Mostrar Información al usuario

Rodrigo Rodriguez
rorodr@gmail.com
Skype: rarodriguezr

Action Views

- Como se vio en la clase pasada, cada acción tiene una vista asociada
- Las vistas pueden generarse usando distintos sistemas de “maquetado”, siendo el más popular ERB.
- Por lo general las vistas incluyen métodos comunes denominados helpers

Templates

- Templates pueden escribirse de diversas maneras, siendo la principal ERB
- Si se utiliza la extensión '.erb', significa que se utiliza ERB(Embedded Ruby) y HTML, ej:

```
<div>
   <%= "Hello World!" * 10 %>
</div>
<div>
   <% alpha = 25 %>
   <span><%= alpha – 2 %></span>
</div>
```

ERB

- En un template ERB, se puede incluir código ruby ejecutable de 2 formas:
  - “<%= %>”: Se utiliza cuando se desea imprimir valores en la pantalla
  - “<% %>”: Se utiliza para ejecutar código que no retorne ningún valor, como condiciones, ciclos o bloques.

Partials (1/3)

- Es una reducción para “Partial Templates”
- Se utilizan para organizar las vistas, pues permite modularizar las vistas en distintas secciones reutilizables
- Los nombres de los parciales siempre empiezan por un ‘_’, ej: “_form.html.erb”
- Para agregar un parcial a un template, se utiliza render:

```
<%= render ‘form’ %>
```

Partials (2/3)

- Es posible pasar una variable a un partial, para ello se utiliza:
  - `render partial: "product", locals: {product: product}`
  - `render "product", {product: product}`
- Ejemplo:

```
<% @products.each do |product| %>
  <%= render partial: "product", locals: {product: product} %>
<% end %>
```

Partials (3/3)

- Algunas veces es necesario imprimir una lista de productos, por lo que podría utilizar algo como esto:

```
<% @products.each do |product| %>
  <%= render partial: "product", locals:
{ product: product } %>
<% end %>
```

- Lo que se podría escribir también:

```
<%= render partial: "product", collection:
@products %>
```

Layouts

- Se utiliza para generar un *template* común para las distintas páginas.
- En muchos de los casos se tienen 2 layouts distintos, uno para la parte administrativa y uno para la parte pública
- Existen layouts parciales que permiten establecer un formato específico para diversas secciones de las vistas

Layouts parciales

- Asumamos el siguiente código:
  - `<%= render partial: 'post', layout: 'box', locals: {post: @post} %>`
- Tenemos un parcial llamado “box” en la misma carpeta, que contiene:

```
<div class='box'>
  <%= yield %>
</div>
```

- Con ello se imprime el contenido del parcial ‘post’

Helpers (1/5)

- Son métodos reutilizables que facilitan la generación de HTML o la ejecución de acciones específicas en las vistas.
- Se utilizan para reducir al máximo posible la cantidad de código Ruby en los archivos ‘erb’
- Rails provee un gran número de helpers nativos, desde generación de HTML hasta generar caché de contenido

Helpers (2/5)

- **content_tag_for**: Permite generar el tag de HTML que uno desee. Ej:

```
<%= content_tag_for(:tr, @post, class:
"frontpage") do %>
  <td><%= @post.title %></td>
<% end %>
```

- **image_tag:** retorna un tag html para imágenes

```
image_tag("icon.png")
 => <img src="/assets/icon.png" alt="Icon" />
```

Helpers (3/5)

- **javascript_include_tag**: permite incluir un/ varios javascript en el HTML. El nombre del jasvascript se envía por parámetro
- **stylesheet_link_tag:** Permite incluir un/varios CSS en el HTML
- **cache**: permite generar caché de una sección del html generado.

```
<% cache do %>
  <%= render "shared/footer" %>
<% end %>
```

Helpers (4/5)

- **content_for**: permite almacenar un bloque de HTML en un identificador para ser utilizado posteriormente

```
<p>This is a special page.</p>
 <% content_for :special_script do %>
  <script>alert('Hello!')</script>
<% end %>
```

  - En otro archivo:

```
<div><%= yield :special_script %></div>
```

Helpers (5/5)

- **date_select/datetime_select**: Retorna dropdowns para elegir fecha/hora para un campo específico

```
date_select("post", "published_on”)
```

- **debug**: muestra en pantalla información de fácil lectura sobre un objeto (lo inspecciona)
- **link_to:** genera un enlace a un url/controller específico

```
link_to profile_path(@profile),
method: :delete, remote:true, confirm: "Are
you sure?"
```

Form Helpers

- Permiten facilitar la creación de formularios:

```
<%= form_for @person, url: {action: "create"}
do |f| %>
  <%= f.text_field :first_name %>
  <%= f.email_field :email %>
  <%= f.password_field :password %>
  <%= f.text_area :address %>
  <%= f.check_box :active %>
  <%= f.label :active %>
  <%= submit_tag 'Create' %>
<% end %>
```

Para más información de rutas visitar:
http://guides.rubyonrails.org/routing.html

Number Helpers

- **number_to_currency:** Muestra un número con un formato monetario
- **number_to_human_size:** Muestra un número en KB, MB
- **number_to_percentage:** Muestra el número como un porcentaje
- **number_with_precision:** Muestra el número con la precisión deseada

Sanitize Helpers

- Incluyen helpers que permiten escapar HTML/CSS, con el fin de evitar ataques del lado del usuario
- **Sanitize:** decodifica todas las etiquetas de html
  - `sanitize @article.body, tags: %w(table tr td), attributes: %w(id class style)`
- **strip_tags(html):** Elimina los tags de HTML que se encuentran en el parámetro
  - `strip_tags("Strip <i>these</i> tags!")`

Helpers Personalizados

- A parte de los helpers nativos, es posible declarar helpers personalizados para el proyecto
- Los helpers se almacenan en app/helpers y son módulos de Ruby (en vez de clases)
- Por defecto Rails incluye todos los helpers en todas las vistas, por lo que nombres repetidos tienen prioridad en el módulo en que se encuentre (ej. UsersHelper en la vista de ver usuario)

Asset Pipeline

- Se incorpora a partir de Rails 3.1
- Ayuda a manejar los assets del HTML, entre los que se citan: CSS, Javascripts, imágenes, fonts, etc.
- Por defecto se utiliza SASS para encapsular el CSS y Coffeescript para encapsular el Javascript
- Pre-procesa los archivos en busca de código ruby (ERB) u sintaxis específica de SASS/ Coffescript, para generar los archivos resultantes

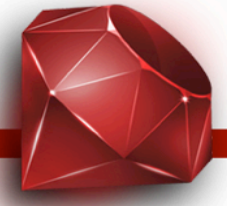
Asset Pipeline

- En modo producción, es requerido precompilar los assets, utilizando el comando:
  - rake assets:precompile
- Además, es factible configurar el URL resultante, el método de compresión y cualquier otra configuración al respecto.
- En caso de no querer utilizar SASS/ CoffeeScript, se puede agregar indistintamente archivos JS/CSS

SASS

- “Lenguaje” de pre-procesamiento de CSS, el cual permite generar hojas de estilo más ordenadas y dinámicas.
- Entre sus principales funcionalidades se encuentran:
  - Creación de variables, métodos (mixins), herencia y otras alternativas que facilitan la creación de CSS
- Cuenta con 2 sintaxis: SASS y SCSS

SASS: ejemplo



```
nav {
  ul {
    padding: 0;
  }
  li { display:
     inline-block; }
  a {
    padding: 6px 12px;
  }
}
```

SCSS

```
nav
  ul
    padding: 0
  li
    display:
inline-block
  a
    padding: 6px
12px
```

SASS

CoffeeScript

- “Pequeño lenguaje que compila en Javascript”
- Antes de ejecutarse, el coffeescript debe de “compilarse” en un javascript.
- La idea de Coffeescript es facilitar la creación de Javascript, por medio de la simplificación de sintaxis y la generación automática de ciertos aspectos de javascript como declaración de variables.

CoffeeScript: ejemplo

```
math =
  root:   Math.sqrt
  square: square
  cube:   (x) -> x *
square x
square = (x) -> x * x
```

```
square = function(x) {
  return x * x;
};

list = [1, 2, 3, 4, 5];

math = {
  root: Math.sqrt,
  square: square,
  cube: function(x) {
    return x * square(x);
  }
};
```

CoffeeScript **Javascript**

CoffeeScript: ejemplos

- En Rails un controlador es una clase que hereda de ApplicationController y tiene sus métodos propios, los que pueden ser públicos, privados y protegidos

```
class StoresController < ApplicationController
  def new
  end
  private
  def sample_calculation; end
end
```

Internacionalización

- Rails cuenta con una librería nativa para internacionalizar las aplicaciones, llamada I18n.
- I18n cuenta con 2 métodos:
  - translate: Traduce una llave específica según el lenguaje que esté activo
    - `I18n.t 'store.title'`
  - localize: convierte objetos (como fechas) en el formato del lenguaje activo
    - `I18n.l Time.now`

I18n: configuración

- Por defecto viene activado el idioma inglés, si se quiere cambiar a otro idioma, basta modificar '/config/application.rb' y agregar:
  - `config.i18n.default_locale = :es`
- Posteriormente, los archivos de internacionalización se almacenan en 'config/locales' y requieren el formato YML o rb con el nombre idéntico al nombre del locale (ej: es.yml)

Recursos adicionales

- ***Libro:*** *Agile Web Development with Rails 4*
- http://guides.rubyonrails.org/v4.1.8/action_view_overview.html
- http://api.rubyonrails.org/classes/ActionView/Helpers/UrlHelper.html
- http://guides.rubyonrails.org/form_helpers.html
- http://guides.rubyonrails.org/v4.1.8/i18n.html