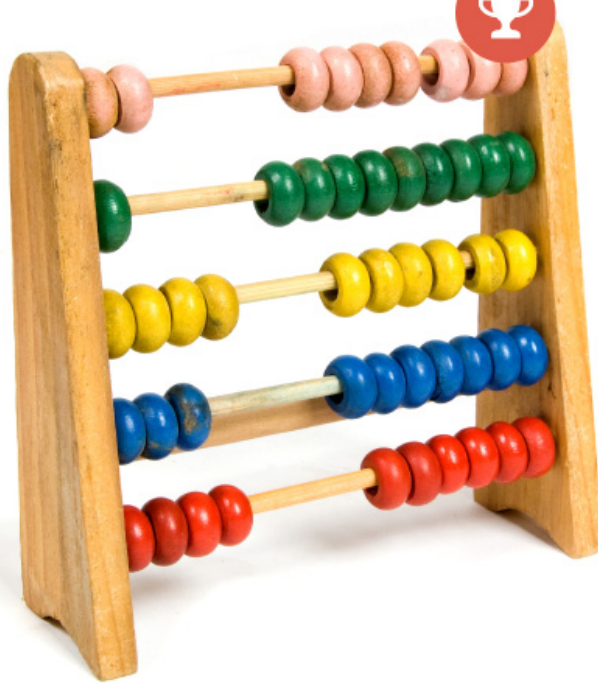


Programación con Ruby on Rails

Tienda (IX)

Delio Tolivia





Indice

Incorporando usuarios

Creando un usuario

Autenticación

Limitando acceso

Enlaces a administración

Consola de Rails

Arreglando el borrado

Incorporando usuarios (I)

- Vamos a crear un sistema de autenticación de usuarios administradores. Creamos el scaffold para los usuarios

rails generate scaffold User name:string password:digest

rake db:migrate

- ***digest*** se utiliza para en vez de guardar la contraseña como string se guarda un valor hash, de tal forma que si nuestra base de datos se ve comprometida no sea posible acceder a las contraseñas directamente.

Incorporando usuarios (II)

- Vamos a asegurarnos que el nombre de usuario es único modificando app/models/user.rb

```
class User < ApplicationRecord  
  validates :name, presence: true, uniqueness: true  
  has_secure_password  
end
```

- **has_secure_password** se añade automáticamente al indicar **password: digest**.
Genera el formulario con dos campos para comprobar la contraseña

Incorporando usuarios (III)

- Para poder encriptar la contraseña tenemos que descomentar en el Gemfile la gema bcrypt-ruby

gem 'bcrypt', '~> 3.1.7'

- Una vez hecho eso la instalamos con

bundle install

- Y finalmente reiniciamos el servidor

Incorporando usuarios (IV)

- Modificamos el controlador de user para que al terminar de crear un usuario dirija al index de users y no a mostrar el usuario tanto en ***create()*** como en ***update()***.
Modificamos en ambas funciones la línea

```
format.html { redirect_to users_url, notice: 'User #{user.name} was successfully created.' }
```

- También hacemos que index devuelva los usuarios ordenados por nombre

```
def index  
  @users = User.order(:name)  
end
```

Incorporando usuarios (V)

- Modificamos la vista app/views/users/index.html.erb

```
<h1>Listing Users</h1>  
<% if notice %>  
<p id="notice"><%= notice %></p>  
<% end %>  
  
.....
```

Incorporando usuarios (VI)

- Modificamos el formulario app/views/users/_form.html.erb (I)

```
<div class="depot_form">
<%= form_with(model: user, local: true) do |form| %>
  <% if user.errors.any? %>
    <div id="error_explanation">
      <h2><%= pluralize(user.errors.count, "error") %> prohibited this user from being saved:</h2>
      <ul>
        <% user.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>
</div>
<% end %>
<fieldset>
  <legend>Enter User Details</legend>
```


Incorporando usuarios (VI)

- Modificamos el formulario app/views/users/_form.html.erb (II)

```
<div class="field">
  <%= form.label :name , 'Name' %>
  <%= form.text_field :name, size:40 %>
</div>
<div class="field">
  <%= form.label :password, 'Password' %>
  <%= form.password_field :password, size:40 %>
</div>
<div class="field">
  <%= form.label :password_confirmation, 'Confirm' %>
  <%= form.password_field :password_confirmation, size:40 %>
</div>
<div class="actions">
  <%= form.submit %>
</div>
</fieldset>
<% end %>
</div>
```

Incorporando usuarios (VII)

- Ya podemos probar el formulario entrando en <http://localhost:3000/users/new>



The screenshot shows the Pragmatic Bookshelf website. The header is green with the logo on the left and the text 'PRAGMATIC BOOKSHELF' on the right. On the left side, there is a dark green sidebar titled 'Your Cart' which shows '1x CoffeeScript \$36.00' and a 'Total \$36.00'. Below the cart, there are links for 'Home', 'Questions', 'News', and 'Contact', and buttons for 'Check Out' and 'Empty cart'. The main content area is titled 'New User' and contains a form titled 'Enter User Details'. The form has three input fields for 'Name', 'Password', and 'Confirm', and a 'Create User' button. A 'Back' link is located below the form.

- En Rails 5 tenemos que modificar `app/controllers/application_controller.rb`

include CurrentCart
before_action :set_cart

Creando un usuario

- Ahora creamos un usuario con nuestro formulario. Al terminar nos redirige al index de usuarios y nos muestra un mensaje confirmando su creación.
- Podemos comprobar que ha sido creado correctamente en la base de datos

```
sqlite3 -line db/development.sqlite3 "select * from users"
```

```
id = 1
```

```
name = delio
```

```
password_digest = $2a$10$rhkL4cygsID32DYzIfStQOEcep7t.FpsPHQXqaeCLKh7WM30O.Hli
```

```
created_at = 2018-01-31 08:07:51.713546
```

```
updated_at = 2018-01-31 08:07:51.713546
```

Autenticación (I)

- Para la autenticación necesitamos:
 - Un formulario para que el usuario introduzca su nombre y contraseña
 - Guardar en la sesión sus datos o hasta que haga log out
 - Restringir el acceso a las partes de administración de la aplicación

Autenticación (II)

- Vamos a crear dos controladores, uno para manejar todo el proceso de logging y otro para manejar el tema de administración

rails generate controller Sessions new create destroy

rails generate controller Admin index

Autenticación (III)

- En la acción ***create()*** de SessionController vamos a guardar el Id del usuario actual en la sesión. Modificamos app/controllers/sessions_controller.rb

def create

user = User.find_by(name: params[:name])

if user.try(:authenticate, params[:password]) #try por si parametro nil

session[:user_id] = user.id

redirect_to admin_url

else

redirect_to login_url, alert: "Invalid user/password combination"

end

end

Autenticación (IV)

- Para el controlador que acabamos de modificar necesitamos un formulario de entrada. Para ello modificamos app/views/sessions/new.html.erb

```
<div class="depot_form">
  <%if flash[:alert]%>
    <p id="notice"><%= flash[:alert] %>
  <%end%>
  <%= form_tag do %>
    <fieldset>
      <legend>Please Log In</legend>
      <div>
        <%= label_tag :name, 'Name'%>
        <%= text_field_tag :name, params[:name]%>
      </div>
      <div>
        <%= label_tag :password, 'Password'%>
        <%= password_field_tag :password , params[:password]%>
      </div>
      <div>
        <%= submit_tag "Login"%>
      </div>
    </fieldset>
  <%end%>
</div>
```

Autenticación (V)

- Este último formulario no es como los utilizados hasta ahora que estaban enlazados a un modelo. En este caso usamos `form_tag` en vez de `form_for`. Esto genera un formulario HTML con varios INPUT en su interior de tal forma que en cada uno podemos asociar sus valores a params para luego recuperarlos.
- El método `destroy()` de nuestro controlador es muy simple en `app/controllers/sessions_controller.rb`

```
def destroy  
  session[:user_id] = nil  
  redirect_to store_url, notice: "Logged out"  
end
```


Autenticación (VI)

- Vamos a crear la pagina index que verán los administradores al entrar. Eso lo hacemos en app/views/admin/index.html.erb

`<h1>Welcome</h1>`

`It's <%= Time.now%>`

`We have <%= pluralize(@total_orders , "order") %>`

- En la vista utilizamos la variable total_orders que inicializamos en nuestro controlador app/controllers/admin_controller.rb

`def index`

`@total_orders = Order.count`

`end`

Autenticación (VII)

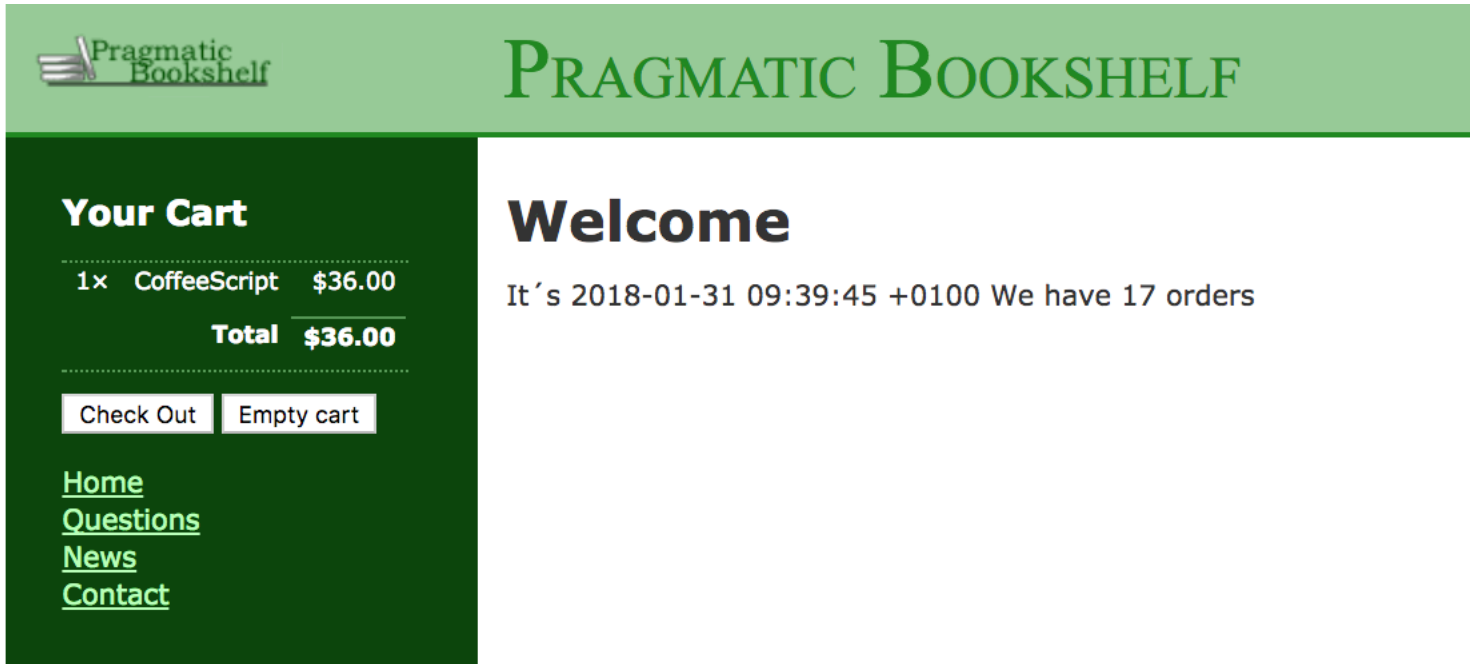
- Como no hemos creado el scaffold Rails no tiene definidas las rutas para la navegación de lo que acabamos de crear. Por ello tenemos que modificar /config/routes

```
get 'admin' => 'admin#index'  
controller :sessions do  
  get 'login' => :new  
  post 'login' => :create  
  delete 'logout' => :destroy  
end
```

- Quitamos la parte “index” de /admin/index para entrar en la parte de administración y las acciones de login las dejamos en login (en vez de session/create y session/new). Lo mismo con el logout.

Autenticación (VIII)

- Ya podemos entrar en la página index de administración con <http://localhost:3000/admin>



The screenshot shows the Pragmatic Bookshelf website. The header is green with the logo on the left and the text "PRAGMATIC BOOKSHELF" on the right. Below the header, there are two main sections. On the left, a dark green sidebar contains the "Your Cart" section, which lists "1x CoffeeScript \$36.00" and a "Total \$36.00". Below the cart, there are links for "Home", "Questions", "News", and "Contact". On the right, a white section contains a "Welcome" heading and a message: "It's 2018-01-31 09:39:45 +0100 We have 17 orders".

Pragmatic Bookshelf

PRAGMATIC BOOKSHELF

Your Cart

1x	CoffeeScript	\$36.00
Total		\$36.00

[Check Out](#) [Empty cart](#)

[Home](#)
[Questions](#)
[News](#)
[Contact](#)

Welcome

It's 2018-01-31 09:39:45 +0100 We have 17 orders

Limitando acceso (I)

- Vamos a utilizar los callbacks de Rails. Esto nos permite capturar las peticiones a un método y obligar a realizar alguna acción previa. En nuestro caso utilizaremos el callback ***before_action*** de forma que antes de realizar cualquier llamada en nuestro controlador de admin compruebe si en sesión está el user_id. Esto lo vamos a realizar en app/controllers/application_controller.rb

```
class ApplicationController < ActionController::Base  
  before_action :authorize  
  .....  
  protected  
  
  def authorize  
    unless User.find_by(id: session[:user_id])  
      redirect_to login_url, notice: "Please log in"  
    end  
  end  
end
```

Limitando acceso (II)

- Tal y como lo hemos realizado limitamos el acceso a todas las partes de la aplicación a solamente los usuarios administradores. Vamos a indicar cuales son los métodos que no tendrán que realizar este control mediante lo que se conoce como whitelisting mediante ***skip_before_action()*** en los controladores StoreController y SessionController permitiendo acceso a todos los métodos con

skip_before_action :authorize

- En el caso de CartsController tenemos que dar acceso a ***create***, ***update*** y ***delete***

skip_before_action :authorize, only: [:create, :update, :destroy]

Limitando acceso (III)

- En LineItemsController daremos permiso a **create**

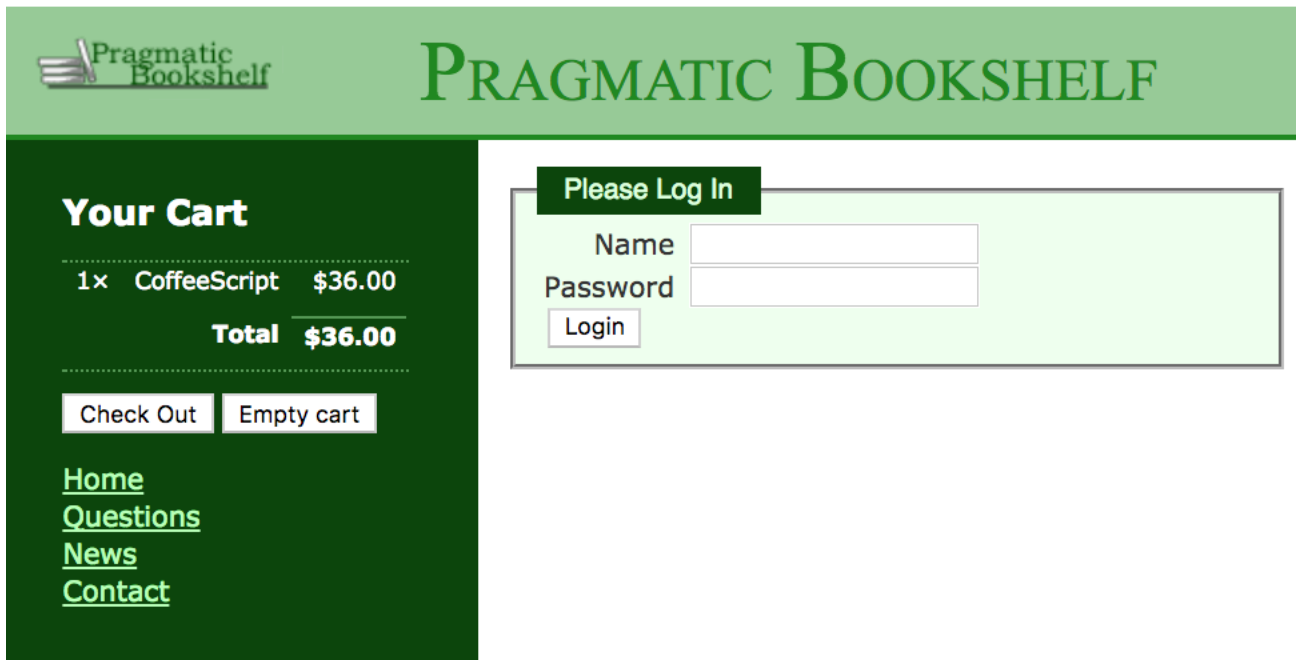
skip_before_action :authorize, only: :create

- Y en OrdersController a **new** y **create**

skip_before_action :authorize, only: [:new, :create]

Limitando acceso (IV)

- Ahora si navegamos a <http://localhost:3000/products> que es nuestro listado de productos ya nos mostrará la ventana de login



The screenshot displays the Pragmatic Bookshelf website interface. The header features the site's logo and name. The main content area is divided into two sections: a shopping cart on the left and a login form on the right.

Pragmatic Bookshelf

Your Cart

1x	CoffeeScript	\$36.00
Total		\$36.00

[Check Out](#) [Empty cart](#)

[Home](#)
[Questions](#)
[News](#)
[Contact](#)

Please Log In

Name

Password

Enlaces a administración (I)

- Vamos a añadir enlaces a las funciones de administración en la barra lateral en app/views/layouts/application.html.erb


.....

```
<li><a href="">Contact</a></li>
</ul>
<% if session[:user_id]>
  <ul>
    <li><%= link_to 'Orders', orders_path %></li>
    <li><%= link_to 'Products', products_path %></li>
    <li><%= link_to 'Users', users_path %></li>
  </ul>
  <%= button_to 'Logout', logout_path, method: :delete %>
<% end %>
</div>
<div id="main">
```

.....

Enlaces a administración (III)

- Con esto ya podemos logearnos y ver los enlaces en la barra lateral.



The screenshot displays the Pragmatic Bookshelf website interface. At the top, there is a green header with the Pragmatic Bookshelf logo and the site name. Below the header, the page is divided into two main sections. On the left, the 'Your Cart' section shows a single item, 'CoffeeScript', priced at \$36.00, with a total of \$36.00. It includes buttons for 'Check Out' and 'Empty cart', and a list of navigation links: Home, Questions, News, Contact, Orders, Products, Users, and a Logout button. On the right, the 'Listing Users' section shows a table with one user, 'delio', and links for 'Show', 'Edit', and 'Destroy'. Below the table is a link for 'New User'.

Pragmatic Bookshelf

Your Cart

1x	CoffeeScript	\$36.00
Total		\$36.00

[Check Out](#) [Empty cart](#)

[Home](#)
[Questions](#)
[News](#)
[Contact](#)
[Orders](#)
[Products](#)
[Users](#)
[Logout](#)

Listing Users

Name
delio

[Show](#) [Edit](#) [Destroy](#)

[New User](#)

Consola de Rails

- Podemos arrancar una consola de rails con ***rails console*** y acceder a los modelos, etc. de nuestra aplicación. Por ejemplo si borramos nuestro último usuario administrador podríamos guardarlo en la base de datos.

2.4.1 :001 > User.count

(0.5ms) SELECT COUNT(*) FROM "users"

=> 1

2.4.1 :002 > User.create(name: 'prueba', password: 'secret', password_confirmation: 'secret')

(0.1ms) begin transaction

User Exists (0.2ms) SELECT 1 AS one FROM "users" WHERE "users"."name" = ?

LIMIT ? [{"name", "prueba"}, ["LIMIT", 1]]

SQL (0.8ms) INSERT INTO "users" ("name", "password_digest", "created_at", "updated_at")

VALUES (?, ?, ?, ?) [{"name", "prueba"}, ["password_digest",

"\$2a\$10\$zC8UR5DCr69ihervDibs7.9SjJqywsQ6/iNNR4ChUZMuG1kdbmh3K"],

["created_at", "2018-01-31 10:19:25.373074"], ["updated_at", "2018-01-31 10:19:25.373074"]]

(0.7ms) commit transaction

=> #<User id: 2, name: "prueba", password_digest: "\$2a\$10\$zC8UR5DCr69ihervDibs7.9SjJqywsQ6/iNNR4ChUZM...", created_at: "2018-01-31 10:19:25", updated_at: "2018-01-31 10:19:25">

2.4.1 :003 > User.count

(0.3ms) SELECT COUNT(*) FROM "users"

=> 2

Arreglando el borrado (I)

- Para evitar el borrado del último usuario vamos a utilizar un hook (un método que se lanza en algún momento de la vida de un objeto). En este caso el ***after_destroy()***. En `app/models/user.rb`

```
after_destroy :ensure_an_admin_remains  
private  
def ensure_an_admin_remains  
  if User.count.zero?  
    raise "Can't delete last user"  
  end  
end
```

Arreglando el borrado (II)

- El lanzar una excepción automáticamente produce un rollback si se lanza dentro de una transición. Y además podemos capturarla en el controlador. En app/controllers/users_controller.rb

```
def destroy  
  begin  
    @user.destroy  
    flash[:notice] = "User #{@user.name} deleted"  
  rescue StandardError => e  
    flash[:notice] = e.message  
  end
```

.....