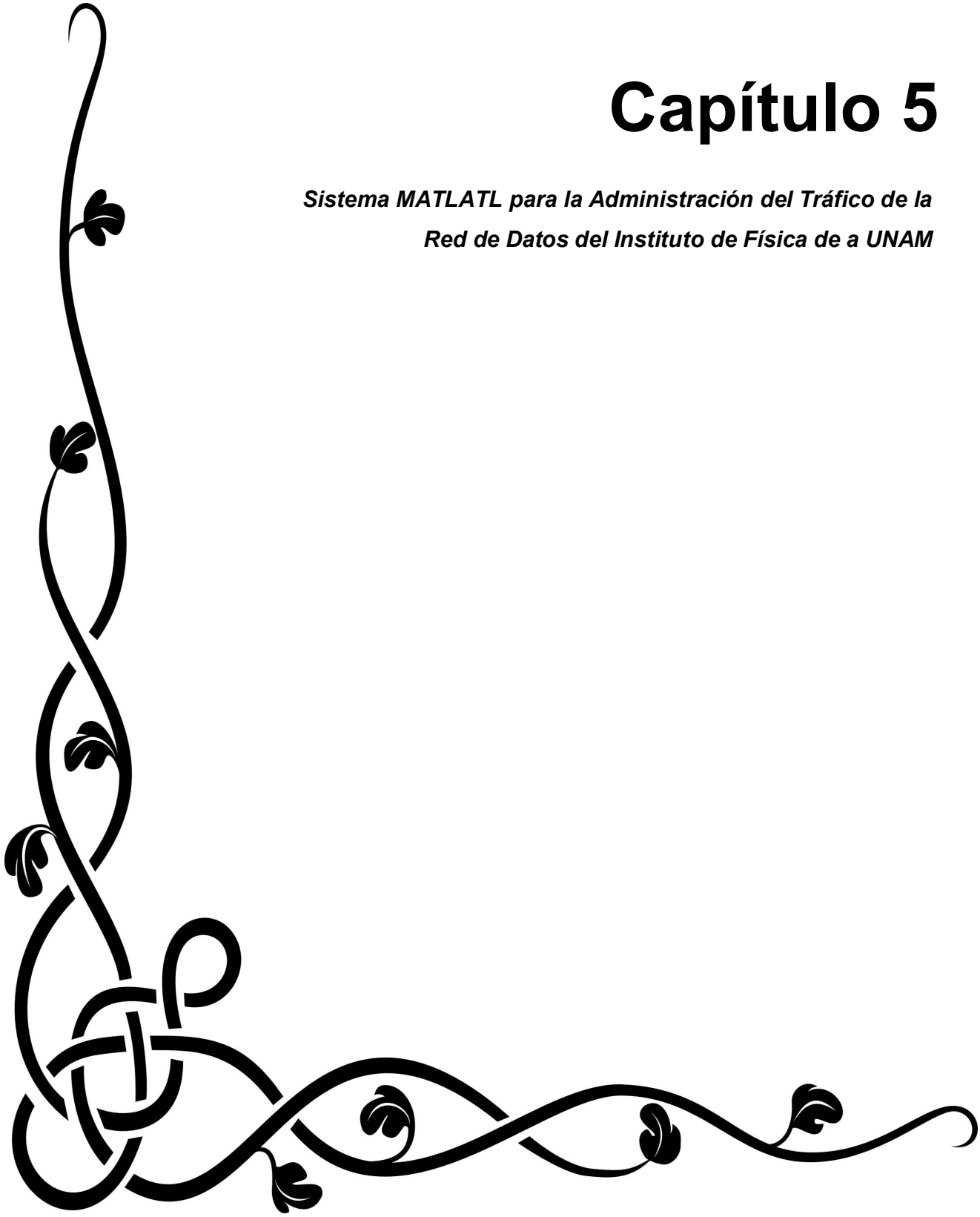

Capítulo 5

Sistema MATLATL para la Administración del Tráfico de la Red de Datos del Instituto de Física de a UNAM

"Hazlo todo tan simple como sea posible, pero no más simple."
Albert Einstein

Sistema MATLATL para la Administración del Tráfico de la Red de Datos del Instituto de Física de la UNAM

5.1 Selección de metodología para el desarrollo de sof tware

Al comenzar a desarrollar un software es importante tener en mente la metodología que se utilizará, ya que nos indicará el camino a seguir para la correcta aplicación de los elementos involucrados en el proyecto.

Los procesos utilizados en el desarrollo de un software hacen referencia a los paradigmas en el desarrollo del software, los cuales en la actualidad se dividen en dos ramas, aquellos que utilizan metodologías tradicionales y los que utilizan metodologías livianas, pero ¿cuál es el método correcto a seguir y porqué?

Ambos métodos tienen sus ventajas y desventajas pero nuestra elección dependerá de las necesidades que tengamos.

5.1.1 Métodos orientados al plan.

Estos métodos también son conocidos como metodologías tradicionales, las cuales son desarrolladas basándose en la teoría de la Ingeniería de Software, lo que nos proporciona técnicas que permiten la mejora de procesos para crear un software de calidad. Su objetivo principal es tener un demo funcional y sin errores al final de cada iteración, así como una documentación comprehensiva y muy completa.

Utilizar dichas metodologías implica seguir una secuencia de pasos por cada iteración (unidad de tiempo) programada, tales como:

- Planificación
- Análisis de requerimientos
- Diseño
- Codificación
- Revisión o pruebas de unidad
- Documentación

5.1.2 Métodos ágiles

Las metodologías ágiles, o también llamadas metodologías livianas, son aquellas enfocadas en el *desarrollo de software iterativo e incremental 28* y se caracterizan por estar enfocados en la gente y en los resultados. Es decir que se realiza un trabajo de colaboración entre desarrollador-cliente, y se le da más importancia a la comunicación cara a cara que a la documentación. Su principal objetivo es la entrega de software funcional dentro de un periodo corto de tiempo antes de continuar con la siguiente etapa.

Las metodologías ágiles tienen como principios básicos promover los siguientes valores:

- Desarrollo
- Simplicidad
- Trabajo en equipo
- Colaboración
- Dar la bienvenida a los cambios.

[28] Entregas pequeñas de software en cortos periodos de tiempo.

A pesar de que las metodologías livianas se utilizaban desde principios de los 90's, el término "ágil" fue formalmente establecido en febrero del 2001, durante una reunión realizada en Utah, EEUU en la que participaron 17 expertos de la industria del software. En dicha reunión se creó La Alianza Ágil (The Agil Alliance) y tuvo como consecuencia el origen del *Manifiesto Ágil* [29], documento que resume el conjunto de ideologías de cada integrante.

Manifiesto para el desarrollo ágil de software:

Estamos descubriendo mejores formas de desarrollar software, haciéndolo y ayudando a otros a hacerlo. A través de este trabajo hemos aprendido a valorar:

- Al individuo y a las interacciones antes que a los procesos y herramientas.
- Al software funcional antes que a una buena documentación.
- A la colaboración con el cliente antes que a las negociaciones de contrato.
- Respuestas al cambio antes que seguir un plan.

5.1.3 Diferencias entre metodologías ágiles y tradicionales

Para facilitar el proceso de comparación se muestra a continuación la tabla donde se aprecian las direfencias existentes entre ambos métodos:

	Métodos ágiles	Métodos tradicionales
Enfoque	Adaptabilidad de los procesos	Planificación de los procesos
Tamaño del Proyecto	Pequeño	Grande
Administración	Descentralizada	Autócrata
Perspectivas al Cambio	Preparados para el cambio	Cierta resistencia la cambio
Cultura	Liderazgo-Colaboración Libertad para lograr objetivos	Comando-Control Roles y tareas definidas
Documentación	Poca	En grandes cantidades
Énfasis	Orientado a la gente	Orientado al proceso
Ciclos	Numerosos	Limitados
Dominios	Impredecible/Exploratorio	Predecibles
Planificación	Mínima	La evolución del proyecto está basada en le planificación
Recuperación de la Inversión	Temprano en el proyecto	Al finalizar el proyecto
Tamaño del equipo	Pequeño/Creativo	Grande

Tabla 5.1 *Diferencias entre metodologías ágiles y metodologías tradicionales* [30]

[29] Fuente: Sitio web Agile Manifesto.
[30] Fuente: M. A. Awad "A comparision between Agil and Traditional Software Development Methodologies"

5.1.4 Razones para usar metodolgías ágiles

Las metodologías tradicionales han estado presentes por mucho tiempo, y han demostrado ser necesarias en proyectos de gran tamaño, pero en la actualidad el desarrollo de software ha evolucionado considerablemente, y el entorno tan cambiante de los sistemas impide que un enfoque "tradicional" y estricto en sus procesos sea flexible en la implementación de software de alta calidad en periodos cortos de tiempo.

Las metodologías ágiles surgen como una solución a este problema, ofreciendo la posibilidad de incorporar nuevas herramientas y buenas prácticas para satisfacer al cliente, el cual es el objetivo de todo proyecto. Por tal motivo, y comparando las características de las metodologías antes mencionadas, se llegó a la conclusión de que los procesos ágiles se adecuaban a nuestras necesidades para desarrollar el Sistema Matlatl debido a las siguientes razones:

1. El grupo de trabajo era pequeño, estaba conformado por el programador y el administrador de red, por lo que era importante mantenerse en comunicación constante.
2. Se buscaba la solución para un problema en concreto, consultar y actualizar (vía web) la información de usuarios de la red de datos del Instituto de Física.
3. Se deseaba usar nueva tecnología que permitiera implementar aplicaciones enfocadas al manejo de información en tiempo real.
4. Las entregas de los avances debían ser constantes.
5. Debía existir la posibilidad de realizar cambios cuando fueran solicitados.

5.2 Ruby on Rails

Si bien es cierto que durante los últimos años Ruby on Rails ha adquirido cierta popularidad, la decisión de utilizar dicha plataforma se basó principalmente en su capacidad de desarrollar aplicaciones web de una forma más rápida, simple y flexible. Además de contar con librerías que proveen la independencia de la base de datos, permiten la transmisión de datos, operaciones básicas CRUD (create, read, update y delete), búsquedas avanzadas, relación

de modelos, etc. En pocas palabras, facilitan la conexión entre nuestra aplicación y la base de datos para el intercambio de información.

5.2.1 El lenguaje de programación Ruby

En 1995 el programador japonés Yukihiro "Matz" Matsumoto presentó públicamente la primera versión de Ruby, un lenguaje de programación dinámico y orientado a objetos, de código abierto y distribuido bajo la licencia GPL. Su sintaxis es una combinación de lenguajes como Perl, Smalltalk, Eiffel, Ada y Lisp, lo que facilita el uso de *programación funcional 31* y *programación imperativa 32*.

Yukihiro ha comentado que "Ruby está diseñado para la productividad y diversión del desarrollador, siguiendo los principios de una buena interfaz de usuario. El diseño de sistemas necesita enfatizar las necesidades humanas más que en las de las máquinas."

Las ventajas más destacables de Ruby son:

- Todo es considerado un objeto.
- Su sintaxis es natural, clara y sencilla.
- El código es más sencillo de entender por lo que se reduce la probabilidad de errores.
- Es portable ya que puede correr en la mayoría de plataformas tales como GNU/Linux, Unix, Mac OS X, Windows 95/98/Me/2000/XP, etc.
- No es necesario declarar las variables y éstas siempre hacen referencias a objetos, no son los objetos mismos.
- Soporta herencia simple para extender su funcionalidad y evitar duplicidad de código.
- Buena funcionalidad al manejar expresiones regulares.

Desventajas:

- El tiempo de ejecución es más lento ya que cada instrucción debe ser interpretada al momento de ejecutarse.

[31] Programación basada en el uso de funciones aritméticas (Haskell, Miranda, Lisp, Scheme, ML, Ocaml, SAP).
[32] Programación basada en un conjunto de instrucciones que indican al equipo los pasos a realizar para conseguir un objetivo (ASP, Basic, C, Fortran, Pascal, Perl, PHP, Java).

5.2.2 El framework Rails

Entorno de programación (framework) diseñado para facilitar el desarrollo de aplicaciones web usando Ruby. Creado por el programador danés David Heinemeier Hansson y liberado en 2004 como software de código libre.

Las aplicaciones de Rails están basadas en la arquitectura Modelo-Vista-Controlador (MVC), ver Figura 5.1, la cual es considerada como la columna central de todo proyecto y que separa los datos, la interfaz de usuario y la lógica de control en tres componentes:

Modelo: Responsable del *comportamiento*[33] y el uso de los *datos*[34] en la aplicación.
Vista: Responsable de generar la interfaz de usuario con los datos de la aplicación.
Controlador: Responde a las acciones del usuario e indica qué se debe hacer.

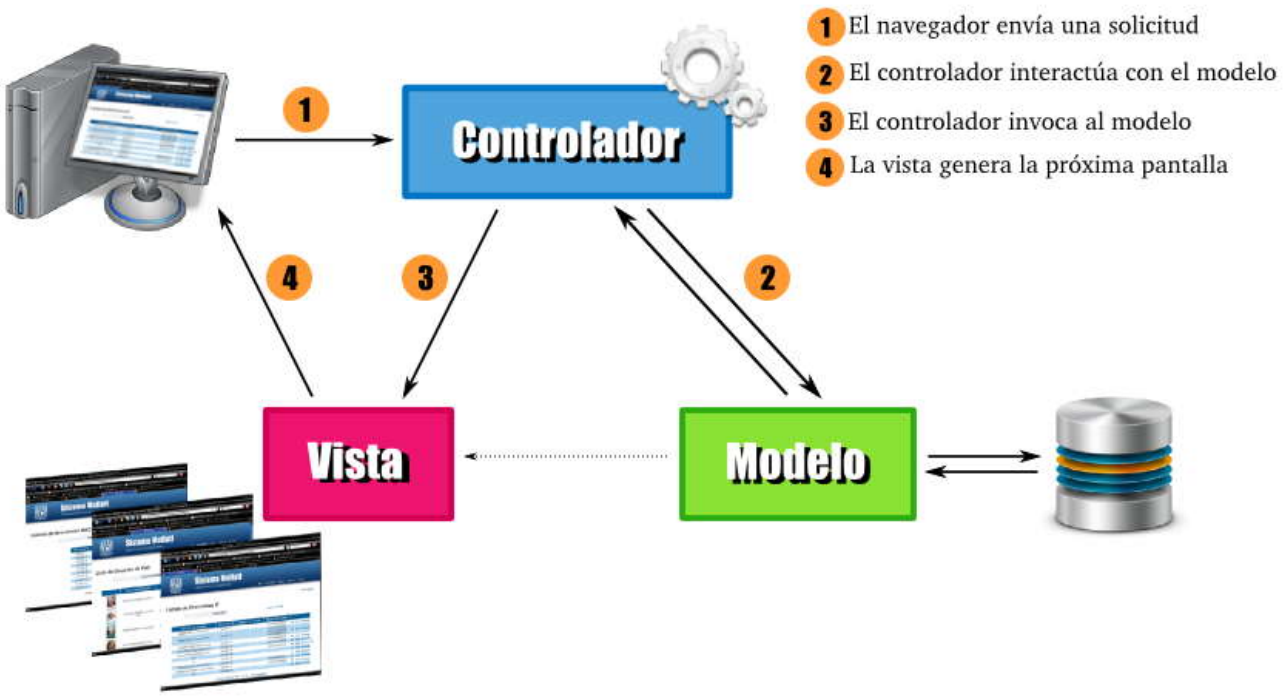

Figura 5.1 *Arquitectura Modelo-Vista-Controlador* [35]

[33] Responde a instrucciones del controlador.
[34] Responde a peticiones de la vista.
[35] Fuente: Sam Ruby, Dave Thomas "Agile Web Development with Rails"

El flujo de control del MVC que se muestra en la figura 5.1 es el siguiente: Se muestra una aplicación en el navegador y el usuario decide hacer una solicitud, por ejemplo dando click a un botón, el controlador interactúa con el modelo para notificar los datos solicitados por la acción del usuario y le encarga a la vista la tarea de mostrar los datos, la vista a su vez obtiene los datos del modelo y así puede generar la próxima pantalla.

Algo que caracteriza a Rails de otros entornos de trabajo son los principios de desarrollo en los que está basado:

DRY - Don't Repeat Yourself (No te repitas).

Cada definición debe realizarse una sola vez, es decir, especificas lo que necesitas en un solo lugar y sigues con el trabajo. Ruby y la arquitectura MVC serán los encargados de interpretar estas definiciones.

Convención sobre configuración.

Al diseñar una aplicación con Rails se escribirá menos código si se siguen las convenciones, pero existe la posibilidad de configurar un nuevo comportamiento sin afectar el desempeño del sistema.

Las ventajas de Rails son:

- Es ágil, por lo que facilita la interacción con los desarrolladores.
- Si se utilizan las convenciones del framework, podemos ahorrarnos mucho trabajo.
- Se pueden desarrollar aplicaciones muy completas usando poco código.
- Utiliza un sistema de plugins para permitir modificaciones y ampliar los proyectos conforme lo vayan requiriendo.
- Utiliza la técnica Mapeo Objeto-Relacional (ORM - Object-Relational Mapping) lo que nos permite expresar fácilmente cada una de las relaciones ya que se trata a cada uno de los registros como objetos.

Desventajas:

- En México es difícil encontrar un *hosting* capaz de soportar aplicaciones de RoR.
- Rails consume mucha memoria.
- Poner en producción una aplicación es muy complicado.

Es importante mencionar que el presente trabajo no se ve directamente afectado por dichas desventajas ya que:

- El instituto cuenta con servidor propio, por lo que no existe la necesidad de contratar un *hosting* y tenemos la ventaja de poder instalar el software que se necesite.
- A pesar de consumir muchos recursos de memoria, el sistema difícilmente será sobresaturado con consultas ya que es para uso interno de la Secretaría Técnica de Cómputo, Telecomunicaciones y Fotografía del Instituto de Física de la UNAM, por lo que solo los encargados de las áreas de Red, Seguridad, Informática, Supercómputo y Desarrollo de Aplicaciones tendrán acceso a él (aproximadamente 10 usuarios), ningún estudiante o investigador del instituto tendrá permisos para ingresar en el sistema. Además el servidor cuenta con 2 GB en RAM.
- Hay programadores que consideran difícil poner en producción aplicaciones de RoR porque no están acostumbrados a trabajar con líneas de comandos y a modificar los archivos de configuración de las aplicaciones, en nuestro caso es diferente ya que se tiene experiencia trabajando en sistemas operativos que utilizan dichos recursos.

5.3 Instalación de Ruby on Rails (RoR)

Antes de iniciar cualquier instalación es recomendable actualizar el sistema operativo, teniendo los permisos de *root* ejecutamos en una terminal las siguientes instrucciones:

```
# apt-get update
# apt-get dist-upgrade
```

1. Instalamos la librería que nos permite compilar paquetes.

```
# apt-get install build-essential
```

2. Instalamos los paquetes necesarios para comenzar a usar RoR.

```
# apt-get install ruby ri rdoc ruby1.8-dev irb1.8 libdbi-perl libnet-daemon-perl libplrpc-perl libreadline-ruby1.8 libruby1.8 rdoc1.8 ri1.8 ruby1.8 irb libopenssl-ruby libopenssl-ruby1.8 libhtml-template-perl libreadline5 psmisc
```

3. RubyGems es un gestor de paquetes utilizado por Ruby para la instalación de gemas (gems), este sistema favorece la distribución de aplicaciones o librerías en un formato estándar, se utiliza un servidor para almacenar las gemas y poder descargarlas desde cualquier máquina que tenga instalado RubyGems. Se descargó la última versión y procedimos con su instalación.

```
# wget http://rubyforge.org/frs/download.php/69365/rubygems-1.3.6.tgz
# tar xzvf rubygems-1.3.6.tgz
# cd rubygems-1.3.6
# ruby setup.rb
```

4. Para evitar el tener que escribir la ubicación completa de los programas durante el desarrollo de un proyecto creamos enlaces simbólicos que apunten a éstos.

```
# ln -s /usr/bin/gem1.8 /usr/local/bin/gem
# ln -s /usr/bin/ruby1.8 /usr/local/bin/ruby
# ln -s /usr/bin/rdoc1.8 /usr/local/bin/rdoc
# ln -s /usr/bin/ri1.8 /usr/local/bin/ri
# ln -s /usr/bin/irb1.8 /usr/local/bin/irb
```

5. Apoyándonos en el gestor rubygems instalamos Rails.

```
# gem install rails
```

6. Se instaló SQLite, como herramienta de base de datos, la cual es una librería que nos ofrece las características necesarias para trabajar con una base de datos relacional y permitir consultas *SQL*. Creada por D. Richard Hipp y de dominio público. Es una herramienta ágil y robusta, ya que la definición de las tablas, los índices y los propios datos son guardados en un archivo local al que se accede mediante llamadas a subrutinas o funciones de la propia librería.

```
# apt-get install sqlite3 swig libsqlite3-ruby libsqlite3-ruby1.8 libsqlite3-dev
# gem install sqlite3-ruby
```

5.4 Implementación del Sistema Matlatl

Figura 5.2 *Nombre del Sistema*

El nombre del sistema (Figura 5.2) tiene su origen en la palabra náhuatl MATLATL, la cuál significa RED, y a pesar de que los antiguos aztecas designaban esta palabra para redes de pesca o caza, se usó este nombre no solo por el hecho de hacer una alusión a la conexión entre puntos, si no porque se realizaría una mezcla de nuestro pasado con el presente, nuestras raíces y cultura con la tecnología actual.

Técnicamente hablando, Matlatl es un sistema web creado para administrar la información de los usuarios de la red de datos del IFUNAM, así como gestionar la asignación de direcciones IP, y por seguridad, llevar un registro de las direcciones MAC asociadas a cada IP. Permite la creación, actualización y eliminación de datos en tiempo real y ofrece el apoyo visual (gráficas) para observar el tráfico de red por usuario. Desarrollado en su totalidad con software de distribución libre para uso exclusivo de la Secretaría Técnica de Cómputo, Telecomunicaciones y Fotografía del Instituto de Física.

Para comprender que es lo que se necesita en un sistema, se recomienda identificar los casos de uso, los cuales son una declaración sencilla de cómo un usuario utiliza el sistema.

5.4.1 Casos de uso de Matlatl

Se identifican 2 tipos de usuarios: el administrador y el usuario.

1. El Administrador utiliza Matlatl para consultar una lista de los usuarios de red, una lista de las direcciones ips asignadas y una lista de las direcciones macs registradas, además puede actualizar los datos o ingresar nuevos. También tiene la opción de registrar nuevos usuarios que puedan acceder al sistema.

2. El Usuario utiliza Matlatl para consultar y buscar las ips o macs asignadas a los usuarios de red.

Para ver el diagrama de flujo de páginas del sistema Matlatl observe la Figura 5.3

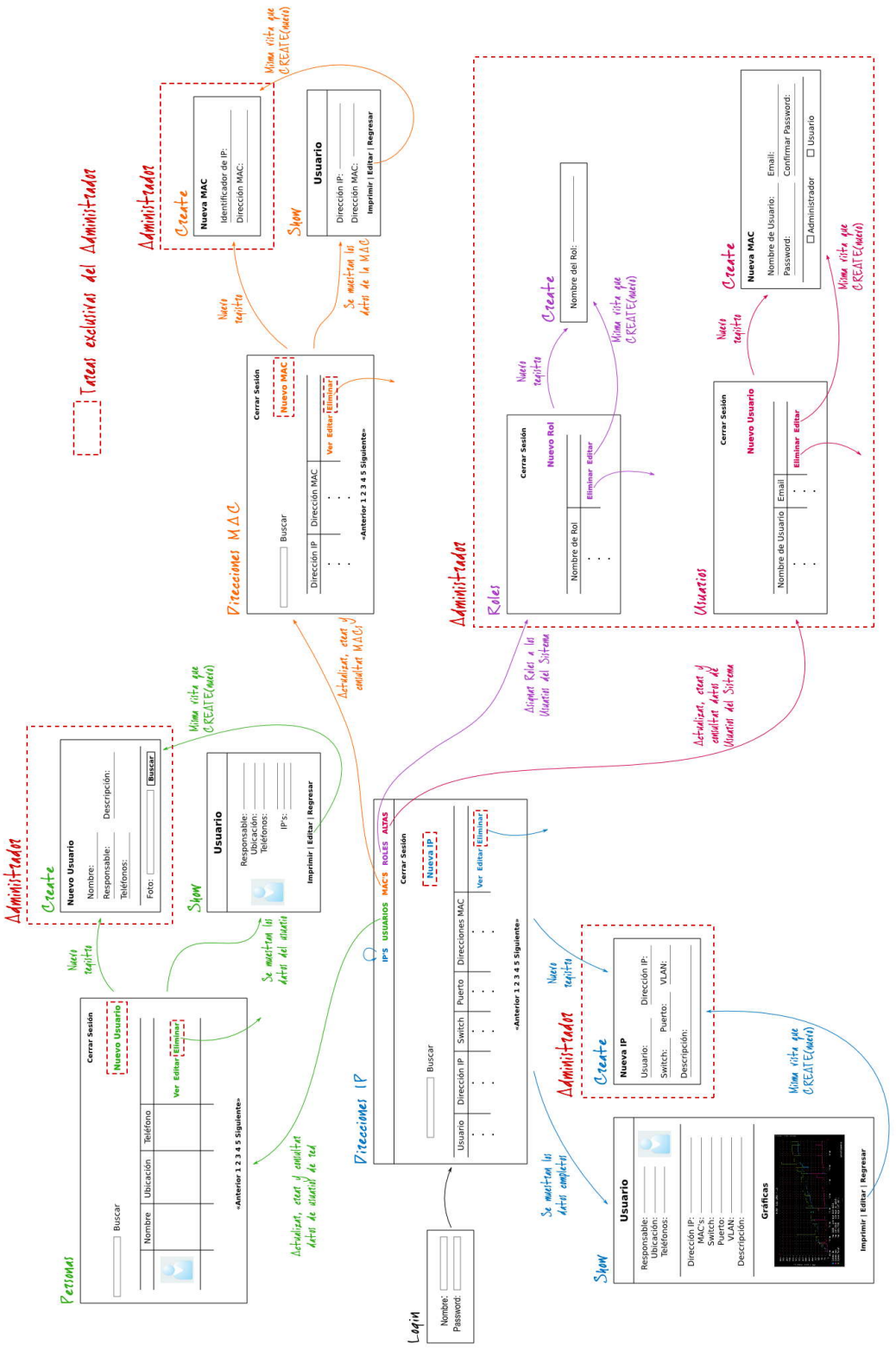

Figura5.3 *Diagrama de flujo de páginas del sistema Matlatl*

5.4.2 Creación de la Base de Datos.

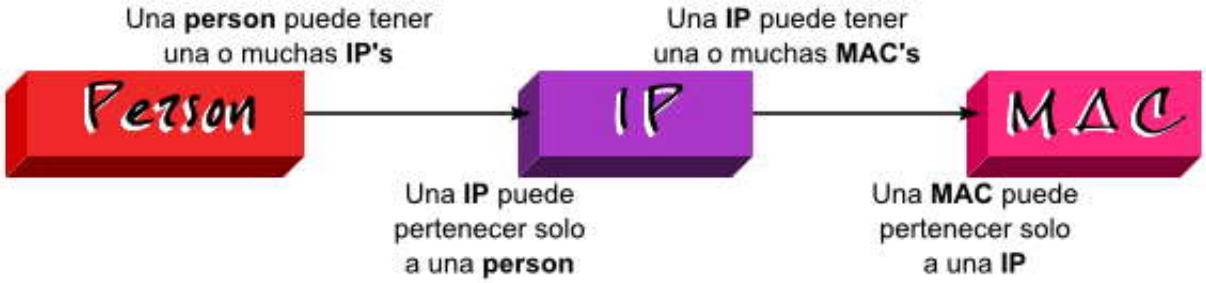

Figura 5.4 *Las 3 tablas principales de la Base de Datos*

1. Se creó un nuevo proyecto:

```
$ rails monitor_redes
```

2. Para generar la estructura principal del proyecto creamos los controladores, los modelos y las vistas utilizando la instrucción scaffold:

- Para *People*

```
$ ruby script/generate scaffold person person_name:string responsible:string location:text tel_person:string
```

- Para *IP's*

```
$ ruby script/generate scaffold ip person_id:integer ip_address:string port:integer switch:string vlan:string description:text
```

- Para *MAC's*

```
$ ruby script/generate scaffold mac ip_id:integer mac_address:string
```

3. Ejecutamos el comando de migración para crear las tablas:

```
$ rake db:migrate
```

4. Para indicar las relaciones existentes entre las clases, indicamos las asociaciones en cada uno de los modelos:

- En el modelo person.rb agregamos la siguiente línea:

```
has_many :ips    # una persona puede tener muchas ip's
```

- En el modelo ip.rb agregamos las siguientes líneas

```
belongs_to :person   # una ip pertenece a una sola persona
has_many :macs      # una ip puede tener muchas mac's
```

- En el modelo mac.rb agregamos la siguiente línea:

```
belongs_to :ip  # una mac pertenece a una sola ip
```

5.4.3 Presentación de los Datos.

Para presentar los datos con un formato estándar se realizaron los siguientes pasos:

1. Se creó una hoja de estilos y se ubicó en:

```
/public/stylesheets/estilo_redes.css
```

2. Se indicó en el *tag head* de cada layout la nueva hoja de estilos a utilizar, ya que RoR genera una predeterminada.

```
<%= stylesheet_link_tag ('estilo_redes', :media => :screen) %>
```

3. Para centrar la plantilla que se mostrará en el navegador web se usó el método de los *div's*, los cuales son declarados en la hoja de estilos, y se agregó a cada layout dentro del *tag body* las siguientes sentencias:

```
<div class="contenedor">
  <div class="contenido">

        Contenido de la página

  </div>
</div>
```

4. Para imprimir los datos de los usuarios usamos *javascript* con hojas de estilo, agregamos en el *tag head* de cada layout donde usemos esta función la siguiente instrucción:

```
<%= stylesheet_link_tag ('estilo_redes', :media => :print) %>
```

- Se agregó un link a la función “Imprimir” con la siguiente instrucción:

```
<%= link_to_function("Imprimir", "javascript:print()") %>
```

5.4.4 Instalación e implementación de Auto_Complete

Plugin oficial de Rails que sirve para desplegar información en un campo de texto, utiliza librerías de JavaScript para proporcionar efectos visuales en interfaces de usuario. Su principal uso es para autocompletar datos en un formulario.

Figura 5.5 *Ejemplo del plugin AutoComplete usado en el sistema*

La decisión de usar dicha extensión en nuestro sistema se debió a la dificultad que presentaba el tener que consultar los identificadores para poder relacionar datos en tiempo real, por ejemplo, para asociar una ip con un usuario era necesario verificar:

1. Si el usuario ya existía.
2. Si existía, era necesario consultar su id.
3. Si el usuario no existía debía crearse para poder tener un id.

4. Por último se relaciona el id del usuario con la ip.

Usando *auto_complete* en lugar de indicar el id nosotros comenzamos a escribir en el campo de texto, el *plugin* ejecuta una instrucción de consulta a la base de datos para presentarnos una lista de los nombres que coincidan con la secuencia de letras que vamos ingresando, al realizar nuestra selección guarda el id correspondiente al usuario elegido. De esta forma evitamos tener que navegar entre diferentes pantallas consultando datos.

Al configurarlo, el desarrollador debe indicar los datos que se deben mostrar y los datos que serán almacenados.

- Descargar el *plugin* del repositorio github.

```
$ wget http://github.com/rails/auto_complete/tree/master/rails-auto_complete.tar.gz
```

- Se descomprimió el paquete y se renombro la carpeta que se generó.

```
$ tar –xvf rails-auto_complete.tar.gz
$ mv –rf rails-auto_complete auto_complete
```

- Copiar la carpeta dentro de la ubicación /vendor/plugins/

```
$ cp –rf auto_complete/ monitor_redes/vendor/plugins/
```

Procedimiento para configurar auto_complete entre ip-person

- Se agregó en el archivo /app/views/layouts/ips.html.erb dentro de la etiqueta *head* la siguiente instrucción:

```
<%= javascript_include_tag :defaults %>
```

- Se buscó en el archivo /app/views/ips/new.html.erb la línea correspondiente a la instrucción del campo de texto para person_id

```
<%= f.text_field :person_id  %>
```

- Dicha instrucción se modificó para tener un campo de autocompletado de datos, además se indicó el parámetro que debe almacenar (*person_id*):

```
<%= text_field_with_auto_complete 'ip', 'person_id',{}, :skip_style => false  %>
```

- Dentro del controlador /app/controller/ips_controller.rb se agregó la siguiente instrucción:

```
skip_before_filter :verify_authenticity_token, :only => [:auto_complete_for_ip_person_id]
```

- En el mismo controlador se definió la función que permite realizar el autocompletado:

```
def auto_complete_for_ip_person_id()
  personname = params[:ip][:person_id]
  @people = Person.find(:all , :conditions=>"person_name like '%"+personname.downcase+"%'")
  render :partial => 'personname'
end
```

- Ubicamos en /app/views/ips/ el *partial* [36]_personname.html.erb:

```
<ul class="allusers">
  <% for person in @people do %>
    <li class="thisuser">
      <div class="useremail">
        <%=h person.id %>
      </div>

      <div class="username">
        <span class="informal">
          <%=h person.person_name %>
        </span>
      </div>
    </li>
  <% end %>
</ul>
```

Procedimiento para configurar auto_complete entre mac-ip

El procedimiento es similar al descrito con anterioridad, solo se cambiaron algunos parámetros.

- Se agregó en el archivo /app/views/layouts/macs.html.erb dentro de la etiqueta *head* la siguiente instrucción:

```
<%= javascript_include_tag :defaults %>
```

- Se modificó la instrucción del campo de texto en el archivo /app/views/macs/new.html.erb para tener un campo de autocompletado, se indicó el parámetro que debe almacenar (ip_*id*):

```
<%= text_field_with_auto_complete 'mac', 'ip_id',{}, :skip_style => false  %>
```

[36] Método utilizado para crear sub-vistas que requieran diferentes parámetros a los establecidos dentro de un controlador. También se usan cuando un mismo código será utilizado por diferentes vistas.

- Se agregó en el controlador /app/controller/macs_controller.rb la siguiente instrucción:

```
skip_before_filter :verify_authenticity_token, :only => [:auto_complete_for_mac_ip_id]
```

- En el mismo controlador se definió la función de autocompletado.

```
def auto_complete_for_mac_ip_id()
 ipaddress = params[:mac][:ip_id]
 @ips = Ip.find(:all , :conditions=>"ip_address like '%"+ipaddress.downcase+"%'")
 render :partial => 'ipaddress'
end
```

- Se creó el *partial* _ipaddress.html.erb ubicado en /app/views/macs/

```
<ul class="allusers">
 <% for ip in @ips do %>
  <li class="thisuser">
   <div class="useremail">
    <%=h ip.id %>
   </div>
   <div class="username">
    <span class="informal">
     <%=h ip.ip_address %>
    </span>
   </div>
  </li>
 <% end %>
</ul>
```

5.4.5 Instalación e implementación de Will_Paginate

Plugin para Rails que tiene como función paginar los resultados en la interfaz de usuario, es decir, que limita a un número preestablecido la cantidad de datos que se mostrarán por página (Figura 5.6).

Hasta este momento en el Instituto se cuentan con 510 direcciones IP, 207 usuarios de red y 410 direcciones MAC's registrados. Presentar dichos datos en una sola pantalla sería impráctico y poco presentable, por lo que se decidió usar un método de paginación para reducir la cantidad de datos que se presentaban durante una consulta.

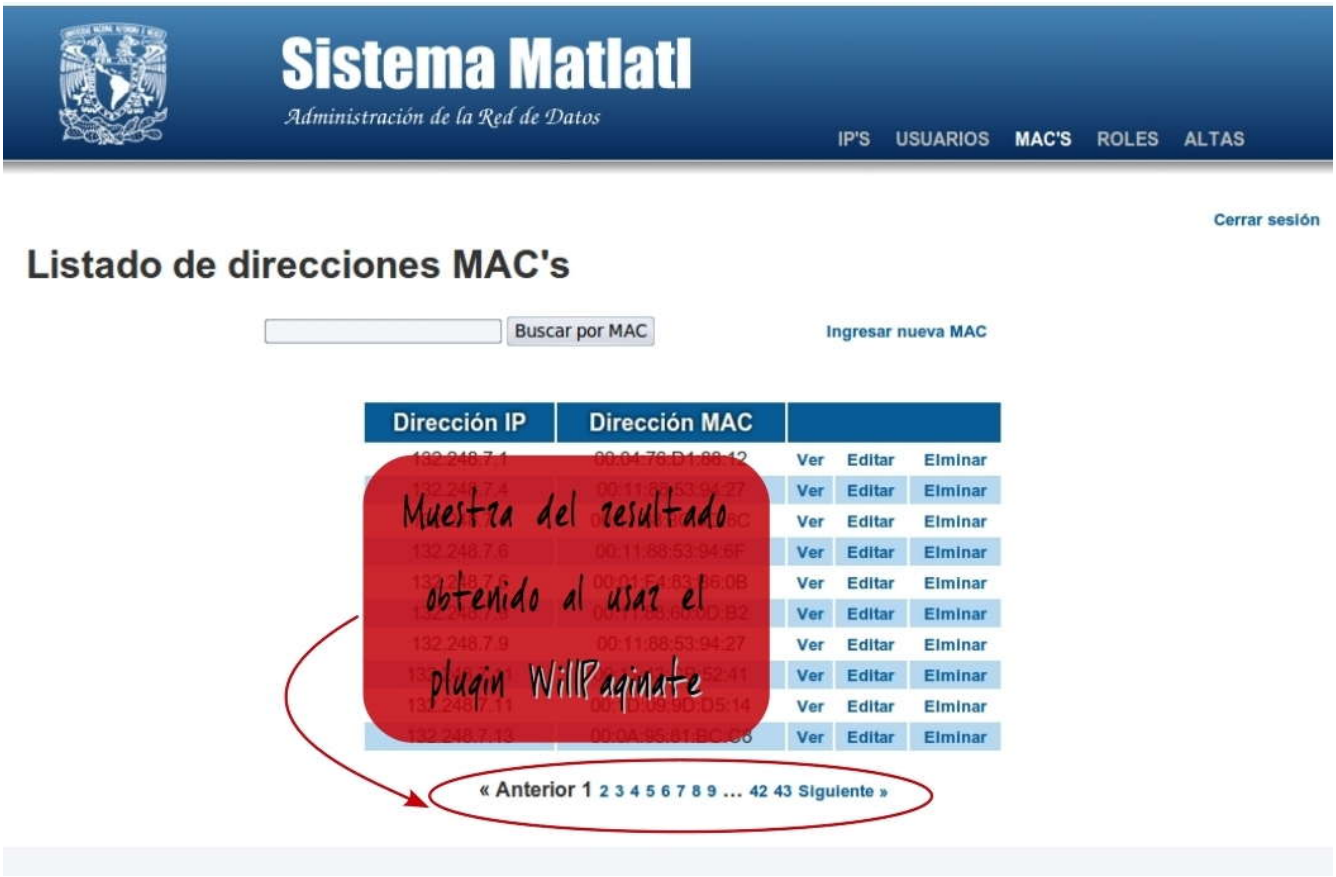

Figura 5.6 *Ejemplo del plugin WillPaginate usado en el sistema*

- Instalamos el código fuente de *GemCutter* para proporcionar nuevas fuentes de gemas:

```
$ sudo gem install gemcutter
```

- Las extensiones en RoR pueden ser instaladas como *plugin* o como gema, en este caso se instaló la gema *will_paginate* con la siguiente instrucción:

```
$ sudo gem install will_paginate
```

- Para habilitar la librería en el proyecto se agregó en el archivo /config/environment.rb el método de configuración de gemas:

```
require "will_paginate"
```

- Se agregó en cada controlador, dentro de la función index, la instrucción para indicar que se usará el método de paginación, además de establecer la cantidad de datos que se presentarán por página:

1. Para /app/controllers/people_controller.rb

   @people = Person.paginate(:page => params[:page], :per_page => 10)

2. Para /app/controllers/ips_controller.rb

   @ips = Ip.paginate(:page => params[:page], :per_page => 10)

3. Para /app/controllers/macs_controller.rb

   @macs = Mac.paginate(:page => params[:page], :per_page => 10)

- Por último se agregó a la interfaz de usuario los enlaces a las páginas para navegar entre ellas:

1. Para /app/views/people/index.html.erb

   <%= will_paginate @people, :prev_label => "« Anterior", :next_label => "Siguiente »" %>

2. Para /app/views/ips/index.html.erb

   <%= will_paginate @ips, :prev_label => "« Anterior", :next_label => "Siguiente »" %>

3. Para /app/views/macs/index.html.erb

   <%= will_paginate @macs, :prev_label => "« Anterior", :next_label => "Siguiente »" %>

5.4.6 Instalación e implementación de PaperClip

Plugin para Rails cuya principal función es adjuntar o almacenar archivos (Figura 5.7), tiene la capacidad de gestionar la presencia y el tamaño de ellos así como validar el tipo de archivo que se está cargando. No hay necesidad de crear nuevas tablas en la base de datos ya que se anexa como un campo en una ya existente.

Figura 5.7 *Ejemplo del plugin PaperClip usado en el sistema*

Para el procesamiento de imágenes se requiere la instalación de la librería *ImageMagick* [37] lo que nos facilita la creación de miniaturas.

En el sistema Matlatl se sube la imagen de los usuario, y de cada original se generan dos tamaños, una miniatura para presentar en la lista general y una mediana para la presentación de los datos generales.

- Instalamos la gema de Paperclip

```
$ sudo gem install paperclip
```

- Usando el método de configuración de gemas se agregó el archivo /config/environment.rb la siguiente línea:

[37] Software para editar, crear y componer imágenes.

```
config.gem 'paperclip', :source => 'http://gemcutter.org'
```

- Después se ejecutaron en la terminal la siguiente instrucción:

```
$ rake gem:install
```

- Al terminar la instalación de la gema se procedió a ejecutar su generador para agregar un campo de adjuntar documentos, en nuestro caso una imagen, al modelo Person:

```
$ ruby script/generate paperclip person photo
```

- Se creó la migración *20100205012445_add_attachments_photo_to_person.rb*, la cual generó automáticamente cuatro nuevos campos en el modelo, un campo para el nombre, un campo para le tipo de documento, un campo para el tamaño y un campo para la fecha de actualización:

```
add_column :people, :photo_file_name, :string
add_column :people, :photo_content_type, :string
add_column :people, :photo_file_size, :integer
add_column :people, :photo_updated_at, :datetime
```

- Para actualizar los cambios en la tabla people de la base de datos se ejecutó la siguiente instrucción:

```
$ rake db:migrate
```

- Se realizaron modificaciones al código del modelo para indicar que se adjuntaría un documento. Se agregó a /app/models/person.rb la siguiente línea:

```
has_attached_file :photo
```

- El código de las vistas también se vio modificado, por lo que en los archivos /app/views/people/new.html.erb y /app/views/people/edit.html.erb dentro de la sentencia form_for se agregaron los atributos que permiten la opción de adjuntar archivos:

```
<% form_for(@person, :html => { :multipart => true }) do |f| %>
```

- Se agregó la instrucción para le campo de búsqueda del archivo:

```
<%= f.file_field :photo %>
```

El método has_attached_file que se agregó en el modelo person.rb tiene diferentes opciones, una de ellas se llama style, lo que nos permite redimensionar el tamaño de las imágenes, en nuestro caso nos interesaba tener una vista en miniatura para presentar en el índice de

usuarios y otra mediana para presentar en los datos generales de cada usuario, por lo que se realizaron algunos cambios:

- Antes de continuar fue necesario instalar en el servidor la librería ImageMagic:

```
$ sudo apt-get install imagemagick
```

- En el archivo /app/models/person.rb se modificó la instrucción agregada con anterioridad:

```
has_attached_file :photo
```

- Se especificaron los atributos para redimensionar las imágenes:

```
has_attached_file :photo, :styles => { :medium => "160x160>", :small => "80x80>" }
```

- En la vista /app/views/people/index.html.erb usamos el estilo :small para el índice de usuarios:

```
<%= image_tag person.photo.url(:small) %>
```

- En la vista /app/views/ips/show.html.erb usamos el estilo :medium para mostrar los datos generales del usuario:

```
<%= image_tag @person.photo.url(:medium) %>
```

- Una opción más que nos ofrece *PaperClip* es la posibilidad de validar los archivos adjuntos, era importante especificar el tamaño límite y el tipo de archivo a usar, por lo que se agregó al modelo /app/models/person.rb las instrucciones que solo permiten archivos con extensiones .jpg o .png y que no rebasen un tamaño mayor a 1 MB:

```
validates_attachment_size :photo, :less_than => 1.megabytes
validates_attachment_content_type :photo, :content_type => ['image/jpeg', 'image/png']
```

5.4.7 Creación de una caja de búsqueda.

La secretaría técnica de cómputo del IFUNAM quería tener la posibilidad de realizar búsquedas dentro del sistema, para localizar direcciones MAC, direcciones IP y usuarios.

Para dicho propósito se generaron cajas de búsqueda en los tres modelos principales: *Person*, *Ip* y *Mac* (Figura 5.8). A continuación se explican los pasos que se siguieron para crearlas.

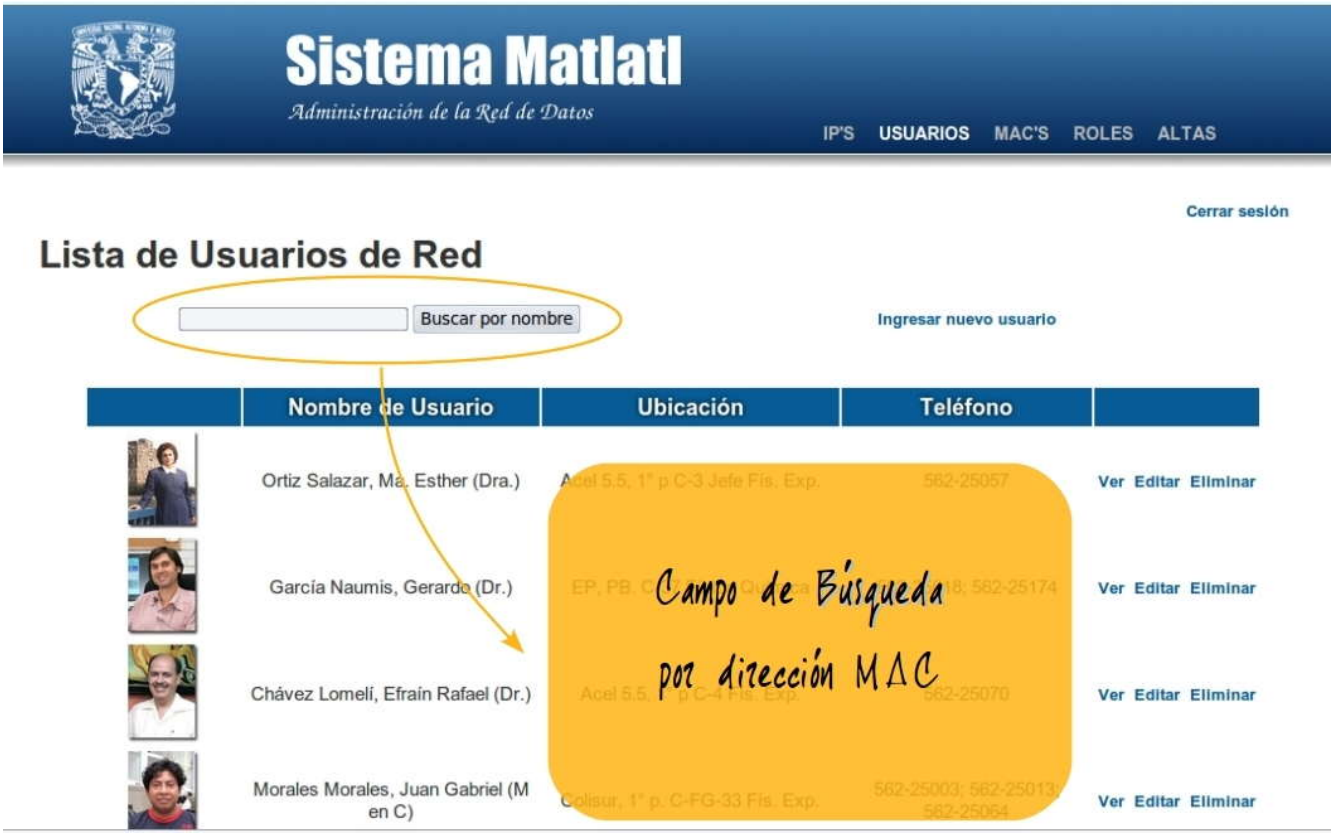

Figura 5.8 *Ejemplo de una caja de búsqueda usada en el sistema*

Forma de búsqueda por persona.

- Primero se modificó la instrucción de paginación ya que dicha sentencia cambia al usar formas de búsqueda, por lo que en el controlador /app/controllers/people_controller.rb se buscó la siguiente instrucción:

```
@people = Person.paginate(:page => params[:page], :per_page => 5)
```

- Y se realizó el cambio para juntar las funciones de búsqueda con paginación:

```
@people = Person.search(params[:search]).paginate :per_page => 5, :page => params[:page]
```

- En el modelo /app/models/person.rb se definieron los patrones de búsqueda agregando la siguiente función:

```
def self.search(search)
  if search
    find(:all, :conditions => ['person_name LIKE ?', "%#{search}%"])
  else  validates_presence_of :ip_id, :mac_address
    find(:all)
  end
end
```

Dicha función busca el nombre de la persona(s) que coincidan con los caracteres tecleados en el campo de texto, si no se ingresa ningún texto muestra los datos de todos los usuarios.

- Por último se agregó a la vista /app/views/people/index.html.erb la instrucción para mostrar el campo de texto y el botón de búsqueda.

```
<% form_tag people_path, :method => 'get' do %>
  <%= text_field_tag :search, params[:search] %>
  <%= submit_tag "Buscar por nombre", :person_name => nil %>
<% end %
```

Forma de búsqueda por dirección IP.

- Se indicó en el controlador /app/controllers/ips_controller.rb la instrucción que permite juntar las funciones de paginación y búsqueda:

```
@ips = Ip.search(params[:search]).paginate :per_page => 5, :page => params[:page]
```

- Se definieron los parámetros de búsqueda en el modelo /app/models/ip.rb

```
def self.search(search)
  if search
    find(:all, :conditions => ['ip_address LIKE ?', "%#{search}%"])
  else
    find(:all)
  end
end
```

- Se agregó a la vista /app/views/ips/index.html.erb la instrucción para mostrar el campo de texto y el botón de búsqueda.

```
<% form_tag ips_path, :method => 'get' do %>
  <%= text_field_tag :search, params[:search] %>
  <%= submit_tag "Buscar por IP", :ip_address => nil %>
<% end %>
```

Forma de búsqueda por dirección MAC.

- Se indicó en el controlador /app/controllers/macs_controller.rb la instrucción que une las funciones de paginación y búsqueda.

```
@macs = Mac.search(params[:search]).paginate :per_page => 5, :page => params[:page]]
```

- Se definieron los parámetros de búsqueda en el modelo /app/models/mac.rb

```
def self.search(search)
  if search
    find(:all, :conditions => ['mac_address LIKE ?', "%#{search}%"])
  else
    find(:all)
  end
end
```

- Se agregó a la vista /app/views/macs/index.html.erb la instrucción para mostrar el campo de texto y el botón de búsqueda.

```
<% form_tag ips_path, :method => 'get' do %>
  <%= text_field_tag :search, params[:search] %>
  <%= submit_tag "Buscar por IP", :ip_address => nil %>
<% end %>
```

Como se puede apreciar, el procedimiento para agregar la caja de búsqueda es el mismo en los tres modelos, la diferencia entre ellos radica en las variables y la definición de los parámetros.

5.4.8 Validaciones

Una de las grandes ventajas al desarrollar sistemas en RoR es que podemos implementar validaciones de una forma rápida y con poco código.

"La capa del modelo es el guardián entre el mundo del código y la base de datos" [38] por lo que es recomendable colocar las validaciones en el modelo, ya que nada que tenga que ver con nuestro sistema sale de la base de datos o se almacena en ella sin antes ser verificado por el modelo. Este método como medida de seguridad es perfecto para mantener la base de datos libre de datos basura además de prevenir al usuario de guardar datos corruptos.

Algunas de las validaciones más utilizadas son:

1. Validaciones de campos vacíos: validate_presence_of

[38] Fuente: Sam Ruby, Dave Thomas "Agile Web Development with Rails"

2. Validaciones de singularidad: validate_uniqueness_of
3. Validaciones de campos numéricos: validate_numericality_of
4. Validaciones de formatos para email o nombres: validate_format_of

En el presente trabajo se realizaron las validaciones para evitar que ciertos campos de texto estuvieran vacíos, ya que al guardar datos nulos se generaban errores que impedían acceder a la información. En la Figura 5.9 se muestra un ejemplo de las validaciones de datos ya que nos indica cuales son los datos que no pueden estar en blanco.

Figura 5.9 *Ejemplo de validación de datos usado en el sistema*

- Para comprobar la presencia del nombre de usuario y el responsable en el modelo /app/models/person.rb se agregó la siguiente validación:

```
validates_presence_of :person_name, :responsible
```

- Para comprobar la presencia del id del usuario y la dirección ip en modelo /app/models/ip.rb se agregó la siguiente validación:

```
validates_presence_of :person_id, :ip_address
```

- Para comprobar la presencia del id de la dirección ip y la dirección mac en el modelo /app/models/mac.rb se agregó la siguiente validación:

```
validates_presence_of :ip_id, :mac_address
```

5.4.9 Logeo y autenticación de usuarios.

Con el motivo de mantener un control en la edición y consulta de datos, se solicitó realizar una verificación de los usuarios que ingresen al sistema.

Instalación e implementación de Authlogic.

Plugin para Rails que ofrece una solución para autenticación de usuarios en un sistema web, de código simple, limpio y discreto. Se caracteriza por que es fácil de extender, ya que hay otros plugins que pueden agregarle funcionalidades.

Authlogic no genera vistas o controladores, se trabaja directamente con el código por lo que nos ofrece mayor flexibilidad y control en la función encargada de administrar la autenticación de usuarios.

A continuación se describe el procedimiento que se siguió para implementar dicho plugin en el sistema Matlatl.

- Se instaló el plugin como gema, por lo que indicamos en /config/environment.rb el método de configuración:

```
config.gem "authlogic"
```

- Ejecutamos en la terminal la instrucción:

```
rake gems:install
```

- Al terminar la instalación procedimos a crear el modelo y el controlador para administrar a los usuarios. Ubicándonos en la carpeta donde se almacena el proyecto se generó el modelo llamado User:

$ ruby script/generate model user

- Para configurar las propiedades del modelo editamos la migración generada, es importante mencionar que *Authlogic* trabaja buscando columnas con un determinado nombre dentro de la base de datos, por lo que éstos no son arbitrarios, son específicos para ser usados por el plugin y pueden consultarse en su documentación. A continuación se muestra el archivo de migración después de agregar los parámetros requeridos:

```
class CreateUsers < ActiveRecord::Migration
 def self.up
   create_table :users do |t|
     t.string :username
     t.string :email
     t.string :crypted_password
     t.string :password_salt
     t.string :persistence_token
     t.timestamps
   end
  end
  def self.down
   drop_table :users
  end
end
```

- Para crear la nueva tabla en la base de datos usamos la siguiente instrucción:

$ rake db:migrate

- Para que el modelo User trabaje con las funcionalidades de *Authlogic* es importante añadir en /app/models/user.rb la siguiente instrucción:

acts_as_authentic

- Se generaron el controlador users y las vistas index, new y edit:

$ ruby script/generate controller users new edit

- Dentro del controlador /app/controllers/users_controller.rb se definió la función que permite crear nuevos usuarios:

```
  def create
   @user = User.new(params[:user])
   if @user.save
     flash[:notice] = "Registro realizado"
     redirect_to :controller => "users", :action => "index"
```

```
  else
    render :action => 'new'
  end
 end
```

Si al ingresar los datos hay un error de validación (realizadas por *authlogic*), se mostrará nuevamente el formulario para nuevos usuarios, en caso contrario se creará el usuario y se almacenará en la base de datos.

- Debido a que el formulario para registrar o editar los datos del usuario es el mismo en ambos casos se creó el *partial* llamado _user.html.erb y éste es invocado desde las vistas /app/views/users/new.html.erb y /app/views/users/edit.html.erb con la instrucción:

```
<%= render @user %>
```

- A*uthlogic* proporciona un script generador llamado UserSession el cual nos permite crear un modelo para administrar toda la lógica de sesión:

```
$ ruby script/generate session user_session
```

- En este caso tampoco se generaron automáticamente el controlador y las vistas necesarias, por lo que creamos el controlador UserSessions y la vista new:

```
$ ruby script/generate controller UserSessions new
```

- En el controlador se definieron las funciones para crear, iniciar y terminar las sesiones:

```
 def new
  @user_session = UserSession.new
 end

 def create
  @user_session = UserSession.new(params[:user_session])
  if @user_session.save
    flash[:notice] = "Sesión Iniciada"
    redirect_to :controller => "ips", :action => "index"
  else
    render :action => 'new'
  end
 end

 def destroy
  @user_session = UserSession.find
  @user_session.destroy
  flash[:notice] = "Sesión Finalizada"
```

```
    redirect_to :controller => "user_sessions", :action => "new"
  end
```

- Se editó el archivo de rutas ubicado en /config/routes.rb para que el controlador sea considerado un recurso y presentar URL's limpias. El formulario de logeo se estableció como la página de inicio, si no se inicia sesión para autenticarse como usuario registrado no se podrá acceder al sistema:

```
map.root :controller => 'user_sessions', :action => 'new'
map.resources :user_sessions
map.login 'login', :controller => 'user_sessions', :action => 'new'
map.logout 'logout', :controller => 'user_sessions', :action => 'destroy'
```

- Para tener en todos los controladores el bloqueo de usuarios se definieron los siguientes métodos en el controlador global /app/controllers/application_controller.rb:

```
helper_method :current_user

def current_user_session
  return @current_user_session if defined?(@current_user_session)
  @current_user_session = UserSession.find
end

def current_user
  @current_user = current_user_session && current_user_session.record
end

def require_user
  unless current_user
    store_location
    flash[:notice] = "Es necesario iniciar sesión"
    redirect_to new_user_session_url
    return false
  end
end
def require_no_user
  if current_user
    store_location
    flash[:notice] = "Iniciar sesión"
    #redirect_to account_url
    return false
  end
end

def store_location
  session[:return_to] = request.request_uri
```

```
end

def redirect_back_or_default(default)
  redirect_to(session[:return_to] || default)
  session[:return_to] = nil
end
```

- Por ultimo para restringir el acceso a usuarios no registrados y hacer cumplir los métodos anteriores, fue necesario establecer un filtro para realizar la autenticación antes de ejecutar cualquier otra acción, se escribió la siguiente instrucción en los controladores que lo requerían:

```
before_filter :require_user
```

Instalación e implementación de Declarative_Authorization

Plugin para *Rails* que permite especificar autorizaciones basadas en roles. Debido a que las reglas de autorización se declaran en un archivo, el desarrollador debe indicar qué roles tienen permitido acceder a ciertos controladores.

Para el caso del sistema Matlatl se contaban con las siguientes características:
Se debían contar con dos modelos *Role* y *User*.
Un usuario puede tener asignado de uno a muchos roles.
Un rol puede tener asignado de uno a muchos usuarios.

Como se contaba con una relación muchos a muchos fue necesario crear un modelo de asignaciones (*Assignment*) para dividir esta relación. Ver Figura 5.10

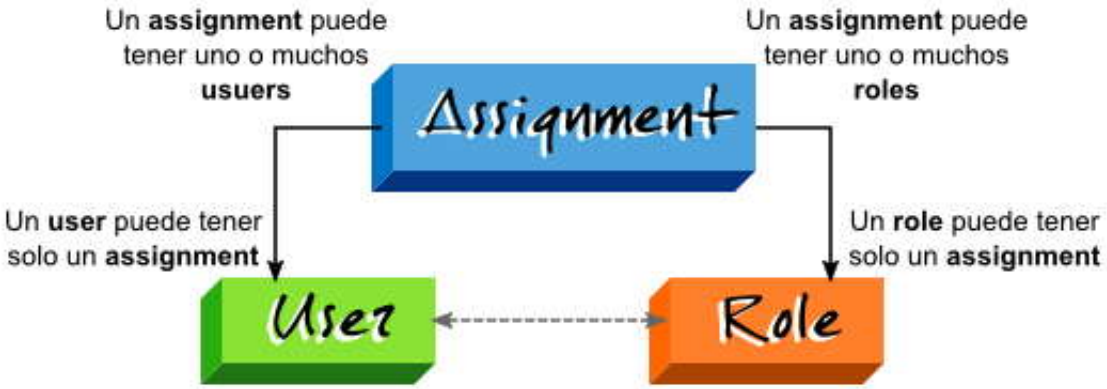

Figura 5.10 *Creación del modelo de Asignación para dividir relación muchos a muchos*

- Usando la instrucción scaffold se generó la vista, el controlador y el modelo de *Role*:

```
$ ruby script/generate scaffold model role name:string
```

- Se generó el modelo *Assignment*:

```
$ ruby script/generate model assignment user_id:integer role_id:integer
```

-Se actualizaron las tablas:

```
$ rake db:migrate
```

- Se editaron los modelos involucrados en la relación muchos a muchos para establecer sus asociaciones:

1. Para users.rb:

```
has_many :assignments
has_many :roles, :through => :assignments
```

2. Para assignment.rb:

```
belongs_to :role
belongs_to :user
```

3. Para role.rb:

```
has_many :assignments
has_many :users, :through => :assignments
```

- A partir de este momento será posible asociar un rol a cualquier usuario. Se procedió a

editar la vista /app/views/users/_user.html.erb para agregar el código correspondiente a un check_box_tag, el cual nos permitirá asignar un tipo de usuario a cada registro que se realice:

```
<% for role in Role.find(:all) %>
  <%= check_box_tag "user[role_ids][]", role.id, @user.roles.include?(role) %>
  <%= role.name %>
<% end %>
```

- Para permitir un array vacío antes de hacer las asignaciones y poder actualizar los datos se agregó al controlador /app/controllers/users_controller.rb, al principio de la acción *update*, la siguiente instrucción:

```
params[:user][:role_ids] ||= []
```

- Acontinuación se indicó en /config/environment.rb la instrucción de configurción de gemas para *declarative_authorization*:

```
config.gem "declarative_authorization", :source => "http://gemcutter.org"
```

- Ejecutamos en la terminal:

```
rake gems:install
```

Los usuarios que se crearon son Administrador y Usuario, ya que fueron los solicitados por la Secretaría Técnica de Cómputo del IFUNAM, y los permisos asignados a cada uno de ellos son:

1. Administrador: Crear, ver, editar y eliminar en los controladores *people*, *ips*, *macs*, *roles* y *users*, es decir que tendrá todos los permisos.
2. Usuario: Ver en los controladores *people*, *ips* y *macs*. Este tipo de usuario tiene permisos limitados, solo podrá consultar información.

- En la carpeta /config se creó el archivo llamado authorization_rules.rb donde se definieron los permisos para cada rol, a continuación se muestra el archivo final:

```
authorization do
  role :administrador do
    has_permission_on [:people, :ips, :macs, :roles, :users], :to => [:index, :show, :new, :create, :edit, :update,
    :destroy, :auto_complete_for_ip_person_id, :auto_complete_for_mac_ip_id]
  end
```

```
  role :usuario do
   has_permission_on [:people, :ips, :macs], :to => [:index, :show]
  end
end
```

- Para que *declarative_authorization* sea capaz de reconocer los permisos de los roles asignados es necesario crear un nuevo metodo llamado role_symbols en el modelo /app/models/user.rb:

```
def role_symbols
  roles.map do |role|
    role.name.underscore.to_sym
  end
end
```

- Se agregó un filtro en /app/controllers/application_controller.rb para poder identificar al usuario de la sesión:

```
helper :all
protect_from_forgery
before_filter { |c| Authorization.current_user = c.current_user }
```

- Se agregó en cada uno de los controladores un filtro para restringir el acceso:

```
filter_resource_access
```

- Al finalizar se descubrió que al usar *declarative_authorization* la gema *auto_complete* no funcionaba adecuadamente ya que no tenía permisos asignados, por lo que fue necesario denominarlas como una colección adicional al plugin y para ello se editaron los siguientes controladores:

1. Para /app/controllers/ips_controller.rb el filtro se definió:

   ```
   filter_resource_access :additional_collection => :auto_complete_for_ip_person_id
   ```
2. Para /app/controllers/macs_controller.rb el filtro se definió:

   ```
   filter_resource_access :additional_collection => :auto_complete_for_mac_ip_id
   ```

5.5 Liberar la versión de producción.

1. Instalamos el módulo Phusion Passenger, el cual permite poner en marcha las aplicaciones web desarrolladas en RoR.

- Primero instalamos las librerías requeridas:

```
# apt-get install libc6 libpcre3 libpcre3-dev libpcrecpp0 libssl0.9.8 libssl-dev zlib1g zlib1g-dev lsb-base
```

- Se creó un directorio para alojar nuestra aplicación:

```
# mkdir -p /var/www/monitor/
```

- Fué necesario cambiar de dueño y grupo al directorio con todo su contenido.

```
# chown -R www-data:www-data /var/www/monitor/
```

- Finalizamos instalando la gema de *passenger* y el módulo para el servidor web nginx.

```
# gem install passenger
# passenger-install-nginx-module
```

Al ejecutar el instalador de Nginx elegimos la primera opción, para permitir que Passenger realice el trabajo de instalación y conservamos el directorio de ubicación predeterminado /opt/nginx.

Nginx es un servidor *http* y *proxy* inverso de alto rendimiento y muy liviano, desarrollado por el Ruso Igor Sysoev y liberado en el 2004 con licencia *Open Source*. Nginx se ha dado a conocer por su estabilidad, su gran conjunto de características, por su configuración sencilla y su bajo consumo de recursos. En la actualidad alberga cerca del 6% de todos los dominios del mundo[39].

2. Fue necesario modificar el archivo de configuración predeterminado de nginx para agregar las opciones que requería el sistema. El archivo se encuentra ubicado en /opt/nginx/cong/nginx.conf y al realizar los cambios se obtuvo lo siguiente:

```
user  www-data www-data;
worker_processes  1;
```

[39] Fuente: Sitio Web Nginx.

```
events {
   worker_connections  1024;
}

http {
   passenger_root /usr/lib/ruby/gems/1.8/gems/passenger-2.2.10;
   passenger_ruby /usr/bin/ruby1.8;

   include       mime.types;
   default_type  application/octet-stream;
   sendfile                on;
   tcp_nopush     on;
   tcp_nodelay    on;
   gzip  on;

   keepalive_timeout  65;

   server {
      listen       80;
      server_name  localhost;
      root /var/www;
      passenger_enabled on;
      passenger_base_uri /matlatl;
      client_max_body_size 50M;

      if (-f $document_root/system/maintenance.html) {
        rewrite ^(.*)$ /system/maintenance.html break;
      }

      error_page   500 502 503 504  /50x.html;
      location = /50x.html {
         root   html;
      }
   }
}
```

- Se creó un enlace simbólico, al que llamamos matlatl, que apunta al directorio public/ de nuestra aplicación, para proporcionar un URI[40] (Uniform Resource Identifier – Identificador Uniforme de Recursos) a nuestro sistema web:

 # ln -s /var/www/monitor/public/ matlatl

3. Al instalar nginx no se crea el archivo init, por lo que se descargó el script de la página oficial, el cual funciona para iniciar, detener y reiniciar el servidor, pero principalmente para

[40] Un "fragmento" o cadena corta de caracteres que permite identificar una parte del recurso principal, es nuestro caso la dirección web del sistema.

asegurar que inicia automáticamente al arrancar el sistema operativo.

- Teniendo los permisos de root se creó en el directorio /etc/init.d/ el archivo llamado nginx y se copió el siguiente contenido:

```
#! /bin/sh

### BEGIN INIT INFO
# Provides:          nginx
# Required-Start:    $all
# Required-Stop:     $all
# Default-Start:     2 3 4 5
# Default-Stop:      0 1 6
# Short-Description: starts the nginx web server
# Description:       starts nginx using start-stop-daemon
### END INIT INFO

PATH=/opt/nginx/sbin:/sbin:/bin:/usr/sbin:/usr/bin
DAEMON=/opt/nginx/sbin/nginx
NAME=nginx
DESC=nginx

test -x $DAEMON || exit 0

# Include nginx defaults if available
if [ -f /etc/default/nginx ] ; then
        . /etc/default/nginx
fi

set -e

case "$1" in
  start)
        echo -n "Starting $DESC: "
        start-stop-daemon --start --quiet --pidfile /opt/nginx/logs/$NAME.pid \
                --exec $DAEMON -- $DAEMON_OPTS
        echo "$NAME."
        ;;
  stop)
        echo -n "Stopping $DESC: "
        start-stop-daemon --stop --quiet --pidfile /opt/nginx/logs/$NAME.pid \
                --exec $DAEMON
        echo "$NAME."
        ;;
  restart|force-reload)
        echo -n "Restarting $DESC: "
        start-stop-daemon --stop --quiet --pidfile \
                /opt/nginx/logs/$NAME.pid --exec $DAEMON
```

```
        sleep 1
        start-stop-daemon --start --quiet --pidfile \
                /opt/nginx/logs/$NAME.pid --exec $DAEMON -- $DAEMON_OPTS
        echo "$NAME."
        ;;
  reload)
        echo -n "Reloading $DESC configuration: "
        start-stop-daemon --stop --signal HUP --quiet --pidfile     /opt/nginx/logs/$NAME.pid \
            --exec $DAEMON
        echo "$NAME."
        ;;
    *)
          N=/etc/init.d/$NAME
          echo "Usage: $N {start|stop|restart|reload|force-reload}" >&2
          exit 1
          ;;
  esac

  exit 0
```

- Se cambiaron los permisos del archivo y se indicó que debía arrancar al momento de iniciar el sistema:

 # sudo chmod +x /etc/init.d/nginx
 # sudo /usr/sbin/update-rc.d -f nginx defaults

A partir de este momento el sistema se encuentra liberado y es posible realizar consultas vía web.