos con el método *count* que simplemente cuenta el número filas de la colección a la que pertenece

No es una buena idea colocar el botón de borrar a la par del botón de edición, se ha dejado así por motivos ilustrativos.

Show debería mostrar el hilo que actualmente vemos, sus comentarios y a la vez permitir el ingreso de nuevos, esto implica una funcionalidad mezclada en una misma vista:

```
./app/views/hilos/index.html.erb
<h1><%=h @hilo.texto %></h1>

<table>
  <tr>
    <th>Usuario</th>
    <th>Texto</th>
  </tr>
<% @hilo.comentarios.each do |comentario| %>
  <tr>
    <td><%= link_to (comentario.usuario.nombre rescue “Sin usuario”), edit_comentario_path(comentario) %></td>
    <td><%=h comentario.texto %></td>
    <td><%= link_to image_tag('delete.png'), comentario, :confirm => '¿Está seguro?', :method => :delete %></td>
  </tr>
<% end %>

<% form_for(@hilo.comentarios.new) do |f| %>
  <%= f.error_messages %>
  <%= f.hidden_field :hilo_id %>
  <td> <%= f.collection_select :usuario_id,Usuario.all,:id,:nombre %></td>
  <td><%= f.text_area :texto, :rows=>2,:cols=>40 %></td>
</table>
<%= f.submit 'Grabar' %>
<% end %>
```

El truco para este tipo de vistas compuestas es copiar y pegar el código ya existente de las otras vistas, y solo se modifica lo que sea necesario.

Aquí aparecen todos los viejos conocidos, sin embargo hay algunos nuevos, nótese la agilidad con la que se crea la tabla donde se mezclan los comentarios y la forma para ingresar nuevos. *@hilo.comentarios.each* es simplemente la forma en que llamamos a los comentarios de este hilo gracias al ORM, pero nótese también otra cosa: *@hilo.comentarios.new* genera un comentario en blanco para este hilo, pero se hace menester el uso del *hidden_field*, ya que sin él, el controlador no sabrá a que hilo pertenece al momento de la grabación.

Véase el uso de *rescue* en la cláusula *comentario.usuario.nombre*, esto nos asegurará que el comentario tenga algo que mostrar aún si el *usuario_id* es nulo.

Aún hay un problema y luce grave, al momento de grabar nos debería mostrar el cambio y quedarse en esta misma vista, para eso necesitaremos modificar el controlador de grabación del comentario.

```
./app/controllers/comentarios_controller.rb (Solo parte)
  def create
    @comentario = Comentario.new(params[:comentario])

    respond_to do |format|
      if @comentario.save
        format.html { redirect_to :controller=>"hilos", :action=>"show", :id=>@comentario.hilo_id}
        format.xml  { render :xml => @comentario, :status => :created, :location => @comentario }
      else
        format.html { render :action => "new" }
        format.xml  { render :xml => @comentario.errors, :status => :unprocessable_entity }
      end
    end
  end
```

En el momento en que el comentario ha sido grabado exitosamente se invoca el comando *redirect_to* con toda la descripción del controlador y la acción, pero show requiere de un id, que extraemos fácilmente de la llave foránea del comentario hacia el hilo, *@comentario.hilo_id*.

Ahora tenemos un problema similar cuando borramos un comentario.

```
./app/controllers/comentarios_controller.rb (Solo parte)
  def destroy
    @comentario = Comentario.find(params[:id])
    hilo_id=@comentario.hilo_id
    @comentario.destroy

    respond_to do |format|
      format.html { redirect_to :controller=>"hilos", :action=>"show", :id=>hilo_id }
      format.xml  { head :ok }
    end
  end
```

Las circunstancias en este caso no son iguales ya que el comentario será borrado por lo tanto astutamente hemos sacado una copia de *hilo_id* antes que eso suceda y es ahora nuestro id para la instrucción *redirect_to*.

Con estos cambios se hacen innecesarias las vistas creadas por el scaffold para los comentarios, pero ya cumplieron su cometido y están ahí perfectamente disponibles y funcionales para en la etapa de desarrollo, sin embargo habrá que eliminarlas en cuanto se pase a la etapa de producción, ya que es código que simplemente no será usado, y por lo mismo no nos conviene tener una fuente de confusión a la hora del mantenimiento y soporte de la aplicación.

Ahora tendremos que castellanizar las vistas del usuario, esto asumiré que el lector puede realizarlo ya que conlleva exactamente los mismos pasos que las vistas de hilo, por favor no haga un parcial en este momento para edit y new, y modificaremos la vista show ya que será especial, nos mostrará todos los hilos y los comentarios del usuario así como también sus generales.

La vista show de usuario quedaría algo así:

```
./app/views/usuarios/show.html.erb
<h1><%=h @usuario.nombre %></h1>
Hilos creados por <%=h @usuario.nombre %>
<table>
<% @usuario.hilos.each do |hilo| %>
  <tr>
    <td><%= link_to hilo.texto, hilo %></td>
    <td><%=
            "Activo" if hilo.estado==1
            "Inactivo" if hilo.estado==2
        %></td>
  </tr>
<% end %>
</table>

<br>
Comentarios hechos por <%=h @usuario.nombre %>
<table>

<% @usuario.comentarios.each do |comentario| %>
  <tr>
    <td><%= link_to comentario.texto, comentario.hilo %></td>
  </tr>
<% end %>
</table>
```

Acá vuelven a aparecer los sospechosos de siempre, sin embargo en el *link_to* de los comentarios hay algo interesante, mostramos el texto del comentario, pero el *link_to* apunta al hilo del comentario, con *comentario.hilo*, es decir que el path puede también ser relacionado, aunque en realidad lo que sucede es que simplemente analiza si es uno o varios para invocar la vista show o index del modelo.

Capítulo 16: El maquillaje

Al finalizar el desarrollo, siempre se vuelve menester hacer cambios en la visual, pero estos cambios en general no deben realizarse en las vistas es preferible un buen archivo CSS, como ya se había dicho el stylesheet que es llamado en **./app/views/layouts/application.html.erb** es el **./public/stylesheets/scaffold.css,** sin embargo lo cambiaremos por algo un poco mas agradable a la vista.

./public/stylesheets/scaffold.css

```
body {
	margin: 30;
	background: #7A287A;
	font-family: "Trebuchet MS",
Arial, Helvetica, sans-serif;
	font-size: 13px;
	color: #C752C7;
}

h1, h2, h3 {
	margin: 0;
	font-family: Georgia, "Times New
Roman", Times, serif;
	font-weight: normal;
	color: #649632;
}

h1 { font-size: 44px; }

h2 { font-size: 20px; }

p, ul, ol {
	margin-top: 0;
	line-height: 240%;
	text-align: justify;
	font-family: "Trebuchet MS",
Arial, Helvetica, sans-serif;
	font-size: 11px;
}

a { color: #FFFFFF; }

th {
	background: #792879;
	color: #fff;
}

#errorExplanation {
  width: 400px;
  border: 2px solid #792879
  padding: 7px;
  padding-bottom: 12px;
  margin-bottom: 20px;
  background-color: #f0f0f0;
}

#errorExplanation h2 {
  text-align: left;
  font-weight: bold;
  padding: 5px 5px 5px 15px;
  font-size: 12px;
  margin: -7px;
  background-color: #792879;
  color: #fff;
}

#errorExplanation p {
  color: #333;
  margin-bottom: 0;
  padding: 5px;
}

#errorExplanation ul li {
  font-size: 12px;
  list-style: square;
}

.fieldWithErrors {
  padding: 2px;
  background-color: #ff54ff;
  color: #000;
  display: table;
}
```

No mejoró, pero es innegable que cambió, el CSS es muy potente y se pueden realizar excelentes trabajos pero eso excedería la intensión de este texto. Lo más sencillo en este caso es buscar un maquetador o conseguir un template ya armado y simplemente adaptarlo.

Capítulo 17: Errores

Cuando se habla de errores de una aplicación es común pensar que fue mal programada, sin embargo existe una miríada de casos en los que una aplicación adecuadamente programada puede enviar un error, sin que esto signifique que algo anda mal, en capítulos anteriores hemos visto que una llave foránea nula puede llevarnos a una falla catastrófica por que simplemente el ORM no puede resolver los métodos del modelo asociado, de ahí el uso de *rescue,* aunque esto nos ayuda increíblemente, este es tan solo un tipo de error.

Algunos tipos de errores comunes y su manejador:

- Registro no encontrado, se maneja con ActiveRecord::RecordNotFound
- Recurso no encontrado, se maneja con ActiveResource::ResourceNotFound
- Error en plantilla (vista), se maneja con ActionView::TemplateError
- Error de rutas, se maneja con ActionController::RoutingError
- Controlador desconocido, se maneja con ActionController::UnknownController
- Método no permitido, se maneja con ActionController::MethodNotAllowed
- Autenticidad no válida, se maneja con ActionController::InvalidAuthenticityToken
- Acción desconocida, se maneja con ActionController::UnknownAction

```
./app/controllers/applicaction_controller.rb
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See ActionController::RequestForgeryProtection for details

  rescue_from ActiveRecord::RecordNotFound, :with => :no_encontrado #400
  rescue_from ActiveResource::ResourceNotFound, :with => :no_encontrado #404
  rescue_from ActionView::TemplateError, :with => :no_encontrado #500
  rescue_from ActionController::UnknownController, :with => :no_encontrado #404
  rescue_from ActionController::MethodNotAllowed, :with => :no_encontrado #405
  rescue_from ActionController::InvalidAuthenticityToken, :with => :no_encontrado #405
  rescue_from ActionController::UnknownAction, :with => :no_encontrado #501

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
         @exception=exception
         render :controller => 'application', :action=>"error"
    else
         super
    end
  end

  protected
  def no_encontrado
    render :text => "Error 404", :status => 404
  end

end
```

Aquí ya tenemos un muy decente manejo de errores, el maquillaje puede personalizarse a gusto, y deberá proveer la información del error, pero no deberá vender la idea de que algo malo sucedió, el símbolo *with* indica el método que manejará el error, en este caso todas son el mismo por motivos ilustrativos.

Capítulo 18: La seguridad y la sesión

No es muy correcto que cuando escribamos un comentario escojamos el usuario, de esta forma es muy simple que otro usuario haga un comentario a nuestro nombre, lo que se requiere es un manejador de sesión que es hermano del manejador de seguridad, uno se encarga de interrogarnos sobre nuestro usuario al iniciar la sesión, y el otro se encargará de pedirnos nuestra clave, de esa forma aseguramos que nuestros comentarios solo son nuestros y facilitamos la labor de ingreso puesto que en toda la sesión siempre seremos conocidos por nuestro nombre de usuario.

Toda aplicación para la web tiene una sección pública y una privada, la pública será para registro de nuevos usuarios, y el foro solo será visible hasta que nos autentiquemos.

```
./app/controllers/applicaction_controller.rb
class ApplicationController < ActionController::Base
  helper :all # include all helpers, all the time
  protect_from_forgery # See ActionController::RequestForgeryProtection for details

  rescue_from ActiveRecord::RecordNotFound, :with => :no_encontrado #400
  rescue_from ActiveResource::ResourceNotFound, :with => :no_encontrado #404
  rescue_from ActionView::TemplateError, :with => :no_encontrado #500
  rescue_from ActionController::UnknownController, :with => :no_encontrado #404
  rescue_from ActionController::MethodNotAllowed, :with => :no_encontrado #405
  rescue_from ActionController::InvalidAuthenticityToken, :with => :no_encontrado #405
  rescue_from ActionController::UnknownAction, :with => :no_encontrado #501

  def rescue_action(exception)
    if ::ActionController::RoutingError === exception
         @exception=exception
         render :controller => 'application', :action=>"error"
    else
         super
    end
  end

  protected
  def no_encontrado
    render :text => "Error 404", :status => 404
  end

  before_filter :autenticar

  def salir
    session[:usuario]=nil
    session[:usuario_id]=nil
  end

  def autenticar
     authenticate_or_request_with_http_basic do |usuario,clave|
        recuperado=Usuario.find(:first,:conditions=>["nombre=?",usuario])
      if clave.crypt("ml")==recuperado.clave
        session[:usuario]=usuario
         session[:usuario_id]=recuperado.id
        true
      else
        false
      end
    end
  end
```

La función *authenticate_or_request_with_http_basic* provee una forma simplista aunque en muchos casos adecuada para obtener un usuario y una clave, al retornar la colección los distribuye en las variables *usuario* y *clave*, se busca al *usuario*, y luego se compara su *clave*, el método *crypt*, encripta la clave obtenida del usuario para que coincida con la guardada en la base de datos, véase que el parámetro debe ser igual al usado al momento de la grabación, esto se discutirá en detalle mas adelante cuando se modifique el modelo. *session* es muy importante por que mantiene su valor durante toda la sesión, de ahí que hacemos uso del método salir para desasignar *session*. El método *before_filter* es el primero en ser ejecutado cuando se inicia la sesión.

Pero hay un problema grave, no tenemos el atributo clave en nuestra tabla usuarios, habrá que añadirlo, simplemente invocamos la consola de la base de datos:

```
usuario@host:~/rails/prueba1$ ./script/dbconsole
```

```
SQLite version 3.6.12
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite>
```

Aquí tenemos control completo de la base de datos, así que modificaremos la tabla, no lo hacemos a través de las migraciones por que eso conllevaría sacar una copia de la tabla, eliminar la tabla, modificar la migración, correr *rake* y recuperar los datos, de ahí que si lo hubiéramos planeado desde el principio entonces nada de esto sería necesario, por eso siempre debe haber un diseño muy bueno antes de iniciar cualquier desarrollo, por motivos didácticos hemos omitido este campo deliberadamente.

```
sqlite> alter table usuario add clave string;
sqlite> .exit
```

Ahora ya tenemos donde alojar nuestra clave, sería muy estúpido de nuestra parte guardar las claves tal como se digitan, puesto que se vuelve posible averiguar cual es la clave de un usuario a nivel de programación y a nivel de DBA, entonces las guardaremos encriptadas.

Modificaremos el modelo del usuario y usaremos un método muy común que se denomina salar, de condimentar, es decir que tenemos un algoritmo clásico de encriptamiento pero le agregamos un ingrediente secreto más, para que aunque alguien sepa el algoritmo principal no pueda averiguar la totalidad de la receta. Rails maneja esto con mucha inteligencia, el método *crypt* recibe un parámetro de 2 caracteres, que representa el ingrediente extra, puede ser cualquier string que solo nosotros conozcamos para que genere una salida personalizada, *after_validation_on_create* correrá en cada creación,

```
./app/models/usuario.rb
class Usuario < ActiveRecord::Base
	has_many :hilos
	has_many :comentarios

	validates_presence_of :nombre, :clave

	def after_validation_on_create()
		self.clave=self.clave.strip.crypt("ml")
	end
end
```

pero no correrá en la edición por que no es inteligente volver a encriptar lo que ya estaba encriptado.

Mucho ojo con el uso de *strip* en el *after_validation_on_create* este método elimina tabuladores, espacios y enters al final y al principio de la clave.

Ahora agregaremos el campo clave a la vista new de usuario que por supuesto es un *password_field*.

Con este cambio podremos capturar la clave al momento de crear el usuario, pero habrá que hacer una forma especializada para el cambio de la clave, ya que no podrá ser la vista *edit*, por que este siempre intentará modificar la clave ya existente, además es una buena idea que la persona digite 2 veces su clave para estar seguros, también es importante que la clave sea ininteligible, es decir el clásico formato de asteriscos o puntos para enmascararla y no sea descubierta por curiosos.

```
./app/views/usuarios/new.html.erb
<h1>Nuevo usuario</h1>

<% form_for(@usuario) do |f| %>
  <%= f.error_messages %>

  <p>
    <%= f.label :nombre %><br />
    <%= f.text_field :nombre %>
  </p>
  <p>
    <%= f.label :clave %><br />
    <%= f.password_field :clave %>
  </p>
  <p>
    <%= f.submit 'Grabar' %>
  </p>
<% end %>
```

Capítulo 19: Regionalización

Capitulo obsoleto funciona en Rails 2 pero no en Rails 3. La regionalización es sumamente simple, en el archivo /config/environment.rb hay que modificar la linea de locales a *config.i18n.default_locale = :es* y agregar el archivo es.yml a /config/locales, y eso es todo.

/config/locales/es.yml

```
es:
  number:
    format:
      separator: ","
      delimiter: "."
      precision: 3
    currency:
      format:
        format: "%n %u"
        unit: "€"
        separator: ","
        delimiter: "."
        precision: 2
    percentage:
      format:
        delimiter: ""
    precision:
      format:
        delimiter: ""
    human:
      format:
        delimiter: ""
        precision: 1
      storage_units:
        format: "%n %u"
        units:
          byte:
            one:   "Byte"
            other: "Bytes"
          kb: "KB"
          mb: "MB"
          gb: "GB"
          tb: "TB"
  datetime:
    distance_in_words:
      half_a_minute: "medio minuto"
      less_than_x_seconds:
        one:  "menos de 1 segundo"
        other: "menos de {{count}} segundos"
      x_seconds:
        one:  "1 segundo"
        other: "{{count}} segundos"
      less_than_x_minutes:
        one:  "menos de 1 minuto"
        other: "menos de {{count}} minutos"
      x_minutes:
        one:  "1 minuto"
        other: "{{count}} minutos"
      about_x_hours:
        one:  "alrededor de 1 hora"
        other: "alrededor de {{count}} horas"
      x_days:
        one:  "1 día"
        other: "{{count}} días"
      about_x_months:
        one:  "alrededor de 1 mes"
        other: "alrededor de {{count}} meses"
      x_months:
        one:  "1 mes"
        other: "{{count}} meses"
      about_x_years:
        one:  "alrededor de 1 año"
        other: "alrededor de {{count}} años"
      over_x_years:
        one:  "más de 1 año"
        other: "más de {{count}} años"
      almost_x_years:
        one: "casi 1 año"
        other: "casi {{count}} años"
    prompts:
      year:   "Año"
      month:  "Mes"
      day:    "Día"
      hour:   "Hora"
      minute: "Minutos"
      second: "Segundos"
  activerecord:
    errors:
      template:
        header:
          one:   "No se pudo guardar este {{model}} porque se encontró un error"
          other:  "No se pudo guardar este {{model}} porque se encontraron {{count}} errores"
        body: "Se encontraron problemas con los siguientes campos:"
      messages:
        inclusion: "no está incluido en la lista"
        exclusion: "está reservado"
        invalid: "no es válido"
        confirmation: "no coincide con la confirmación"
        accepted: "debe ser aceptado"
        empty: "no puede estar vacío"
        blank: "no puede estar en blanco"
        too_long: "es demasiado largo ({{count}} caracteres máximo)"
        too_short: "es demasiado corto ({{count}} caracteres mínimo)"
        wrong_length: "no tiene la longitud correcta ({{count}} caracteres exactos)"
        taken: "ya está en uso"
        not_a_number: "no es un número"
        greater_than: "debe ser mayor que {{count}}"
        greater_than_or_equal_to: "debe ser mayor que o igual a {{count}}"
        equal_to: "debe ser igual a {{count}}"
        less_than: "debe ser menor que {{count}}"
        less_than_or_equal_to: "debe ser menor que o igual a {{count}}"
        odd: "debe ser impar"
        even: "debe ser par"
        record_invalid: "La validación falló: {{errors}}"
      models:
      attributes:
  date:
    formats:
      default: "%e/%m/%Y"
      short: "%e de %b"
      long: "%e de %B de %Y"

    day_names: [Domingo, Lunes, Martes, Miércoles, Jueves, Viernes, Sábado]
    abbr_day_names: [Dom, Lun, Mar, Mie, Jue, Vie, Sab]
    month_names: [~, Enero, Febrero, Marzo, Abril, Mayo, Junio, Julio, Agosto, Septiembre, Octubre, Noviembre, Diciembre]
    abbr_month_names: [~, Ene, Feb, Mar, Abr, May, Jun, Jul, Ago, Sep, Oct, Nov, Dic]
    order:
[ :day, :month, :year ]
  time:
    formats:
      default: "%A, %e de %B de %Y %H:%M:%S %z"
      short: "%e de %b %H:%M"
      long: "%e de %B de %Y %H:%M"
    am: "am"
    pm: "pm"
  support:
    select:
      prompt: "Por favor seleccione"
    array:
      words_connector: ", "
      two_words_connector: " y "
      last_word_connector: " y "
```

Capítulos propuestos por los descarrilados

AJAX

Los cambios

Las pruebas

GIT

Respaldo

Importación

Informes

Exportación

Publicación

SOAP, XML, JSON, WebServices

Ruby

Los generadores

JRuby

Hosting