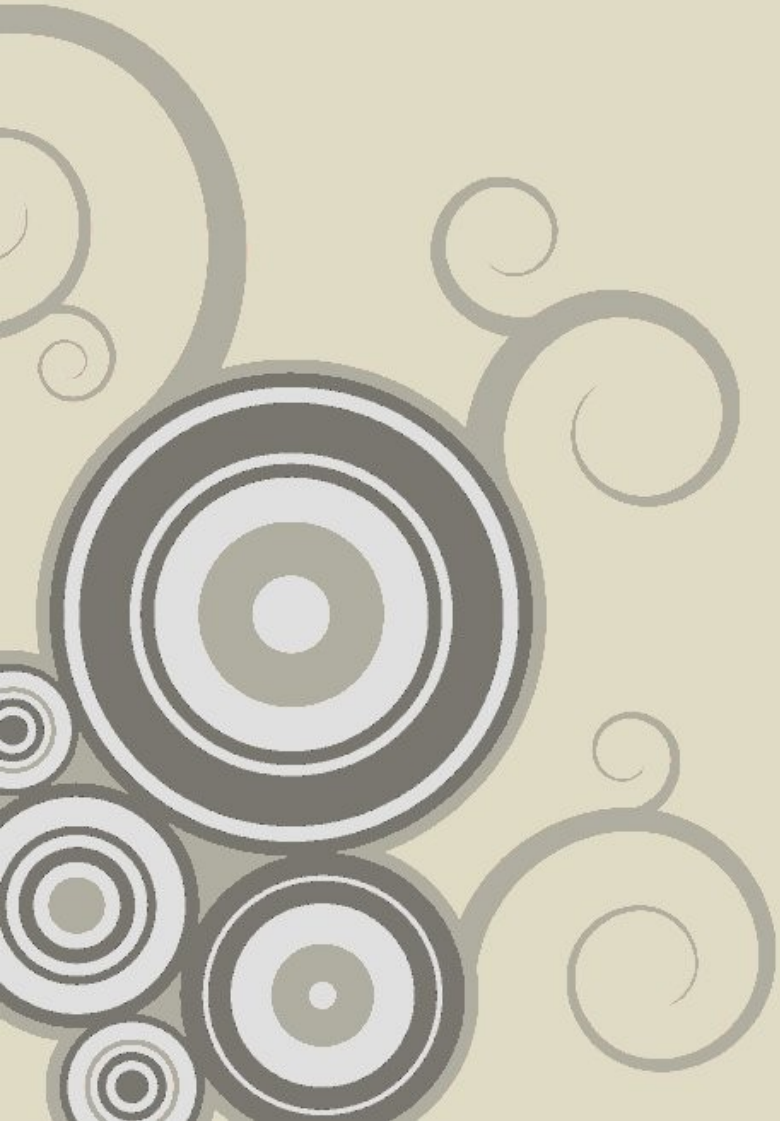
Laboratorio de Computación IV



Clase 12

Andrés Fortier

Repaso

- CRUD completo.
- *Helpers*.

Blog – validaciones

- La implementación de *ActiveRecord* de Rails provee validaciones.
- Las validaciones se definen en el modelo mismo.

/app/models/article.rb

```
class Article < ActiveRecord::Base
  validates :title,
              presence: true,
              length: { minimum: 5 }
end
```

Blog - Modelo

- En una consola

```
$ bin/rails console
> Article.new.save!
   (0.1ms)  begin transaction
   (0.1ms)  rollback transaction
ActiveRecord::RecordInvalid: Validation failed: Title can't
be blank, Title is too short (minimum is 5 characters)
   from /home/andres/.../validations.rb:57:in `save!'
>
```

Blog - Modelo

- Probemos ahora

```
$ bin/rails console
> Article.new.save
   (0.2ms)  begin transaction
   (0.1ms)  rollback transaction
 => false
>
```

Blog - Modelo

- Probemos ahora

```
$ bin/rails console
> articulo = Article.new
 => #<Article id: nil, title: nil, text: nil, created_at:
nil, updated_at: nil>
> articulo.save
   (0.2ms)  begin transaction
   (0.1ms)  rollback transaction
 => false
> articulo.errors
 => #<ActiveModel::Errors:0x000000034a3550 @base=#<Article
id: nil, title: nil, text: nil, created_at: nil, updated_at:
nil>, @messages={:title=>["can't be blank", "is too short
(minimum is 5 characters)"]}>
```

Blog – Modelo

- Las validaciones son bastante amplias
  - http://guides.rubyonrails.org/v4.1.8/active_record_validations.html
- Deben tenerlo presente al definir un modelo.
- ¿Cuánto validar?
- ¿Tests de unidad? / ¿Tests de integración?
- Cuidado con las migraciones.

Blog - Artículos

- Recordemos nuestro controller

/app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController

  ...

  def update
    @article = Article.find(params[:id])

    if @article.update(article_params)
      redirect_to @article
    else
      render 'edit'
    end
  end

end
```

Blog - Artículos

/app/views/articles/edit.html.erb

```
<%= form_for :article, url: article_path(@article),
method: :patch do |f| %>

  <% if @article.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@article.errors.count, "error") %>
prohibited
      this article from being saved:</h2>
    <ul>
    <% @article.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
  <% end %>
  ...
```

Blog – Artículos

- Probemos editar un artículo

Blog – Factorizar código

- Páginas de creación y editar casi iguales
- Rails permite reutilizar vistas con el concepto de vistas parciales.
- Por convención comienzan con guión bajo (_)

Blog - Factorizar código

/app/views/articles/**_form.html.erb**

```
<%= form_for @article do |f| %>
  <% if @article.errors.any? %>
  <div id="error_explanation">
    <h2><%= pluralize(@article.errors.count, "error") %>
prohibited
      this article from being saved:</h2>
    <ul>
    <% @article.errors.full_messages.each do |msg| %>
      <li><%= msg %></li>
    <% end %>
    </ul>
  </div>
  <% end %>
  <p>
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  ...
```

Blog - Factorizar código

/app/views/articles/_form.html.erb

```
  ...
    <%= f.label :title %><br>
    <%= f.text_field :title %>
  </p>

  <p>
    <%= f.label :text %><br>
    <%= f.text_area :text %>
  </p>

  <p>
    <%= f.submit %>
  </p>
<% end %>
```

Blog - Factorizar código

- Modificamos las vistas

/app/views/articles/new.html.erb

```
<h1>New article</h1>

<%= render 'form' %>

<%= link_to 'Back', articles_path %>
```

/app/views/articles/edit.html.erb

```
<h1>Editing article</h1>

<%= render 'form' %>

<%= link_to 'Back', articles_path %>
```

Blog - Factorizar código

- Y el *controller*

/app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController

  def update
    @article = Article.find(params[:id])

    begin
      @article.update!(article_params)
      redirect_to @article
    rescue ActiveRecord::RecordInvalid
      render 'edit'
    end
  end

end
```

Blog - Factorizar código

/app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController
  def new
    @article = Article.new
  end

  def create
    @article = Article.new(article_params)

    begin
      @article.save!
      redirect_to @article
    rescue ActiveRecord::RecordInvalid
      render 'new'
    end
  end
end
```

Rails layouts

- Un layout es una diseño que se aplica a un conjunto de páginas.
- Hay uno creado por Rails que se aplica a todas las vistas
- Se pueden definir múltiples layouts y aplicarse en diferentes casos
  - Ej. por controller.
  - http://guides.rubyonrails.org/layouts_and_rendering.html

Rails layouts

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

```
<!DOCTYPE html>
<html>
<head>
  <title>Ejemplorails</title>
  <%= stylesheet_link_tag    'application', media: 'all',
'data-turbolinks-track' => true %>
  <%= javascript_include_tag 'application', 'data-
turbolinks-track' => true %>
  <%= csrf_meta_tags %>
</head>
<body>

<%= yield %>

</body>
</html>
```

Rails layouts

- Hagan un pequeño cambio

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

```
...
  <body>
  <h1> Ejemplo de sitio en Rails </h1>
  <%= yield %>

  </body>
</html>
```

- Y naveguen el sitio para verlo aplicado.
- Deshagan el cambio.

Rails flash

- `flash` es un diccionario compartido por los componentes de una acción.
- Es una forma sencilla de pasar información básica.
- Generalmente se utiliza para pasar mensajes del controlador a la vista.

Rails flash

- Varamos a nuestro controller

/app/controllers/articles_controller.rb

```
class ArticlesController < ApplicationController

  def index
    flash[:notice] = "This is a notice"
    @articles = Article.all
  end

end
```

Rails flash

- Y en nuestra vista

/app/views/articles/index.html.erb

```
<% if flash[:notice] %>
  <div><%= flash[:notice] %></div>
<% end %>

<h1>Articles</h1>

<% @articles.each do |article| %>
  <h3>
    <%= link_to article.title, article_path(article) %>
  </h3>
<% end %>

<%= link_to 'New article', new_article_path %>
```

Rails flash

- Vayan a http://localhost:3000/
- Podemos generalizar esto y llevarlo al layout!
  - Eliminar el cambio del index
  - Llevarlo al layout

Rails flash

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

```
...
  <body>
    <% if flash[:notice] %>
      <div><%= flash[:notice] %></div>
    <% end %>

    <%= yield %>

  </body>
</html>
```

- Vayan a http://localhost:3000/

Rails flash

- Una vuelta de rosca

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

```
...
  <body>
  <% flash.each do |name, msg| %>
    <%= content_tag
          :div,
          msg,
          :class => "flash_#{name}" %>
    <% end %>

    <%= yield %>

  </body>
</html>
```

Rails flash

- Volver al index
- Eliminar “This is a notice” del controller.

Blog - Autenticación

- Agreguemos la gema

/Gemfile

```
..
gem 'devise', '~> 3.4.1'
...
```

```
$ bundle install
```

```
$ rails generate devise:install
   create  config/initializers/devise.rb
   create  config/locales/devise.en.yml
   ...
```

Blog - Autenticación

- Agregar configuración para el mailer

config/environments/development.rb

```
...

config.action_mailer.default_url_options = { host:
'localhost', port: 3000 }


...
```

```
$ rails generate devise User
   invoke  active_record
   create    db/migrate/20150503182243_devise_create_users.rb
   create    app/models/user.rb
   insert    app/models/user.rb
    route  devise_for :users
```

Blog - Autenticación

```
$ bin/rake db:migrate
== 20150503182243 DeviseCreateUsers: migrating ==============
-- create_table(:users)
   -> 0.0171s
-- add_index(:users, :email, {:unique=>true})
   -> 0.0007s
-- add_index(:users, :reset_password_token, {:unique=>true})
   -> 0.0008s
== 20150503182243 DeviseCreateUsers: migrated (0.0189s) =====
```

Blog - Autenticación

```
$ cat db/schema.rb
ActiveRecord::Schema.define(version: 20150503182243) do
  create_table "articles", force: true do |t|
    ...
  end
  create_table "users", force: true do |t|
    t.string   "email",                  default: "", null: false
    t.string   "encrypted_password", default: "", null: false
    t.string   "reset_password_token"
    t.datetime "reset_password_sent_at"
    t.datetime "remember_created_at"
    t.integer  "sign_in_count",          default: 0,  null: false
    t.datetime "current_sign_in_at"
    t.datetime "last_sign_in_at"
    t.string   "current_sign_in_ip"
    t.string   "last_sign_in_ip"
    t.datetime "created_at"
    t.datetime "updated_at"
  end
    ...
end
```

Blog – Autenticación

- Reinicien el servidor.
- Vayan al índice.
  - ¡No cambió nada!
- Debemos indicar explícitamente que páginas queremos proteger
- Recordemos que nuestro controller es (y en general todos los que creemos serán) subclase de *ApplicationController*.

Blog - Autenticación

- Agreguemos autenticación

/app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base
  # Prevent CSRF attacks by raising an exception.
  # For APIs, you may want to use :null_session instead.
  protect_from_forgery with: :exception

  before_action :authenticate_user!

end
```

- Vayan nuevamente al home

Tarea para el hogar

- Agregar una validación para el texto.
- Sacar los errores del form de edición/creación y hacer que use flash.
- Definir una página root que no sea el índice de artículos. Colocar el texto “Welcome to my amazing blog”.
- Dar acceso a usuarios no logueados al índice.

Tarea para el hogar

- Modificar el layout para que muestre
  - Ususario logueado: <mail> [Link logout]
  - Ususario no logueado: [Link login]

Links

- http://guides.rubyonrails.org/v4.1.8/active_record_validations.html
- http://guides.rubyonrails.org/v4.1.8/routing.html
- http://api.rubyonrails.org/classes/ActionDispatch/Flash.html
- http://guides.rubyonrails.org/layouts_and_rendering.html

Links

- https://github.com/plataformatec/devise
- http://apidock.com/rails/v4.0.2/AbstractController/Callbacks/ClassMethods/skip_before_action
- http://stackoverflow.com/questions/26268351/how-to-make-a-public-page-in-an-app-that-uses-devise