Apuntes de la Asignatura Sistemas y Tecnologías Web

Casiano R. León [1]

24 de abril de 2013

[1]DEIOC Universidad de La Laguna

A Juana

For it is in teaching that we learn
And it is in understanding that we are understood

Agradecimientos/Acknowledgments

I'd like to thank

Parte I

SINATRA

Capítulo 1

Puesta en Escena

1.1. Routing

1.2. Acceso al Objeto Request

El objeto que representa la solicitud *the request object* es un hash con información de la solicitud: quien hizo la petición, que versión de HTTP usar, etc.

El objeto que representa la solicitud puede ser accedido desde el nivel de solicitud: filtros, rutas y manejadores de error.

Véase https://github.com/crguezl/sinatra_intro/blob/master/accesing_the_request_object.rb

1.3. Sinatra Authentication

1.3.1. Referencias

1. Ejemplo de uso de sinatra-authentication

1.4. Caches

1. Caching Tutorial

1.5. Práctica: Servicio de Syntax Highlighting

Construya una aplicación que provee syntax higlighting para un código que se vuelca en un formulario. Use la gema syntaxi.

El siguiente ejemplo muestra como funciona la gema `syntaxi`:

```
[~/rubytesting/syntax_highlighting]$ cat ex_syntaxi.rb
require 'syntaxi'
text = <<"EOF"
[code lang="ruby"]
  def foo
    puts 'bar'
  end
[/code]
EOF
formatted_text = Syntaxi.new(text).process
puts formatted_text
```

Ejecución:

```
[~/rubytesting/syntax_highlighting]$ ruby ex_syntaxi.rb
<pre>
<code>
<span class="line_number">1</span> <span class="keyword">def </span><span class="method">foo</
<span class="line_number">2</span> <span class="ident">puts</span>
<span class="punct">'</span><span class="string">bar</span><span class="punct">'</span>
<span class="line_number">3</span> <span class="keyword">end</span>
</code>
</pre>
```

La gema `syntaxi` usa la gema `syntax`:

```
[~/rubytesting/syntax_highlighting]$ gem which syntaxi/Users/casiano/.rvm/gems/ruby-1.9.2-head
[~/rubytesting/syntax_highlighting]$ grep "require.*'" /Users/casiano/.rvm/gems/ruby-1.9.2-hea
require 'syntax/convertors/html'
```

Es en esta gema que se definen las hojas de estilo:

```
[~/rubytesting/syntax_highlighting]$ gem which syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/lib/syntax.rb
[~/rubytesting/syntax_highlighting]$ tree /Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax
/Users/casiano/.rvm/gems/ruby-1.9.2-head/gems/syntax-1.0.0/
|-- data
|   |-- ruby.css
|   |-- xml.css
|   `-- yaml.css
|-- lib
|   |-- syntax
|   |   |-- common.rb
|   |   |-- convertors
|   |   |   |-- abstract.rb
|   |   |   `-- html.rb
|   |   |-- lang
|   |   |   |-- ruby.rb
|   |   |   |-- xml.rb
|   |   |   `-- yaml.rb
|   |   `-- version.rb
|   `-- syntax.rb
`-- test
    |-- ALL-TESTS.rb
    |-- syntax
    |   |-- tc_ruby.rb
    |   |-- tc_xml.rb
    |   |-- tc_yaml.rb
    |   `-- tokenizer_testcase.rb
    `-- tc_syntax.rb

7 directories, 17 files
```

En el esquema incompleto que sigue se ha hecho para el lenguaje Ruby. Añada que se pueda elegir el lenguaje a colorear (xml, yaml).

```
$ tree -A
.
|-- Gemfile
```

```
|-- Gemfile.lock
|-- toopaste.rb
‘-- views
    |-- layout.erb
    |-- new.erb
    ‘-- show.erb

$ cat Gemfile
source ’https://rubygems.org’

# Specify your gem’s dependencies in my-gem.gemspec
# gemspec
# gem ’guard’
# gem ’guard-rspec’
# gem ’guard-bundler’
# gem ’rb-fsevent’, ’~> 0.9.1’’

gem ’syntaxi’
```

Este es un fragmento de la aplicación:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat  toopaste.rb
require ’sinatra’
require ’syntaxi’

class String
  def formatted_body
    source = "[code lang=’ruby’]
                #{self}
              [/code]"
    html = Syntaxi.new(source).process
    %Q{
      <div class="syntax syntax_ruby">
        #{html}
      </div>
    }
  end
end

get ’/’ do
  erb :new
end

post ’/’ do
  .....
end
```

Una versión simple de lo que puede ser `new.erb`:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/new.erb
<div class="snippet">
  <form action="/" method="POST">
    <textarea name="body" id="body" rows="20"></textarea>
```

```
    <br/><input type="submit" value="Save"/>
  </form>
</div>
```

Véase la página HTML generada por el programa para la entrada `a = 5`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Toopaste!</title>
  <style>
    html {
      background-color: #eee;
    }
    .snippet {
      margin: 5px;
    }
    .snippet textarea, .snippet .sbody {
      border: 5px dotted #eee;
      padding: 5px;
      width: 700px;
      color: #fff;
      background-color: #333;
    }
    .snippet textarea {
      padding: 20px;
    }
    .snippet input, .snippet .sdate {
      margin-top: 5px;
    }

    /* Syntax highlighting */
    #content .syntax_ruby .normal {}
    #content .syntax_ruby .comment { color: #CCC; font-style: italic; border: none; margin: no
    #content .syntax_ruby .keyword { color: #C60; font-weight: bold; }
    #content .syntax_ruby .method { color: #9FF; }
    #content .syntax_ruby .class { color: #074; }
    #content .syntax_ruby .module { color: #050; }
    #content .syntax_ruby .punct { color: #0D0; font-weight: bold; }
    #content .syntax_ruby .symbol { color: #099; }
    #content .syntax_ruby .string { color: #C03; }
    #content .syntax_ruby .char { color: #F07; }
    #content .syntax_ruby .ident { color: #0D0; }
    #content .syntax_ruby .constant { color: #07F; }
    #content .syntax_ruby .regex { color: #B66; }
    #content .syntax_ruby .number { color: #FF0; }
    #content .syntax_ruby .attribute { color: #7BB; }
    #content .syntax_ruby .global { color: #7FB; }
    #content .syntax_ruby .expr { color: #909; }
    #content .syntax_ruby .escape { color: #277; }
    #content .syntax {
      background-color: #333;
      padding: 2px;
      margin: 5px;
```

```
      margin-left: 1em;
      margin-bottom: 1em;
    }
    #content .syntax .line_number {
      text-align: right;
      font-family: monospace;
      padding-right: 1em;
      color: #999;
    }
  </style>
</head>
<body>
  <div class="snippet">
  <div class="snippet">
  <div class="sbody" id="content">
      <div class="syntax syntax_ruby">
        <pre>
           <code>
              <span class="line_number">1</span>
              <span class="ident">a</span>
              <span class="punct">=</span>
              <span class="number">5</span>
            </code>
         </pre>
      </div>
  </div>
  <br/><a href="/">New Paste!</a>
</div>
</body>
</html>
```

La gema

Una versión resumida de `layout.erb`:

```
[~/srcSTW/syntax_highlighting(withoutdm)]$ cat views/layout.erb
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DT
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title><%= @title || 'Toopaste!' %></title>
  <style>
    ..............
  </style>
</head>
<body>
  <%= yield %>
</body>
</html>
```

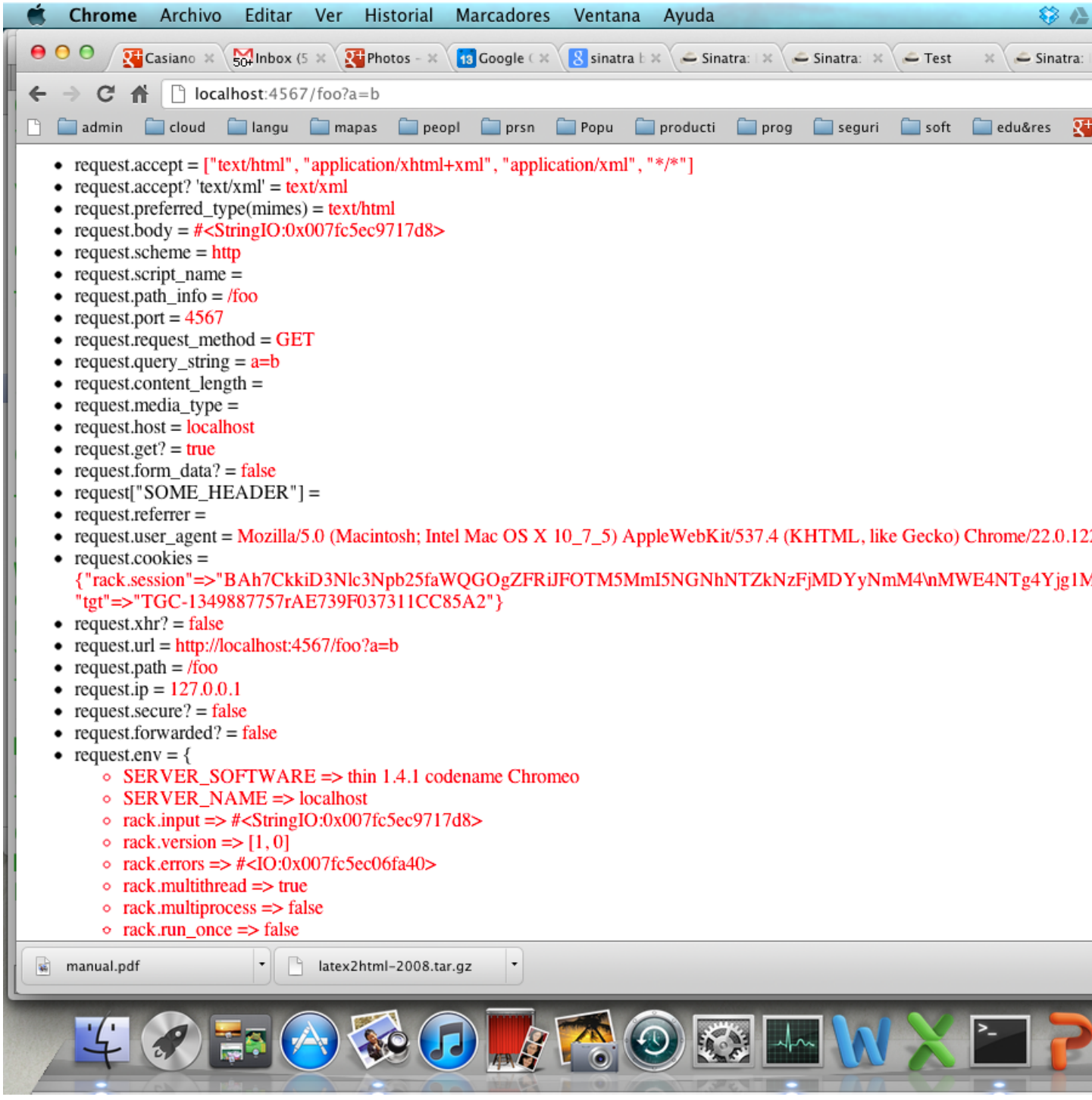

Figura 1.1: El Objeto Request

Capítulo 2

Fundamentos

2.1. Rutas

2.2. Ficheros Estáticos

2.3. Vistas

2.4. Manejo de Errores

2.5. Cabeceras HTTP

2.6. Caching

2.7. Cookies

2.8. Adjuntos

2.9. Streaming

2.9.1. Introducción

En Sinatra 1.3 se introdujo el stream helper. El stream object imita a un objeto IO.

En ocasiones queremos empezar a enviar datos mientras se esta generando aún el cuerpo de la respuesta. Puede que incluso queramos mantener el envío hasta que el cliente cierra la conexión.

Para eso podemos usar el `stream` helper:

```
get '/' do
  stream do |out|
    out << "It's gonna be legen -\n"
    sleep 0.5
    out << " (wait for it) \n"
    sleep 1
    out << "- dary!\n"
  end
end
```

Esto nos permite implementar

- streaming APIs,

- Server Sent Events[1]
- y que puede ser usada como base para WebSockets[2].
- Puede también ser utilizada para incrementar el throughput si parte pero no todo el contenido depende de un recurso que es lento.

Nótese que la conducta de streaming, en particular el número de solicitudes concurrentes, depende en gran medida del servidor web que sirve la aplicación.

- Algunos servidores como `Webrick` no soportan streaming. En tal caso el cuerpo es envíado en una tacada.
- `Shotgun` tampoco soporta streaming

`Sinatra::Streaming`

Sinatra 1.3 introduced the stream helper. The addon provided by the gem `sinatra-contrib` improves the streaming API by making the stream object immitate an `IO` object, making the body play nicer with middleware unaware of streaming.

This is useful when passing the stream object to a library expecting an IO or StringIO object.

```
06:31][~/srcSTW/streaming/upandrunning_streaming]$ cat -n simple.rb
     1  # http://www.sinatrarb.com/contrib/streaming.html
     2  # $ gem install sinatra-contrib
     3  require 'sinatra'
     4  require 'sinatra/streaming'
     5  set server: 'thin'
     6  #set server: 'unicorn'
     7
     8  get '/' do
     9    stream do |out|
    10      puts out.methods
    11      out.puts "Hello World!", "How are you?"
    12      out.write "Written #{out.pos} bytes so far!\n"
    13      out.putc(65) unless out.closed?
    14      out.flush
    15    end
    16  end
```

Manteniendo la Conexión Abierta: `keep_open`

- Si el parámetro opcional se pone a `keep_open`, `close` no será llamado al finalizar el bloque, permitiendo su cierre en un punto posterior del flujo de ejecución.
- Esto sólo funciona en *evented servers*[3], como Thin y Rainbows. Otros servidores cerrarán el stream:

[1] Server-sent events es una tecnología que proporciona notificaciones push desde el servidor a un navegador cliente en forma de eventos DOM

[2] WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor

[3] Broadly speaking, there are two ways to handle concurrent requests to a server. Threaded servers use multiple concurrently-executing threads that each handle one client request, while evented servers run a single event loop that handles events for all connected clients.

```
[07:15][~/sinatra/sinatraupandrunning/chapter2/streaming(master)]$ cat -n keep_open.rb
     1  require 'sinatra'
     2
     3  set :server, :thin
     4  set :connections, []
     5
     6  before do
     7   content_type :txt
     8  end
     9
    10  get '/' do
    11    # keep stream open
    12    stream(:keep_open) do |out|
    13      puts out.inspect
    14      puts out.class
    15      puts out.instance_variables
    16      puts out.class.class_variables
    17      puts out.methods(false)
    18      settings.connections << out
    19    end
    20  end
    21
    22  get '/b/:message' do |b|
    23    # write to all open streams
    24    settings.connections.each do |out|
    25      out << b
    26     end
    27    "message '#{b}' sent"
    28  end
```

Cuando se ejecuta nos informa sobre el objeto `out`:

```
[06:47][~/sinatra/sinatraupandrunning/chapter2/streaming(master)]$ ruby keep_open.rb
== Sinatra/1.3.3 has taken the stage on 4567 for development with backup from Thin
>> Thin web server (v1.3.1 codename Triple Espresso)
>> Maximum connections set to 1024
>> Listening on 0.0.0.0:4567, CTRL+C to stop
#<Sinatra::Helpers::Stream:0x007f9b78d09ee8 @keep_open=:keep_open,
   @scheduler=EventMachine,
   @back=#<Proc:0x007f9b78d09e48@/usr/local/rvm/gems/ruby-1.9.2-p290/gems/sinatra-1.3.3/lib/si
   @callbacks=[#<Proc:0x007f9b78c4ab88@/usr/local/rvm/gems/ruby-1.9.2-p290/gems/thin-1.3.1/lib
   #<Proc:0x007f9b78c4ab10@/usr/local/rvm/gems/ruby-1.9.2-p290/gems/thin-1.3.1/lib/thin/connec
   @closed=false,
   @front=#<Proc:0x007f9b78c4adb8@/usr/local/rvm/gems/ruby-1.9.2-p290/gems/thin-1.3.1/lib/thin
>
Sinatra::Helpers::Stream
@keep_open
@scheduler
@back
@callbacks
@closed
@front
^C>> Stopping ...
```

```
== Sinatra has ended his set (crowd applauds)
127.0.0.1 - - [15/Nov/2012 06:49:51] "GET / HTTP/1.1" 200 - 30.7024
```

Vemos que los objetos `out` están en la clase Sinatra::Helpers::Stream.

El siguiente ejemplo elabora el anterior y muestra como mantener una conexión persitente, abierta para enviar mensajes de broadcast a unos suscriptores:

```
casiano@nereida:~/srcSTW/streaming/upandrunning_streaming$ cat -n a_simple_straming_example.rb
 1  # coding: utf-8
 2  # page 46. Example 2-37.
 3  require 'sinatra'
 4
 5  before do
 6    content_type :txt
 7  end
 8
 9  set server: 'thin', connections: []
10
11  get '/consume' do
12    stream(:keep_open) do |out|
13      # store connection for later on
14      settings.connections << out
15
16      # remove connection when closed properly
17      out.callback { settings.connections.delete(out) }
18
19      # remove connection when due to an error
20      out.errback do
21        logger.warn 'we just lost the connection!'
22        settings.connections.delete(out)
23      end
24
25    end # stream
26  end
27
28
29  get '/broadcast/:message' do
30    settings.connections.each do |out|
31      out << "#{Time.now} -> #{params[:message]}" << "\n"
32    end
33
34    "Sent #{params[:message]} to all clients."
35  end
```

Para usar este ejemplo arrancamos el programa. Obsérvese que el servidor usado es `thin`. Después visitamos con uno o mas navegadores la página `localhost:4567/consume`. El navegador queda a la espera del servidor que ejecuta el bloque en las líneas 12-25. la página que será enviada en streaming. En una terminal usando `curl` nos conectamos a la URL `http://localhost:4567/broadcast/choco`. Esto hace que se ejecute el bloque de las líneas 30-31 y que un mensaje con el formato `fecha -> choco` se envíe al objeto stream `out`. En el navegador aparecerá el mensaje `Sent choco to all clients`.

2.9.2. Streaming y Valores de Retorno

El valor de retorno del bloque de una ruta determina el cuerpo de respuesta pasado al cliente HTTP, o al menos al siguiente middleware en la pila de Rack. Lo habitual es que sea una `String` pero

puede ser otra cosa.

De hecho, podemos retornar cualquier tipo de objeto que sea una respuesta Rack válida, un objeto cuerpo Rack o un código de estatus HTTP:

- Un `Array` con tres elementos: `[status (Fixnum), headers (Hash), response body (responds to #eac`
- Un `Array` con dos elementos: `[status (Fixnum), response body (responds to #each)]`
- Un objeto que responde a `#each` y que le pasa strings al bloque dado
- Un `Fixnum` que representa el status code

Así podemos, implementar el siguiente ejemplo que funciona en streaming:

```
[~/sinatra/intro(master)]$ cat stream.rb
require 'sinatra'

before do
  content_type :txt
end

class Flujo
  def each
    100.times { |i| yield "#{i}\n"}
  end
end

get '/' do
  Flujo.new
end
```

Enlaces Relacionados

- Streaming Responses at Sinatra Intro
- Sinatra::Streaming
- A simple demonstration of streaming Redis pub/sub data over HTTP via Sinatra's streaming capabilities.

2.9.3. Chat Utilizando Streaming y Server Sent Events (SSE)

```
[~/srcSTW/streaming/chat_with_streaming(master)]$ cat -n chat.rb
 1  # coding: utf-8
 2  require 'sinatra'
 3  set server: 'thin', connections: []
 4
 5  get '/' do
 6    halt erb(:login) unless params[:user]
 7    erb :chat, locals: { user: params[:user].gsub(/\W/, '') }
 8  end
 9
10  get '/stream', provides: 'text/event-stream' do
11    stream :keep_open do |out|
12      settings.connections << out
13      out.callback { settings.connections.delete(out) }
14    end
15  end
```

Routes may include a variety of matching conditions, such as the user `agent:`, `:host_name` and `:provides`:

```
get '/', :provides => ['rss', 'atom', 'xml'] do
  builder :feed
end
```

Sending an event stream from the source is a matter of constructing a plaintext response, served with a `text/event-stream` Content-Type, that follows the Server Sent Event (SSE) format.

Sigamos:

```
16
17  post '/' do
18    settings.connections.each { |out| out << "data: #{params[:msg]}\n\n" }
19    204 # response without entity body
20  end
```

204 No Content

The server has fulfilled the request but does not need to return an entity-body, and might want to return updated metainformation.

If the client is a user agent, it SHOULD NOT change its document view from that which caused the request to be sent. This response is primarily intended to allow input for actions to take place without causing a change to the user agent's active document view, although any new or updated metainformation SHOULD be applied to the document currently in the user agent's active view.

The 204 response MUST NOT include a message-body, and thus is always terminated by the first empty line after the header fields.

```
21
22  __END__
23
24  @@ layout
25  <html>
26    <head>
27      <title>Super Simple Chat with Sinatra</title>
28      <meta charset="utf-8" />
29      <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
30    </head>
31    <body><%= yield %></body>
32  </html>
```

The `<script>` tag is used to define a client-side script, such as a JavaScript.

The `<script>` element either contains scripting statements, or it points to an external script file through the `src` attribute.

Common uses for JavaScript are image manipulation, form validation, and dynamic changes of content.

The Google Hosted Libraries is a content distribution network for the most popular, open-source JavaScript libraries. To add a library to your site, simply use `<script>` tags to include the library. See https://developers.google.com/speed/libraries/devguide:

```
jQuery
snippet: <script src="//ajax.googleapis.com/ajax/libs/jquery/1.8.3/jquery.min.js"></script>
site: http://jquery.com/
versions: 1.8.3, 1.8.2, 1.8.1, 1.8.0, ...
note: 1.2.5 and 1.2.4 are not hosted due to their short and unstable lives in the wild.
```

jQuery is a fast and concise JavaScript Library that simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

`jquery.min.js` is a minified version of the JQuery JavaScript library, which provides a number of basic functions for websites.

```
34  @@ login
35  <form action='/'>
36    <label for='user'>User Name:</label>
37    <input name='user' value='' />
38    <input type='submit' value="GO!" />
39  </form>
40
41  @@ chat
42  <pre id='chat'></pre>
43
44  <script>
45    // reading
46    var es = new EventSource('/stream');
47    es.onmessage = function(e) { $('#chat').append(e.data + "\n") };
48
49    // writing
50    $("form").live("submit", function(e) {
51      $.post('/', {msg: "<%= user %>: " + $('#msg').val()});
52      $('#msg').val(''); $('#msg').focus();
53      e.preventDefault();
54    });
55  </script>
56
57  <form>
58    <input id='msg' placeholder='type message here...' />
59  </form>
```

- *Server-Sent Events (SSE)* are a standard describing how servers can initiate data transmission towards clients once an initial client connection has been established.
- They are commonly used to send message updates or continuous data streams to a browser client and designed to enhance native, cross-browser streaming through a JavaScript API called EventSource, through which a client requests a particular URL in order to receive an event stream.
- The idea behind *SSE*s is natural: a web app "subscribes"to a stream of updates generated by a server and, whenever a new event occurs, a notification is sent to the client.
- When communicating using SSEs, a server can push data to your app whenever it wants, without the need to make an initial request. In other words, updates can be streamed from server to client as they happen.
- SSEs open a *single unidirectional channel* between server and client.
- A Ruby/EventMachine based server implementation for WebSocket and Server-Sent Events is provided by Cramp.
- The *EventSource* interface is used to manage *server-sent events*. The server-sent event API is contained in the EventSource interface.
- To open a connection to the server to begin receiving events from it, you create a new `EventSource` object, specifying the URI of a script that generates the events:

```
var es = new EventSource('/stream')
```

- Next, we set up a handler for the message event.

  When updates are pushed from the server, the `onmessage` handler fires and new data is available in its `e.data` property:

```
47    es.onmessage = function(e) { $('#chat').append(e.data + "\n") };
```

- The magical part is that whenever the connection is closed, the browser will automatically reconnect to the source after ~3 seconds. Your server implementation can even have control over this reconnection timeout.
- You can also call `addEventListener()` to listen for events just like any other event source:

```
source.addEventListener('message', function(e) {
  console.log(e.data);
}, false);

source.addEventListener('open', function(e) {
  // Connection was opened.
}, false);

source.addEventListener('error', function(e) {
  if (e.readyState == EventSource.CLOSED) {
    // Connection was closed.
  }
}, false);
```

- Código HTML generado:

```
<html>
  <head>
    <title>Super Simple Chat with Sinatra</title>
    <meta charset="utf-8" />
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
  </head>
  <body><pre id='chat'></pre>

<script>
  // reading
  var es = new EventSource('/stream');
  es.onmessage = function(e) { $('#chat').append(e.data + "\n") };

  // writing
  $("form").live("submit", function(e) {
    $.post('/', {msg: "casiano: " + $('#msg').val()});
    $('#msg').val(''); $('#msg').focus();
    e.preventDefault();
  });
</script>

<form>
  <input id='msg' placeholder='type message here...' />
</form></body>
</html>
```

- We start with the dollar sign and parentheses: `$("form")` to specify a selector. The dollar sign (`$`) is simply shorthand for jQuery. With `$("form")` we select all the elements with tag name `form`

- `.append( content [, content] )`

  - `content`: DOM element, HTML string, or jQuery object to insert at the end of each element in the set of matched elements.
  - `content`: One or more additional *DOM* elements, arrays of elements, HTML strings, or jQuery objects to insert at the end of each element in the set of matched elements.

- `.live( events, handler(eventObject) ) Returns: jQuery`

  Description: Attach an event handler for all elements which match the current selector, now and in the future.

  - `events`: A string containing a JavaScript event type, such as `click` or `keydown.` As of jQuery 1.4 the string can contain multiple, space-separated event types or custom event names.
  - `handler(eventObject)`: A function to execute at the time the event is triggered.

- `jQuery.post( url [, data] [, success(data, textStatus, jqXHR)] [, dataType] )`

  - `url`: A string containing the URL to which the request is sent.
  - `data`: A map or string that is sent to the server with the request.
  - `success(data, textStatus, jqXHR)`: A callback function that is executed if the request succeeds.
  - `dataType`: The type of data expected from the server. Default: Intelligent Guess (xml, json, script, text, html).

  This is an easy way to send a simple `POST` request to a server. It allows a single callback function to be specified that will be executed when the request is complete (and only if the response has a successful response code).

  `$.post()` returns the `XMLHttpRequest` that it creates. In most cases you won't need that object to manipulate directly, but it is available if you need to abort the request manually.

- `.val`

  Get the input value of the first matched element.

  A value is returned for all input elements, including selects and textareas. For multiple selects an array of values is returned.

  For example:

```
$('select.foo option:selected').val();    // get the value from a dropdown select
$('select.foo').val();                    // get the value from a dropdown select even eas
$('input:checkbox:checked').val();        // get the value from a checked checkbox
$('input:radio[name=bar]:checked').val(); // get the value from a set of radio buttons
```

- `.focus( handler(eventObject) )`
  `.focus( [eventData], handler(eventObject) )`
  `.focus()`

  `handler(eventObject)`: A function to execute each time the event is triggered. `eventData`: A map of data that will be passed to the event handler.

This method is a shortcut for `.trigger('focus')`. The focus event is sent to an element when it gains focus. This event is implicitly applicable to a limited set of elements, such as form elements (`<input>`, `<select>`, etc.) and links (`<a href>`). In recent browser versions, the event can be extended to include all element types by explicitly setting the element's tabindex property. An element can gain focus via keyboard commands, such as the Tab key, or by mouse clicks on the element. Elements with focus are usually highlighted in some way by the browser, for example with a dotted line surrounding the element. The focus is used to determine which element is the first to receive keyboard-related events.

- `event.preventDefault() Returns: undefined`

  Description: If this method is called, the default action of the event will not be triggered.

  For example, clicked anchors will not take the browser to a new URL. We can use `event.isDefaultPrevented` to determine if this method has been called by an event handler that was triggered by this event.

Enlaces Relacionados

- Simple Chat Application using the Sinatra Streaming API
- Stream Updates With Server-Sent Events
- Using server-sent events
- EventSource
- Chat Server with server-sent Events
- A shared canvas where multiple clients can draw lines using EM-Websocket
- JQuery documentation
- What is the Document Object Model?
- JavaScript and HTML DOM Reference

Código Completo del Chat

```
[17:51][~/srcSTW/streaming/chat_with_streaming(master)]$ cat chat.rb
# coding: utf-8
require 'sinatra'
set server: 'thin', connections: []

get '/' do
  halt erb(:login) unless params[:user]
  erb :chat, locals: { user: params[:user].gsub(/\W/, '') }
end

get '/stream', provides: 'text/event-stream' do
  stream :keep_open do |out|
    settings.connections << out
    out.callback { settings.connections.delete(out) }
  end
end

post '/' do
  settings.connections.each { |out| out << "data: #{params[:msg]}\n\n" }
  204 # response without entity body
```

```
end

__END__

@@ layout
<html>
  <head>
    <title>Super Simple Chat with Sinatra</title>
    <meta charset="utf-8" />
    <script src="http://ajax.googleapis.com/ajax/libs/jquery/1/jquery.min.js"></script>
  </head>
  <body><%= yield %></body>
</html>

@@ login
<form action='/'>
  <label for='user'>User Name:</label>
  <input name='user' value='' />
  <input type='submit' value="GO!" />
</form>

@@ chat
<pre id='chat'></pre>

<script>
  // reading
  var es = new EventSource('/stream');
  es.onmessage = function(e) { $('#chat').append(e.data + "\n") };

  // writing
  $("form").live("submit", function(e) {
    $.post('/', {msg: "<%= user %>: " + $('#msg').val()});
    $('#msg').val(''); $('#msg').focus();
    e.preventDefault();
  });
</script>

<form>
  <input id='msg' placeholder='type message here...' />
</form>[
```

2.9.4. Práctica: Chat con Mensajes Individuales

Mejore el chat utilizando streaming y server sent events descrito en la sección 2.9.3 para que:

- Si el usuario escribe en su caja `/nickname: mensaje` ese `mensaje` sólo se entregue al usuario `nickname`

2.9.5. Práctica: Chat con Frames

Mejore la vista del chat utilizando streaming y server sent events descrito en la sección 2.9.3 para que:

- Usando frames, framesets iframes, tablas u otros mecanismos, haga que en un marco salgan los nicks de los usuarios conectados, en otro la conversación y en un tercero el texto que se va a

enviar.

2.9.6. WebSockets

Que es WebSocket y para que sirve

WebSocket es una tecnología que proporciona un canal de comunicación bidireccional y full-duplex sobre un único socket TCP. Está diseñada para ser implementada en navegadores y servidores web, pero puede utilizarse por cualquier aplicación cliente/servidor.

The *WebSocket* specification—developed as part of the HTML5 initiative—introduced the *WebSocket JavaScript interface*, which defines a full-duplex single socket connection over which messages can be sent between client and server.

The WebSocket standard simplifies much of the complexity around bi-directional web communication and connection management.

One of the more unique features WebSockets provide is its ability to traverse firewalls and proxies, a problem area for many applications.

Comet-style applications typically employ long-polling as a rudimentary line of defense against firewalls and proxies.

The technique is effective, but is not well suited for applications that have sub-500 millisecond latency or high throughput requirements.

Plugin-based technologies such as Adobe Flash, also provide some level of socket support, but have long been burdened with the very proxy and firewall traversal problems that WebSockets now resolve.

A WebSocket detects the presence of a proxy server and automatically sets up a tunnel to pass through the proxy.

The tunnel is established by issuing an HTTP CONNECT statement to the proxy server, which requests for the proxy server to open a TCP/IP connection to a specific host and port.

Once the tunnel is set up, communication can flow unimpeded through the proxy.

Since HTTP/S works in a similar fashion, secure WebSockets over SSL can leverage the same HTTP CONNECT technique.

Note that WebSockets are just beginning to be supported by modern browsers (Chrome now supports WebSockets natively).

However, backward-compatible implementations that enable today's browsers to take advantage of this emerging technology are available.

En el lado del cliente, WebSocket está ya implementado en Mozilla Firefox 8, Google Chrome 4 y Safari 5, así como la versión móvil de Safari en el iOS 4.2.1

Negociación del protocolo WebSocket

Para establecer una conexión WebSocket, el cliente manda una petición de negociación WebSocket, y el servidor manda una respuesta de negociación WebSocket, como se puede ver en el siguiente ejemplo:

```
Petición del navegador al servidor:
GET /demo HTTP/1.1
Host: example.com
Connection: Upgrade
Sec-WebSocket-Key2: 12998 5 Y3 1 .P00
Sec-WebSocket-Protocol: sample
Upgrade: WebSocket
Sec-WebSocket-Key1: 4 @1 46546xW%0l 1 5
Origin: http://example.com

 ^n:ds[4U
```

```
Respuesta del servidor:
HTTP/1.1 101 WebSocket Protocol Handshake
Upgrade: WebSocket
Connection: Upgrade
Sec-WebSocket-Origin: http://example.com
Sec-WebSocket-Location: ws://example.com/demo
Sec-WebSocket-Protocol: sample

  8jKS'y:G*Co,Wxa-
```

The WebSocket protocol was designed to work well with the existing Web infrastructure. As part of this design principle, the protocol specification defines that the WebSocket connection starts its life as an HTTP connection, guaranteeing full backwards compatibility with the pre-WebSocket world.

The protocol switch from HTTP to WebSocket is referred to as a the WebSocket handshake.

Una vez establecida, las tramas WebSocket de datos pueden empezar a enviarse en ambos sentidos entre el cliente y el servidor en modo full-duplex.

Las tramas de texto pueden ser enviadas en modo full-duplex también, en ambas direcciones al mismo tiempo.

Tramas de datos binarios no están soportadas todavía en el API.

- About HTML5 WebSockets en http://www.websocket.org

2.9.7. Una Aplicación Usando Websockets en la que Múltiples Clientes Dibujan en un Lienzo

server.rb

```
[19:36][~/srcSTW/streaming/websocketsDrawEM(master)]$ cat -n server.rb
     1  require 'em-websocket'
     2  require 'json'
     3  require 'sinatra/base'
     4
     5  EventMachine.run {
     6    @channel = EM::Channel.new
     7    @users = {}
     8    @messages = []
     9
    10    EventMachine::WebSocket.start(:host => "0.0.0.0", :port => 8080) do |ws|
    11
    12      ws.onopen {
    13        #Subscribe the new user to the channel with the callback function for the push a
    14        new_user = @channel.subscribe { |msg| ws.send msg }
    15
    16        #Add the new user to the user list
    17        @users[ws.object_id] = new_user
    18
    19        #Push the last messages to the user
    20        @messages.each do |message|
    21          ws.send message
    22        end
    23     }
    24
    25      ws.onmessage { |msg|
    26
    27        #append the message at the end of the queue
    28        @messages << msg
    29        @messages.shift if @messages.length > 10
    30
    31        #Broadcast the message to all users connected to the channel
    32        @channel.push msg
    33      }
    34
    35      ws.onclose {
    36        @channel.unsubscribe(@users[ws.object_id])
```

```
37        @users.delete(ws.object_id)
38      }
39    end
40
41    #Run a Sinatra server for serving index.html
42    class App < Sinatra::Base
43      set :public_folder, settings.root
44
45      get '/' do
46        send_file 'index.html'
47      end
48    end
49    App.run!
50  }
```

index.html.haml

```
!!! 5
%html
  %head
    %meta(charset="utf-8")
    %meta(content="IE=edge,chrome=1" http-equiv="X-UA-Compatible")
    %meta(name="viewport" content="width=device-width, user-scalable=0, initial-scale=1.0, max
    %link(rel="stylesheet" href="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css")
    %script(type="text/javascript" src="http://code.jquery.com/jquery-1.6.4.min.js")
    %script(type="text/javascript" src="http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.mi
    %title
      WebSockets Drawing
  %body
    :javascript
      var WebSocket = window.WebSocket || window.MozWebSocket;

      $(function() {

        var socket = new WebSocket("ws://" + location.hostname + ":8080");

        var currentX = 0;
        var currentY = 0;
        var lastX;
        var lastY;
        var lastReceivedX;
        var lastReceivedY;

        var ctx = $('#canvas')[0].getContext('2d');

        $('#canvas').bind('vmousemove',function(ev){
          ev = ev || window.event;
          currentX = ev.pageX || ev.clientX;
          currentY = ev.pageY || ev.clientY;
        });

        socket.onopen = function(event) {
          setInterval(function(){
            if(currentX !== lastX || currentY !== lastY){
```

```
            lastX = currentX;
            lastY = currentY;
            socket.send( JSON.stringify({x:currentX, y: currentY}) );
          }
        }, 30); // send every 300 milliseconds if position has changed
      }
      socket.onmessage = function(event) {
        var msg = $.parseJSON(event.data);

        ctx.beginPath();
        ctx.moveTo(lastReceivedX,lastReceivedY);
        ctx.lineTo(msg.x,msg.y);
        ctx.closePath();
        ctx.stroke();
        lastReceivedX = msg.x;
        lastReceivedY = msg.y;
      }
    });

  %div(data-role="page")
    %canvas#canvas(width='1000' height='1000')
```

index.html

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset='utf-8' />
    <meta content='IE=edge,chrome=1' http-equiv='X-UA-Compatible' />
    <meta content='width=device-width, user-scalable=0, initial-scale=1.0, maximum-scale=1.0;'
    <link href='http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.css' rel='stylesheet'
    <script src='http://code.jquery.com/jquery-1.6.4.min.js' type='text/javascript'></script>
    <script src='http://code.jquery.com/mobile/1.0/jquery.mobile-1.0.min.js' type='text/javasc
    <title>
      WebSockets Drawing
    </title>
  </head>
  <body>
    <script type='text/javascript'>
      //<![CDATA[
        var WebSocket = window.WebSocket || window.MozWebSocket;

        $(function() {

          var socket = new WebSocket("ws://" + location.hostname + ":8080");

          var currentX = 0;
          var currentY = 0;
          var lastX;
          var lastY;
          var lastReceivedX;
          var lastReceivedY;
```

```
          var ctx = $('#canvas')[0].getContext('2d');

          $('#canvas').bind('vmousemove',function(ev){
            ev = ev || window.event;
            currentX = ev.pageX || ev.clientX;
            currentY = ev.pageY || ev.clientY;
          });

          socket.onopen = function(event) {
            setInterval(function(){
              if(currentX !== lastX || currentY !== lastY){
                lastX = currentX;
                lastY = currentY;
                socket.send( JSON.stringify({x:currentX, y: currentY}) );
              }
            }, 30); // send every 300 milliseconds if position has changed
          }
          socket.onmessage = function(event) {
            var msg = $.parseJSON(event.data);

            ctx.beginPath();
            ctx.moveTo(lastReceivedX,lastReceivedY);
            ctx.lineTo(msg.x,msg.y);
            ctx.closePath();
            ctx.stroke();
            lastReceivedX = msg.x;
            lastReceivedY = msg.y;
          }
        });
      //]]>
    </script>
    <div data-role='page'>
      <canvas height='1000' id='canvas' width='1000'></canvas>
    </div>
  </body>
</html>
```

Enlaces Relacionados

- A shared canvas where multiple clients can draw lines using EM-Websocket
- Este ejemplo está tomado del material del curso *Mobile Web Applications Development with HTML5* [1]

2.10. Correo

```
[~/srcSTW/sinatra-faq/mail(esau)]$ cat app.rb
require 'sinatra'
require 'pony'
raise "Execute:\n\t#{$0} password email_to email_from" if ARGV.length.zero?
get '/' do
    email = ARGV.shift
    pass = ARGV.shift
```

```
    Pony.mail({
      :to => email,
      :body => "Hello Casiano",
      :subject => 'Howdy, Partna!',
      :via => :smtp,
      :via_options => {
          :address              => 'smtp.gmail.com',
          :port                 => '587',
          :enable_starttls_auto => true,
          :user_name            => ARGV.shift,
          :password             => pass,
          :authentication       => :plain, # :plain, :login, :cram_md5, no auth by default
          :domain               => "localhost.localdomain" # the HELO domain provided by the c
        }
    })
    "Check your email at #{email}"
end
```

2.11. Autentificación Básica

```
[~/srcSTW/sinatra-faq/authentication/basic(esau)]$ cat app.rb
require 'rubygems'
require 'sinatra'

use Rack::Auth::Basic, "Restricted Area" do |username, password|
  [username, password] == ['admin', 'admin']
end

get '/' do
  "You're welcome"
end

get '/foo' do
  "You're also welcome"
end
```

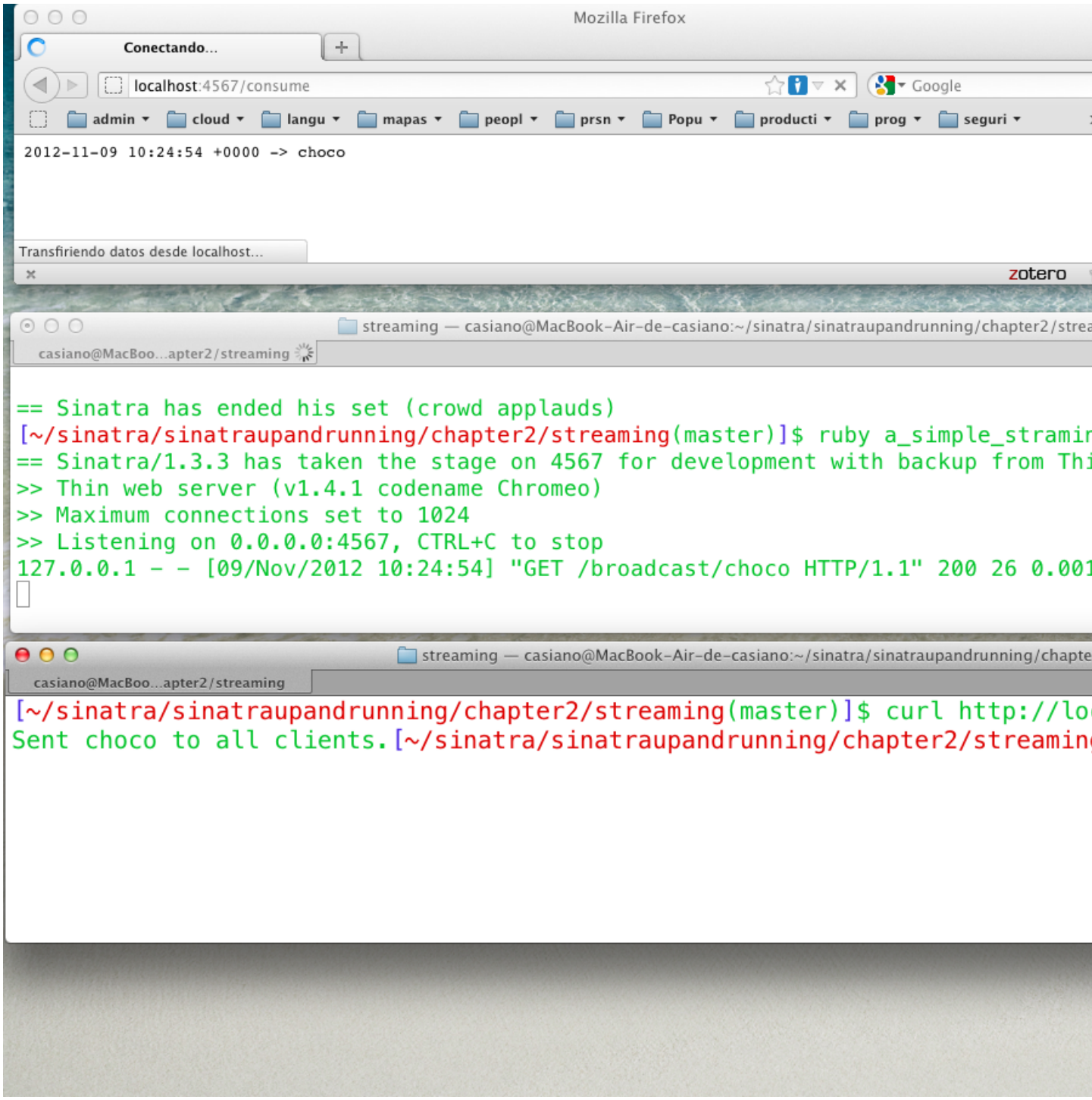

Figura 2.1: Manteniendo una Conexión Abierta

Figura 2.2: Chat en Sinatra Usando Streaming

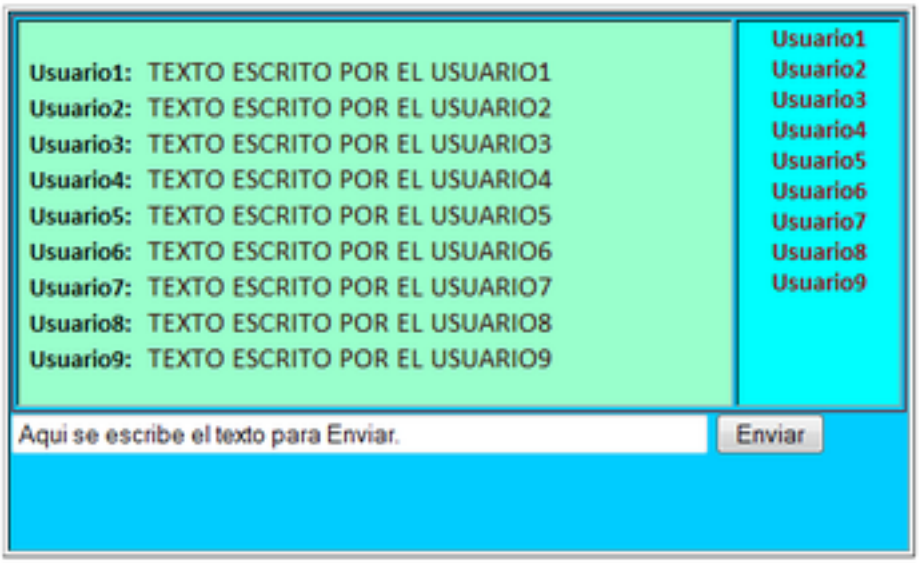

Figura 2.3: Típica disposición de un chat

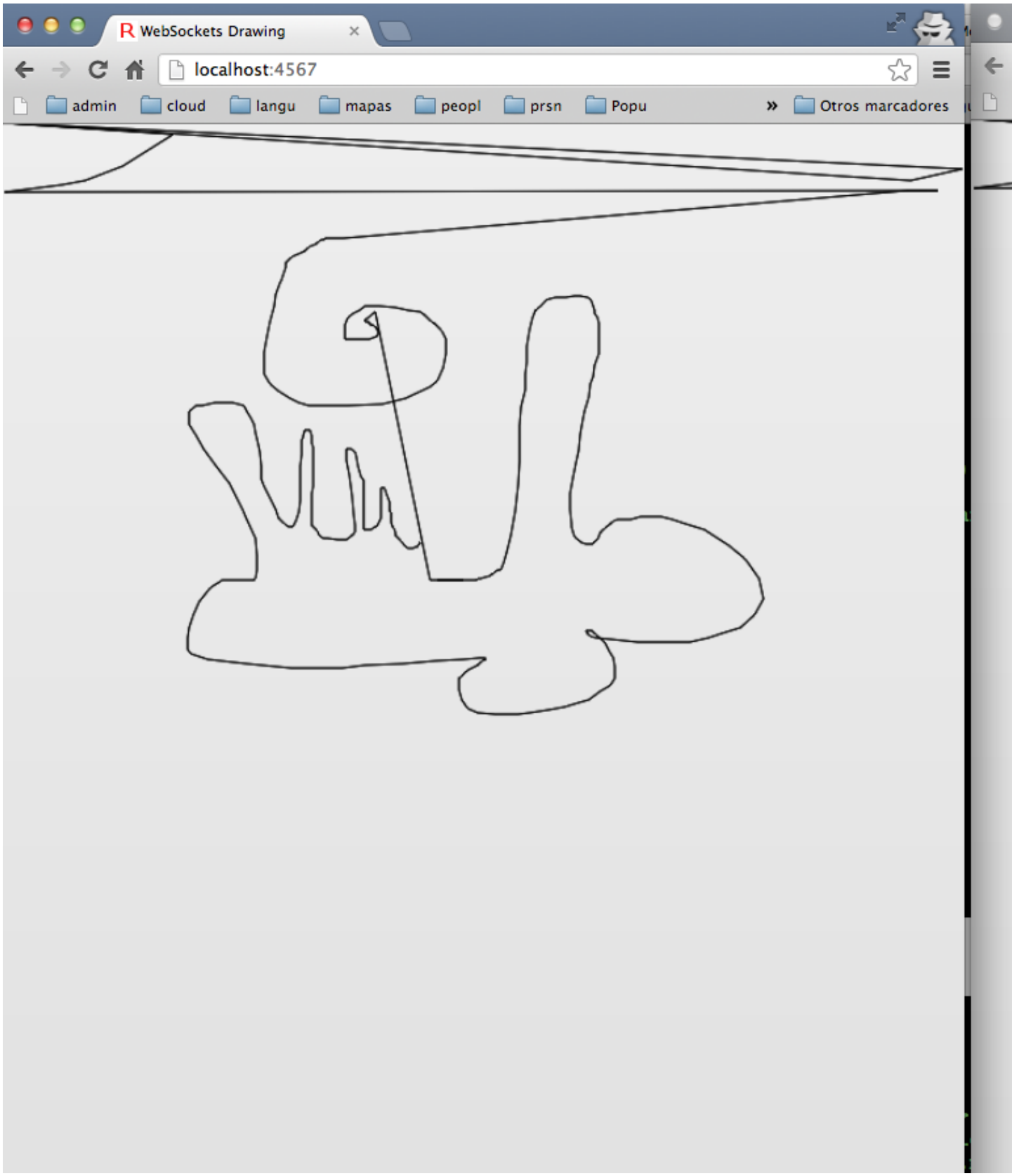

Figura 2.4: Múltiples clientes pueden dibujar en el lienzo

Capítulo 3

Mirando Detrás de Las Cortinas

Capítulo 4

Aplicaciones Modulares

Las aplicaciones normales Sinatra se denominan *alicaciones clásicas sinatra* y viven en Sinatra::Application, que es una subclase de Sinatra::Base.

En las aplicaciones clásicas Sinatra extiende la clase `Object` en el momento de cargarse lo que, en cierto modo, contamina el espacio de nombres global. Eso dificulta que nuestra aplicación pueda ser distribuída como una gema y que se puedan tener varias aplicaciones clásicas en un único proceso.

Una aplicación Sinatra se dice una *aplicación modular sinatra* si no hace uso de Sinatra::Application, renunciando al DSL de alto nivel proveído por Sinatra, sino que hereda de Sinatra::Base.

Podemos combinar una aplicación clásica con una modular, pero sólo puede haber una aplicación clásica por proceso.

Capítulo 5

Las Manos en la Masa: Nuestro Propio Blog Engine

Capítulo 6

Alternativas a Sinatra

6.1. Cuba

Cuba es una alternativa a Sinatra.

1. cuba

6.2. Grape

Grape es una alternativa a Sinatra.

1. Grape

6.3. Ramaze

Ramaze es una alternativa a Rails

1. Ramaze

Capítulo 7

Pruebas

1. NetTuts+ tutorial: Testing Web Apps with Capybara and Cucumber Andrew Burgess on Aug 22nd 2011 with 22 Comments

Parte II

RAILS

Capítulo 8

Active Record

8.1. Migraciones

8.1.1. Introducción a las Migraciones en Active Record

Las migraciones nos permiten gestionar la evolución del esquema utilizado por varias bases de datos físicas[1].

Es una solución al problema habitual de tener que añadir un campo para una nueva característica a nuestra base de datos pero no saber como sincronizar dicho cambio con el resto de los desarrolladores y con el servidor de producción.

Con las migraciones podemos *describir las transformaciones que hacemos al esquema de la base de datos en clases autocontenidas que pueden ser añadidas al sistema de control de versiones y ejecutadas contra otras bases de datos que pueden estar una, dos o cinco versiones atrás.*

Veamos un ejemplo sencillo de migración:

```
class AddSsl < ActiveRecord::Migration
  def up
    add_column :accounts, :ssl_enabled, :boolean, :default => 1
  end

  def down
    remove_column :accounts, :ssl_enabled
  end
end
```

Esta migración añade un flag booleano `ssl_enabled` a la tabla `accounts` o lo suprime si retiramos la migración.

Muestra como todas las migraciones disponen de dos métodos `up` y `down` que describen las transformaciones requeridas para implementar o suprimir la migración.

Estos métodos pueden contener llamadas a métodos específicos de migración como `add_column` o `remove_column`, pero pueden por supuesto código Ruby normal para, por ejemplo, generar los datos requeridos por las transformaciones.

Sigue un ejemplo de una migración mas compleja que también inicializa los datos:

```
class AddSystemSettings < ActiveRecord::Migration
  def up
    create_table :system_settings do |t|
      t.string  :name
      t.string  :label
      t.text    :value
```

[1] Cuando hablamos del nivel físico de una base de datos relacional nos referimos al nivel en el sistema de archivos, también podemos referirnos al nivel de las estructuras de datos

```
      t.string  :type
      t.integer :position
    end

    SystemSetting.create  :name => "notice",
                          :label => "Use notice?",
                          :value => 1
  end

  def down
    drop_table :system_settings
  end
end
```

Esta migración añade la tabla `system_settings` utilizando un bloque en el que se especifica el esquema de la tabla.

Después se crea la primera fila utilizando el modelo `SystemSetting` proveído por Active Record a partir de la descripción dada de la tabla.

8.1.2. Transformaciones Disponibles

1. `create_table(name, options)`: Crea una tabla denominada `name` y deja el objeto tabla disponible en el bloque, el cual puede añadir columnas siguiendo el mismo formato que `add_column`.

2. `drop_table(name)`:

   Descarta la tabla `name`.

3. `rename_table(old_name, new_name)`: Renombra la tabla de `old_name` a `new_name`

4. `add_column(table_name, column_name, type, options)`: Añade una nueva columna `column_name` a la tabla `table_name` especificada mediante uno de los siguientes tipos:

   a) `:string`
   b) `:text`
   c) `:integer`
   d) `:float`
   e) `:decimal`
   f) `:datetime`
   g) `:timestamp`
   h) `:time`
   i) `:date`
   j) `:binary`
   k) `:boolean`

   Se puede especificar un valor por defecto mediante el hash `options`, por ejemplo `{ :default => 11 }`. Otras opciones son `:limit` y `:null`. Por ejemplo `{ :limit => 50, :null => false}`

5. `rename_column(table_name, column_name, new_column_name)`: Renombra la columna pero conserva el tipo y el contenido

6. `change_column(table_name, colum_name, type, options)`: Cambia la columna a un tipo diferente utilizando los mismos parámetros que `add_column`

7. `remove_column(table_name, column_names)`: Suprime las columnas listadas en `column_names` de la tabla con nombre `tabel_name`

8. `add_index(table_name, clumn_names, options)`: Añade un nuevo çindice con e nombre de la columna. Otras opciones incluyen `:name`, `:unique` (Por ejemplo `{ :name => "users_name_index", :uniqu`
o `:order` (por ejemplo `{ :order => { :name => :desc}}`)

9. `remove_index(table_name, :column => column_name)`: Suprime el índice especificado por `column_name`

10. `remove_index(table_name, :name => index_name)`: Suprime el índice especificado por `index_name`

8.1.3. Transformaciones Irreversibles

Algunas transformaciones son destructivas en un modo que no puede ser desecho. Las migraciones de esta clase deberían generar una excepción `ActiveRecord::IrreversibleMigration` en su método `down`.

8.1.4. Ejecutando las Migraciones con Sinatra

Para usar las migraciones ActiveRecord con Sinatra (y en general con un proyecto que no esté dentro de Rails) debemos añadir lo siguiente a nuestro Rakefile:

```
namespace :db do
  task :environment do
    require 'active_record'
    ActiveRecord::Base.establish_connection :adapter => 'sqlite3', :dbfile =>  'db/test.sqlite
  end

  desc "Migrate the database"
  task(:migrate => :environment) do
    ActiveRecord::Base.logger = Logger.new(STDOUT)
    ActiveRecord::Migration.verbose = true
    ActiveRecord::Migrator.migrate("db/migrate")
  end
end
```

Este código crea una tarea denominada `:environment` que carga el entorno de nuestra aplicación (requiere los ficheros adecuados, establece la conexión con la base de datos, etc.)

Después deberemos crear un directorio denominado `db/migrate` y rellenarlo con nuestras migraciones. Podemos irlas enumerando con nombres como `001_init.rb`, etc. o bien usar la fecha y la hora como hace en Rails hoy en día.

Lo mas sencillo para utilziar ActiveRecord con Sinatra es usar la gema Sinatra ActiveRecord Extension

8.2. Práctica: Servicio para Abreviar URLs

Escriba un acortador de URLs usando ActiveRecords y sinatra-activerecord Asegúrese de tener instalados:

1. gem install activerecord

2. gem install sinatra-activerecord

3. gem install sqlite3

Este es un ejemplo de estructura de la aplicación en su forma final:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ tree -A
.
|-- Gemfile
|-- Gemfile.lock
|-- README
|-- Rakefile
|-- app.rb
|-- db
|   |-- config.yml
|   `-- migrate
|       `-- 20121017115717_shortened_urls.rb
|-- shortened_urls.db
|-- shortened_urls_bak.db
`-- views
    |-- index.haml
    |-- layout.haml
    `-- success.haml

3 directories, 12 files
```

Una vez instaladas las gemas implicadas:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat Gemfile
source 'https://rubygems.org'

#gem 'alphadecimal'
gem 'sinatra-activerecord'
gem 'sqlite3'
```

con `bundle install`, procedemos a añadir objetivos al `Rakefile`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat Rakefile
$: << '.' # add current path to the search path
require 'sinatra/activerecord/rake'
require 'app'

desc "Reset the data base to initial state"
task :clean do
  sh "mv shortened_urls.db tmp/"
  sh "mv db/migrate /tmp/"
end

desc "Create the specific ActiveRecord migration for this app"
task :create_migration do
  sh "rake db:create_migration NAME=create_shortened_urls"
end

desc "shows the code you have to have in your db/migrate/#number_shortened_urls.rb file"
task :edit_migration do
  source = <<EOS
class ShortenedUrls < ActiveRecord::Migration
  def up
    create_table :shortened_urls do |t|
      t.string :url
```

```
    end
    add_index :shortened_urls, :url
  end

  def down
    drop_table :shortened_urls
  end
end
EOS
  puts "Edit the migration and insert this code:"
  puts source
end

desc "run the url shortener app"
task :run do
  sh "ruby app.rb"
end
```

Este `Rakefile` tiene los siguientes objetivos:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ rake -T
rake clean                # Reset the data base to initial state
rake create_migration     # Create the specific ActiveRecord migration for this app
rake db:create_migration  # create an ActiveRecord migration in ./db/migrate
rake db:migrate           # migrate the database (use version with VERSION=n)
rake db:rollback          # roll back the migration (use steps with STEP=n)
rake edit_migration       # shows the code you have to have in your db/migrate/#number_shorten
rake run                  # run the url shortener app
[~/srcSTW/url_shortener_with_active_records(master)]$
```

De ellos los tres que nos importan ahora son `db:create_migration`, `db:migrate` y `db:rollback` que son creados por la línea `require 'sinatra/activerecord/rake'`.

Las migraciones nos permiten gestionar la evolución de un esquema utilizado por varias bases de datos. Es una solución al problema habitual de añadir un campo para proveer una nueva funcionalidad en nuestra base de datos, pero no tener claro como comunicar dicho cambio al resto de los desarrolladores y al servidor de producción. Con las migraciones podemos describir las transformaciones mediante clases autocintenidas que pueden ser añadadidas a nuestro repositorio `git` y ejecutadas contra una base de datos que puede estar una, dos o cinco versiones atrás.

Comenzemos configurando el modelo para nuestra aplicación. En el directorio `db` creamos el fichero `config.yml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat db/config.yml
development:
  adapter: sqlite3
  encoding: utf8
  database: shortened_urls_dev.sqlite

test:
  adapter: sqlite3
  encoding: utf8
  database: shortened_urls_test.sqlite

production:
```

```
  adapter: sqlite3
  encoding: utf8
  database: shortened_urls_live.sqlite
```

Comenzaremos creando nuestro modelo. Para ello ejecutamos `rake db:create_migration`. Como vemos debemos pasar una opción `NAME` para nuestra migración. En nuestro caso pasamos `NAME=create_shortened_`
Esto crea la carpeta `db/migrate` que contienen nuestra migración.

```
[~/srcSTW/url_shortener_with_active_records(master)]$ tree db
db
|-- config.yml
`-- migrate
    `-- 20121017115717_shortened_urls.rb
```

ahora rellenamos los métodos `up` y el `down`:

```
1 directory, 2 files
[~/srcSTW/url_shortener_with_active_records(master)]$ cat db/migrate/20121017115717_shortened_
class ShortenedUrls < ActiveRecord::Migration
  def up
    create_table :shortened_urls do |t|
      t.string :url
    end
    add_index :shortened_urls, :url
  end

  def down
    drop_table :shortened_urls
  end
end
```

cuando ejecutamos `rake db:migrate` se ejecuta la migración y crea la base de datos con la tabla `shortened_urls`.

En nuestro fichero `app.rb` creamos nuestro modelo `ShortenedUrl`. Para ello escribimos:

```
class ShortenedUrl < ActiveRecord::Base
end
```

Después incluímos algunas validaciones:

```
class ShortenedUrl < ActiveRecord::Base
  # Validates whether the value of the specified attributes are unique across the system.
  validates_uniqueness_of :url
  # Validates that the specified attributes are not blank
  validates_presence_of :url
  #validates_format_of :url, :with => /.*/
  validates_format_of :url,
       :with => %r{^(https?|ftp)://.+}i,
       :allow_blank => true,
       :message => "The URL must start with http://, https://, or ftp:// ."
end
```

A continuación escribimos las rutas:

```
get '/' do

end
```

```
post '/' do
end
```

Creamos también una ruta `get` para redireccionar la URL acortada a su destino final:

```
get '/:shortened' do
end
```

Supongamos que usamos el `id` de la URL en la base de datos para acortar la URL. Entonces lo que tenemos que hacer es encontrar mediante el método find la URL:

```
get '/:shortened' do
  short_url = ShortenedUrl.find(params[:shortened].to_i(36))
  redirect short_url.url
end
```

Mediante una llamada de la forma `Model.find(primary_key)` obtenemos el objeto correspondiente a la clave primaria específicada. que casa con las opciones suministradas.

El SQL equivalente es:

```
SELECT * FROM shortened_urls WHERE (url.id = params[:shortened].to_i(36)) LIMIT 1
```

`Model.find(primary_key)` genera una excepción `ActiveRecord::RecordNotFound` si no se encuentra ningún registro.

Veamos una sesión on `sqlite3`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ sqlite3 shortened_urls.db
SQLite version 3.7.7 2011-06-25 16:35:41
Enter ".help" for instructions
Enter SQL statements terminated with a ";"
sqlite> .schema
CREATE TABLE "schema_migrations" ("version" varchar(255) NOT NULL);
CREATE TABLE "shortened_urls" ("id" INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, "url" varchar(
CREATE INDEX "index_shortened_urls_on_url" ON "shortened_urls" ("url");
CREATE UNIQUE INDEX "unique_schema_migrations" ON "schema_migrations" ("version");
sqlite> select * from  shortened_urls;
1|https://mail.google.com/mail/u/0/
2|https://plus.google.com/u/0/
3|http://campusvirtual.ull.es/1213m2/mod/forum/discuss.php?d=5629
4|http://www.sinatrarb.com/intro#Accessing%20Variables%20in%20Templates
sqlite> select * from shortened_urls where (id = 3) limit 1;
3|http://campusvirtual.ull.es/1213m2/mod/forum/discuss.php?d=5629
sqlite>  .quit
```

Este es el código completo de `app.rb`:

```
$ cat app.rb
require 'sinatra'
require 'sinatra/activerecord'
require 'haml'

set :database, 'sqlite3:///shortened_urls.db'
#set :address, 'localhost:4567'
set :address, 'exthost.etsii.ull.es:4567'
```

```
class ShortenedUrl < ActiveRecord::Base
  # Validates whether the value of the specified attributes are unique across the system.
  validates_uniqueness_of :url
  # Validates that the specified attributes are not blank
  validates_presence_of :url
  #validates_format_of :url, :with => /.*/
  validates_format_of :url,
       :with => %r{^(https?|ftp)://.+}i,
       :allow_blank => true,
       :message => "The URL must start with http://, https://, or ftp:// ."
end


get '/' do
  haml :index
end

post '/' do
  @short_url = ShortenedUrl.find_or_create_by_url(params[:url])
  if @short_url.valid?
    haml :success, :locals => { :address => settings.address }
  else
    haml :index
  end
end

get '/:shortened' do
  short_url = ShortenedUrl.find(params[:shortened].to_i(36))
  redirect short_url.url
end
```

Las Vistas. Primero el fichero `views/layout.haml`:

```
$ cat views/layout.haml
!!!
%html
    %body
        =yield
        %form(action="/" method="POST")
            %label(for="url")
            %input(type="text" name="url" id="url" accesskey="s")
            %input(type="submit" value="Shorten")
```

Creamos un formulario que envía con `method="POST"` a la raíz `action="/"`. tiene un elemento `input` para obtener la URL.

El formulario es procesado por la ruta `post '/'`:

```
post '/' do
  @short_url = ShortenedUrl.find_or_create_by_url(params[:url])
  if @short_url.valid?
    haml :success, :locals => { :address => settings.address }
  else
    haml :index
  end
end
```

El método find_or_create encontrará la URL o creará una nueva.

El fichero `views/index.haml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat views/index.haml
- if @short_url.present? && !@short_url.valid?
  %p Invalid URL: #{@short_url.url}
```

El fichero `views/success.haml`:

```
[~/srcSTW/url_shortener_with_active_records(master)]$ cat views/success.haml
%p #{params}
%p http://#{address}/#{@short_url.id.to_s(36)}
```

Puede ir añadiendo extensiones a la práctica:

1. Añada una opción para mostrar la lista de URLs abreviadas El método `find` puede serle útil:

   ```
   get '/show' do
     urls = ShortenedUrl.find(:all)
     ...
     haml :show
   end
   ```

2. Añada una opción para buscar por una abreviación y mostrar la URL

3. Añada una opción para buscar por una URL y mostrar la abreviación

4. Añada una opción que permita una abreviación personalizada, siempre que esté libre. Por ejemplo, abreviar `http://www.sinatrarb.com/documentation` a `http://localhost:4567/sindoc`

   Esto obliga a poner una opción para ello en el formulario:

   ```
   %form(action="/" method="POST")
           %label(for="url") URL
           %input(type="text" name="url" id="url" accesskey="s")
           %br
           %label(for="custom") Custom Shortened URL(optional)
           %input(type="text" name="custom" id="custom" accesskey="t")
           %br
           %input(type="submit" value="Shorten" class="btn btn-primary")
   ```

   y a comprobar de alguna manera si la opción `custom` contiene algo.

8.3. Práctica: Servicio para Abreviar URLs Teniendo en Cuenta el País de Visita

Añádale a la práctica 8.2 la funcionalidad de mostrar el número de visitas y el número de visitas por país.

Puede usar para ello la API de http://www.hostip.info/:

```
after :create, :set_country

def set_country
  xml = RestClient.get "http://api.hostip.info/get_xml.php?ip=#{ip}"
  self.country = XmlSimple.xml_in(xml.to_s, { 'ForceArray' => false })['featureMember']['Hos
  self.save
end
```

- XMLSimple le puede ayudar a parsear el XML: http://xml-simple.rubyforge.org/
- https://github.com/sausheong/tinyclone
- Véase la API de http://www.hostip.info/

Capítulo 9

devise

- https://github.com/plataformatec/devise

Parte III

PADRINO

- http://www.padrinorb.com/

Parte IV

TESTING

Capítulo 10

Rspec

Capítulo 11

Cucumber

Parte V

COMPUTACIÓN MÓVIL

Capítulo 12

Ruboto

- http://ruboto.org/

Capítulo 13

RubyMotion

- http://www.rubymotion.com/

Parte VI

BASES DE DATOS

Capítulo 14

Postgres

- How to Setup PostgreSQL for Rails Development on OS X por CooperPress

Capítulo 15

DataMapper

15.1. Enlaces

1. Screencast: Ruby for Newbies: Working with DataMapper Andrew Burgess on Apr 19th 2011

Capítulo 16

MongoDB

16.1. Que es

> MongoDB wasn't designed in a lab. We built MongoDB from our own experiences building large scale, high availability, robust systems. We didn't start from scratch, we really tried to figure out what was broken, and tackle that. So the way I think about MongoDB is that if you take MySql, and change the data model from relational to document based, you get a lot of great features: embedded docs for speed, manageability, agile development with schema-less databases, easier horizontal scalability because joins aren't as important. There are lots of things that work great in relational databases: indexes, dynamic queries and updates to name a few, and we haven't changed much there. For example, the way you design your indexes in MongoDB should be exactly the way you do it in MySql or Oracle, you just have the option of indexing an embedded field.

– Eliot Horowitz, 10gen CTO and Co-founder

16.2. Instalación

```
# En imac
[12:29][~/Downloads/mongodb-osx-x86_64-2.2.0]$ brew install mongodb
Warning: It appears you have MacPorts or Fink installed.
Software installed with other package managers causes known problems for
Homebrew. If a formula fails to build, uninstall MacPorts/Fink and try again.
==> Downloading http://fastdl.mongodb.org/osx/mongodb-osx-x86_64-2.0.4.tgz
######################################################################## 100,0%
==> Caveats
If this is your first install, automatically load on login with:
    mkdir -p ~/Library/LaunchAgents
    cp /usr/local/Cellar/mongodb/2.0.4-x86_64/homebrew.mxcl.mongodb.plist ~/Library/LaunchAgen
    launchctl load -w ~/Library/LaunchAgents/homebrew.mxcl.mongodb.plist

If this is an upgrade and you already have the homebrew.mxcl.mongodb.plist loaded:
    launchctl unload -w ~/Library/LaunchAgents/homebrew.mxcl.mongodb.plist
    cp /usr/local/Cellar/mongodb/2.0.4-x86_64/homebrew.mxcl.mongodb.plist ~/Library/LaunchAgen
    launchctl load -w ~/Library/LaunchAgents/homebrew.mxcl.mongodb.plist

Or start it manually:
    mongod run --config /usr/local/etc/mongod.conf

The launchctl plist above expects the config file to be at /usr/local/etc/mongod.conf.
==> Summary
```

```
/usr/local/Cellar/mongodb/2.0.4-x86_64: 17 files, 121M, built in 17 seconds
```

16.3. tutorial

Tutorial

```
12:31][~/Downloads/mongodb-osx-x86_64-2.2.0]$ mongo
MongoDB shell version: 2.0.4
connecting to: test
> help
  db.help()                    help on db methods
  db.mycoll.help()             help on collection methods
  rs.help()                    help on replica set methods
  help admin                   administrative help
  help connect                 connecting to a db help
  help keys                    key shortcuts
  help misc                    misc things to know
  help mr                      mapreduce

  show dbs                     show database names
  show collections             show collections in current database
  show users                   show users in current database
  show profile                 show most recent system.profile entries with time >= 1ms
  show logs                    show the accessible logger names
  show log [name]              prints out the last segment of log in memory, 'global' is defau
  use <db_name>                set current database
  db.foo.find()                list objects in collection foo
  db.foo.find( { a : 1 } )     list objects in foo where a == 1
  it                           result of the last line evaluated; use to further iterate
  DBQuery.shellBatchSize = x   set default number of items to display on shell
  exit                         quit the mongo shell
> use mydb
switched to db mydb
> j = { name : "mongo" };
{ "name" : "mongo" }
> t = { x : 3 };
{ "x" : 3 }
> db.things.save(j);
>  db.things.save(t);
> db.things.find();
{ "_id" : ObjectId("5078042d00abf72f8a7945bd"), "name" : "mongo" }
{ "_id" : ObjectId("5078043600abf72f8a7945be"), "x" : 3 }
> for (var i = 1; i <= 20; i++) db.things.save({x : 4, j : i});
> db.things.find();
{ "_id" : ObjectId("5078042d00abf72f8a7945bd"), "name" : "mongo" }
{ "_id" : ObjectId("5078043600abf72f8a7945be"), "x" : 3 }
{ "_id" : ObjectId("5078045800abf72f8a7945bf"), "x" : 4, "j" : 1 }
{ "_id" : ObjectId("5078045800abf72f8a7945c0"), "x" : 4, "j" : 2 }
{ "_id" : ObjectId("5078045800abf72f8a7945c1"), "x" : 4, "j" : 3 }
{ "_id" : ObjectId("5078045800abf72f8a7945c2"), "x" : 4, "j" : 4 }
{ "_id" : ObjectId("5078045800abf72f8a7945c3"), "x" : 4, "j" : 5 }
{ "_id" : ObjectId("5078045800abf72f8a7945c4"), "x" : 4, "j" : 6 }
{ "_id" : ObjectId("5078045800abf72f8a7945c5"), "x" : 4, "j" : 7 }
```

```
{ "_id" : ObjectId("5078045800abf72f8a7945c6"), "x" : 4, "j" : 8 }
{ "_id" : ObjectId("5078045800abf72f8a7945c7"), "x" : 4, "j" : 9 }
{ "_id" : ObjectId("5078045800abf72f8a7945c8"), "x" : 4, "j" : 10 }
{ "_id" : ObjectId("5078045800abf72f8a7945c9"), "x" : 4, "j" : 11 }
{ "_id" : ObjectId("5078045800abf72f8a7945ca"), "x" : 4, "j" : 12 }
{ "_id" : ObjectId("5078045800abf72f8a7945cb"), "x" : 4, "j" : 13 }
{ "_id" : ObjectId("5078045800abf72f8a7945cc"), "x" : 4, "j" : 14 }
{ "_id" : ObjectId("5078045800abf72f8a7945cd"), "x" : 4, "j" : 15 }
{ "_id" : ObjectId("5078045800abf72f8a7945ce"), "x" : 4, "j" : 16 }
{ "_id" : ObjectId("5078045800abf72f8a7945cf"), "x" : 4, "j" : 17 }
{ "_id" : ObjectId("5078045800abf72f8a7945d0"), "x" : 4, "j" : 18 }
has more
>
```

[2]

Capítulo 17

MongoMapper: Un ORM Mongo para Ruby

- MongoMapper

Parte VII

HTML y JAVASCRIPT

Capítulo 18

Node.js

Capítulo 19

CoffeScript

1. CoffeeScript book
2. Railcast: CoffeeScript
3. vim plugin para CoffeScript

Capítulo 20

HAML

Capítulo 21

Backbone

- http://backbonejs.org/
- Introduction to Backbone.js Part 1: Models – Video Tutorial

Capítulo 22

Responsive Design

Esta técnica de diseño web consiste en crear una estructura de una página web que según el tamaño de la pantalla (o ventana) en la que se visualice variará su contenido para que siempre sea visible y cómodo de usar desde PC, tablets y smartphones (principalmente, porque esto afectará a todo con lo que lo visualicemos, si así lo establecemos) y se puede poner en práctica esta forma de adaptar el contenido a todo tipo de resoluciones con hojas de estilo CSS (que es en lo que me voy a centrar) y con JavaScript (tenemos que tener en cuenta también en que puede haber personas o bots que tengan desactivado JavaScript).

- Responsive Web Design
- Response JS

Parte VIII

SEGURIDAD Y AUTENTIFICACIÓN

Capítulo 23

SSL

Capítulo 24

TSL

Capítulo 25

CAS

Capítulo 26

Auth

Capítulo 27

OpenID

- OmniAuth

Capítulo 28

Ataques

- MITM (Man In The Middle) por Alberto Pérez

Parte IX

COMERCIO ELECTRÓNICO

Capítulo 29

Dinero Elctrónico

Capítulo 30

Joomla

- http://www.joomla.org/
- http://virtuemart.net/
- Joomla + virtualMart = Ecomerce Uso de las CMS como herramienta para el diseño de web comerciales. por Leonardo Siverio

Capítulo 31

EBay

- http://www.ebay.es/
- http://developer.ebay.com/common/api/

Parte X

ADMINISTRACIÓN ELECTRÓNICA

Capítulo 32

DNI electrónico

Parte XI

PUBLICACIONES ELECTRÓNICAS Y LIBRERÍAS DIGITALES

- Introduction to Digital Libraries
- Digital Libraries, by William Arms

Parte XII

HERRAMIENTAS y SERVICIOS

Capítulo 33

Herramientas Colaborativas

33.1. Pivotal Tracker

http://www.pivotaltracker.com/

33.2. Open Atrium

33.3. Trello

https://trello.com/

Capítulo 34

Editores y Entornos de Desarrollo

- Sublime

Capítulo 35

Gravatar

- https://es.gravatar.com/ Gravatar en Ruby

Parte XIII

BITACORA

Capítulo 36

2012/2013

36.1. Jueves 08/11/2012

1. Request Object Véase la sección *Request Object* 1.2
2. Streaming Véase la sección *Streaming* 2.9
3. Práctica. Chat Véase la sección *Práctica. Chat* 2.9.3
4. Correo: Pony Véase la sección *Correo: Pony* 2.10
5. Basic Authentication Véase la sección *Basic Authentication* 2.11
6. Active Record Véase la sección *Active Record* 8

Índice general

Índice de figuras

Índice de cuadros

Índice alfabético

Bibliografía

[1] Claudio Riva, Mikael Blomberg, and Häkan Mitts. *Mobile Web Applications Development with HTML5. http://aaltowebapps.com/index.html*, 2012.

[2] Paolo Perrota. *Metaprogramming Ruby.* The Pragmatic programmers, 2010.