

AUTENTICACIÓN

**AUTENTICACIÓN CONSISTE EN VALIDAR
SI UN USUARIO ES QUIEN DICE SER.**

AUTENTICACIÓN DESDE CERO

**ANTES DE ABORDAR AUTENTICACIÓN
VAMOS A VER DOS CONCEPTOS NUEVOS**

SESSION Y COOKIES

COOKIES

Las cookies son un mecanismo que nos permite guardar valores en el navegador del **cliente**

```
cookies[:user_name] = "david"  
cookies.delete(:user_name)
```

Normalmente se agregan dentro del controller y se envían al cliente durante el response

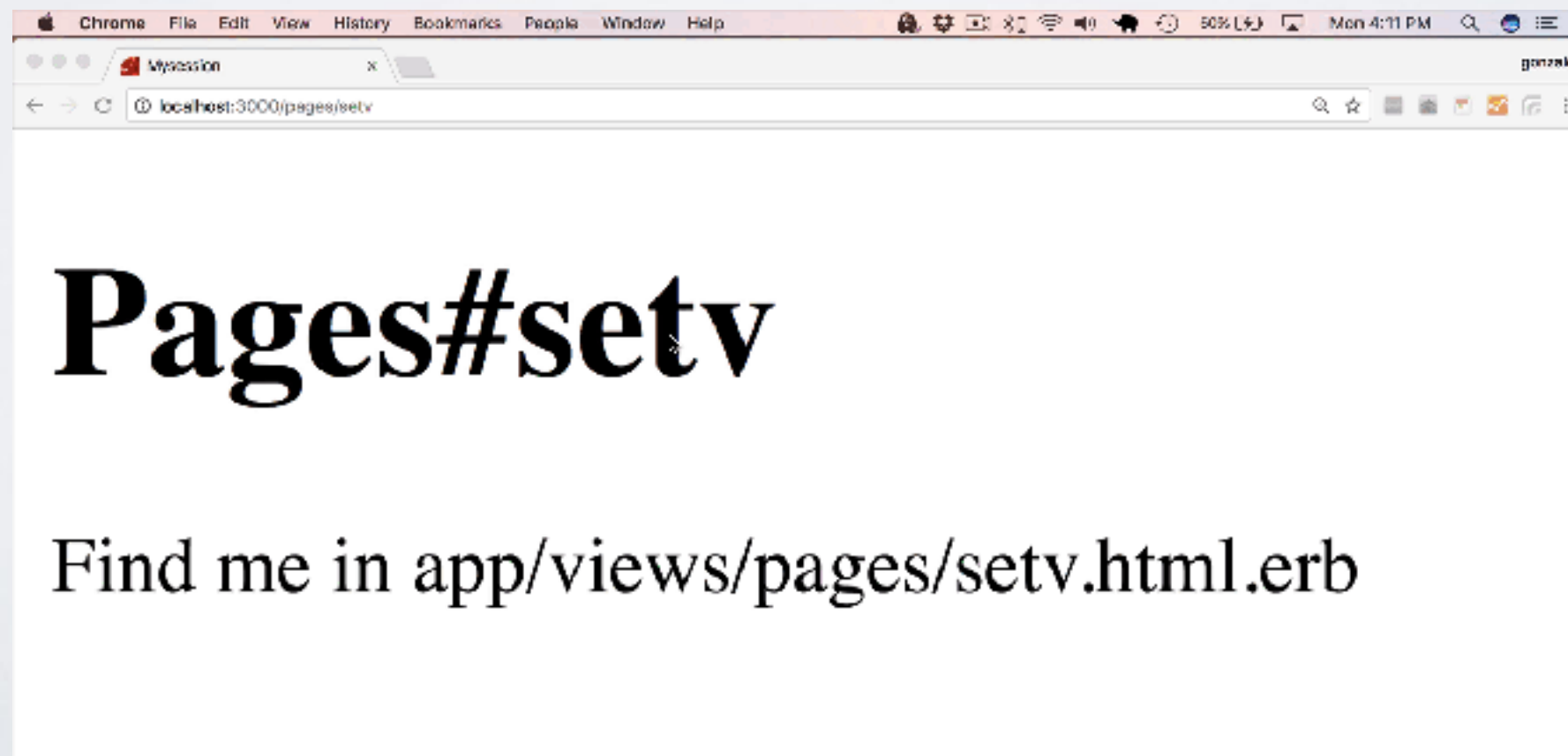
SESSION

Las sesiones son un mecanismo que nos permite guardar valores en el **servidor**, están asignados a un único usuario a través de una cookie guardada en el lado del cliente

CONSTRUYAMOS UNA APLICACIÓN SENCILLA PARA ENTENDER COMO FUNCIONAN LAS SESIONES

NUESTRA APLICACIÓN TENDRÁ 2 PÁGINAS

Una para inicializar el valor de la sesión y otro para cambiar el valor.



SEGURIDAD EN LA SESIÓN

- El concepto de seguridad asociado a cookies es complejo e innecesario para aprender a utilizarlas, pero este tutorial te ayudará a comprenderlas a fondos.
- <http://www.justinweiss.com/articles/how-rails-sessions-work/>

**UTILIZAREMOS SESIONES PARA
AUTENTICAR A NUESTROS USUARIOS**

AUTENTICACIÓN

**AUTENTICACIÓN CONSISTE EN VALIDAR
SI UN USUARIO ES QUIEN DICE SER.**

EN ESTA EXPERIENCIA CREAREMOS UN FORMULARIO DE REGISTRO Y DE LOGIN DE USUARIO

Más adelante veremos que esto se puede de forma automática con la gema **devise**

LA IMPLEMENTACIÓN QUE HAREMOS
TIENE MOTIVOS PEDAGÓGICOS Y NO ES
EXACTAMENTE IGUAL QUE LA DE DEVISE.

CONSTRUIREMOS UN SISTEMA DE AUTENTICACIÓN BÁSICO

Más adelante construiremos uno que abarque elementos importantes de seguridad como encriptación del password.

SETUP DE PROYECTO

Necesitaremos como base un proyecto nuevo con un simple scaffold y que el `root_path` apunte a ese index

CREAREMOS EL MODELO USER

```
rails g model user name email password
```

CREAREMOS EL CONTROLLER USER

`rails g controller users`

NECESITAREMOS CREAR MÉTODOS PARA:

1. Registrar un usuario nuevo (user#new)
2. Procesar el usuario nuevo registrado(user #create)
3. Entrar como usuario existente (session#new)
4. Cerrar la sesión (session#destroy)
5. Procesar la información del usuario que ingresa (session#create)

RECORDEMOS LOS PASOS QUE TENEMOS QUE PROGRAMAR POR CADA UNO DE LOS MÉTODOS

1. Agregar la ruta
2. Crear el controller (sólo si no ha sido creado)
3. Agregar el método al controller
4. Crear la vista

REGISTRAR UN USUARIO NUEVO

1) AGREGAMOS LA(S) RUTAS PARA CREAR UN USUARIO

```
get 'users/sign_up', to: "users#new"  
post 'users', to: "users#create"
```

2) EL CONTROLLER YA LO CREAMOS

3) AGREGAMOS EL(LOS) MÉTODOS AL CONTROLLER

```
class UsersController < ApplicationController
  def new
    @user = User.new
  end

  def create
    @user = User.new(user_params)
    if @user.save
      redirect_to root_path
    else
      render :new
    end
  end

  def user_params
    params.require(:user).permit(:email, :password)
  end
end
```

4) CREAR UNA VISTA DE REGISTRO, ESTA SERÁ LA VISTA NEW DE USER

```
<h1>Sign Up</h1>
<%= form_with(model: @user, local: true) do |f| %>
  <% if @user.errors.any? %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% @user.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </div>
  <div class="field">
    <%= f.label :password %><br />
    <%= f.password_field :password %>
  </div>
  <div class="actions"><%= f.submit "Sign Up" %></div>
<% end %>
```

CREANDO LA SESIÓN

Para eso vamos a guardar el **user_id** dentro de la sesión, después de crear el usuario.

```
class UsersController < ApplicationController
  def create
    @user = User.new(user_params)
    if @user.save
      session[:user_id] = @user.id
      redirect_to root_path
    else
      render :new
    end
  end
end
```

AHORA CREAREMOS MÉTODOS PARA OBTENER EL OBJETO USUARIO A PARTIR DE LA SESIÓN

```
module UsersHelper
  def current_user
    User.find(session[:user_id])
  end

  def logged?
    session[:user_id].present? ? true : false
  end
end
```

LOS HELPERS SE CARGAN AUTOMÁTICAMENTE EN LAS VISTAS

Aprovecharemos esto para mostrar el mail del
usuario ingresado en el layout

CURRENT_USER

AGREGANDO EL MAIL EN EL LAYOUT

```
<!DOCTYPE html>
<html>
  <head>
    <title>Authscratch</title>
    <%= csrf_meta_tags %>

    <%= stylesheet_link_tag 'application', media: 'all', 'data-turbolinks-track': 'reload' %>
    <%= javascript_include_tag 'application', 'data-turbolinks-track': 'reload' %>
  </head>

  <body>
    <% if logged? %>
      Bienvenido <%= current_user.email %>
    <% end %>

    <%= yield %>
  </body>
</html>
```

NECESITAREMOS CREAR MÉTODOS PARA:

1. ~~Registrar un usuario nuevo (user#new)~~
2. ~~Procesar el usuario nuevo registrado (user #create)~~
3. Cerrar la sesión (session#destroy)
4. Entrar como usuario existente (session#new)
5. Procesar la información del usuario que ingresa

**YA TENEMOS UN USUARIO INGRESADO,
ASÍ QUE CREAREMOS EL SALIR PRIMERO**

RECORDAMOS LOS PASOS POR CADA MÉTODOS

1. Agregar la ruta
2. Crear el controller (sólo si no ha sido creado)
3. Agregar el método al controller
4. Crear la vista

1) AGREGAMOS LA(S) RUTAS PARA CREAR UNA SESIÓN

```
resources :posts  
root 'posts#index'  
get 'users/sign_up', to: 'users#new'  
post 'users', to: 'users#create'  
resources :sessions, only: [:create, :destroy]
```

2) CREAREMOS EL CONTROLLER SESSION

rails g controller sessions

3) AGREGAMOS EL MÉTODO DESTROY EN EL CONTROLLER

```
class SessionsController < ApplicationController
  def destroy
    reset_session
    redirect_to root_path
  end
end
```

4) NO NECESITAMOS VISTAS
PORQUE ESTE MÉTODO
REDIRECCIONA

AGREGAREMOS UN LINK PARA QUE EL USUARIO PUEDA CERRAR SU SESIÓN

```
<%= link_to "salir", session_path(current_user), method: :delete %>
```

NECESITAREMOS CREAR MÉTODOS PARA:

1. ~~Registrar un usuario nuevo (user#new)~~
2. ~~Procesar el usuario nuevo registrado (user #create)~~
3. ~~Cerrar la sesión (session#destroy)~~
4. Entrar como usuario existente (session#new)
5. Procesar la información del usuario que ingresa (session#create)

RECORDAMOS LOS PASOS POR CADA MÉTODOS

1. Agregar la ruta
2. Crear el controller (sólo si no ha sido creado)
3. Agregar el método al controller
4. Crear la vista

1) AGREGAMOS LA(S) RUTAS PARA CREAR UN USUARIO

```
resources :posts  
root 'posts#index'  
get 'users/sign_up', to: 'users#new'  
post 'users', to: 'users#create'  
get 'users/sign_in', to: 'sessions#new'
```

```
resources :sessions, only: [:create, :destroy]
```

2) CREAR EL CONTROLLER

Pero ya lo tenemos creado :)

3) AGREGAMOS EL MÉTODO NEW EN EL CONTROLLER

```
class SessionsController < ApplicationController
  def new
    @user = User.new
  end

  def destroy
    reset_session
    redirect_to root_path
  end
end
```

4) CREAMOS EL FORMULARIO DE INGRESO

```
<h1>Sign In</h1>
<%= form_with(model: @user, url: sessions_path, method: :post, local: true) do |f| %>
  <% if @user.errors.any? %>
    <div class="error_messages">
      <h2>Form is invalid</h2>
      <ul>
        <% @user.errors.full_messages.each do |message| %>
          <li><%= message %></li>
        <% end %>
      </ul>
    </div>
  <% end %>

  <div class="field">
    <%= f.label :email %><br />
    <%= f.text_field :email %>
  </div>
  <div class="field">
    <%= f.label :password %><br />
    <%= f.password_field :password %>
  </div>
  <div class="actions"><%= f.submit "Sign In" %></div>
<% end %>
```

NECESITAREMOS CREAR MÉTODOS PARA:

1. ~~Registrar un usuario nuevo (user#new)~~
2. ~~Procesar el usuario nuevo registrado (user #create)~~
3. ~~Cerrar la sesión (session#destroy)~~
4. ~~Entrar como usuario existente (session#new)~~
5. Procesar la información del usuario que ingresa (session#create)

1) AGREGAMOS LA RUTA

Esta ya la habíamos adelantado

```
resources :posts
root 'posts#index'
get 'users/sign_up', to: 'users#new'
post 'users', to: 'users#create'
get 'users/sign_in', to: 'sessions#new'

resources :sessions, only: [:create, :destroy]
```

2) CREAR EL CONTROLLER

Pero ya lo tenemos creado :)

3) AGREGAMOS EL MÉTODO EN EL CONTROLLER

Este método tiene que buscar al usuario en la base de datos con el nombre y el password, si se encuentra iniciamos sesión con ese usuario, en caso contrario lo redirigimos

3) AGREGAMOS EL MÉTODO EN EL CONTROLLER

```
def create
  @user = User.find_by(email: params[:user][:email],
                        password: params[:user][:password])

  if @user.present?
    session[:user_id] = @user.id
    redirect_to root_path, notice: 'Has ingresado'
  else
    redirect_to root_path, alert: 'Tus credenciales no son válidas'
  end
end
```

**4) NO REQUERIMOS VISTA,
PORQUE EL MÉTODO
REDIRECCIONA.**

**AHORA VAMOS A IMPEDIR QUE
EL USUARIO INGRESE A /POSTS
SIN ESTAR AUTENTICADO**

PARA ESTO VAMOS A CONSTRUIR UN MÉTODO QUE NOS AYUDARÁ

```
class ApplicationController < ActionController::Base
  protect_from_forgery with: :exception

  private
  def authenticate_user!
    redirect_to users_sign_in_path unless helpers.logged?
  end
end
```

AHORA ESTE MÉTODO LO PODEMOS OCUPAR EN LOS CALLBACKS DE LOS CONTROLLER

```
class PostsController < ApplicationController
  before_action :set_post, only: [:show, :edit, :update, :destroy]
  before_action :authenticate_user!
end
```

SI SOLO QUEREMOS FILTRAR SOLO
EN CIERTOS MÉTODOS PODEMOS
OCUPAR :ONLY O :EXCEPT

**AHORA PODEMOS ASIGNAR LOS
POSTS CREADOS AL USUARIO
INGRESADO.**

TENEMOS MODELO DE POST Y DE USUARIO, PERO NOS FALTA UNA MIGRACIÓN Y LAS RELACIONES

```
rails g migration addUserToPost user:references
```

ASIGNAMOS EL POST AL USUARIO ACTUAL

```
def create
  @post = Post.new(post_params)
  @post.user = helpers.current_user

  respond_to do |format|
    if @post.save
      format.html { redirect_to @post, notice: 'Post was successfully created.' }
      format.json { render :show, status: :created, location: @post }
    else
      format.html { render :new }
      format.json { render json: @post.errors, status: :unprocessable_entity }
    end
  end
end
```

**Y AHORA PODEMOS MOSTRAR
INFORMACIÓN DEL USUARIO EN
EL POST**

YA SABEMOS LO BÁSICO DE AUTENTICACIÓN

Ahora necesitamos aprender un poco de seguridad

NOS INTERESA QUE:

- Personas del equipo de desarrollo (o con acceso a la base de datos) no puedan ver los password de los usuarios.
- Si por algún motivo los datos de tu base de datos quedan comprometidos, los passwords de los usuarios estarán seguros.

PARA APRENDER ESTO VAMOS A TENER QUE INTRODUCIR 3 CONCEPTOS NUEVOS

- Atributos virtuales
- Callbacks en los modelos
- Encriptación

**PARA APRENDER ESTO TRABAJAREMOS
EN UN PROYECTO NUEVO**

SETUP DEL PROYECTO

Necesitaremos como base un proyecto nuevo con un simple scaffold y que el `root_path` apunte a ese index

CREAREMOS EL MODELO USER

```
rails g model user email password_digest
```

¿POR QUÉ PASSWORD_DIGEST?

Por que aquí guardaremos el password encriptado

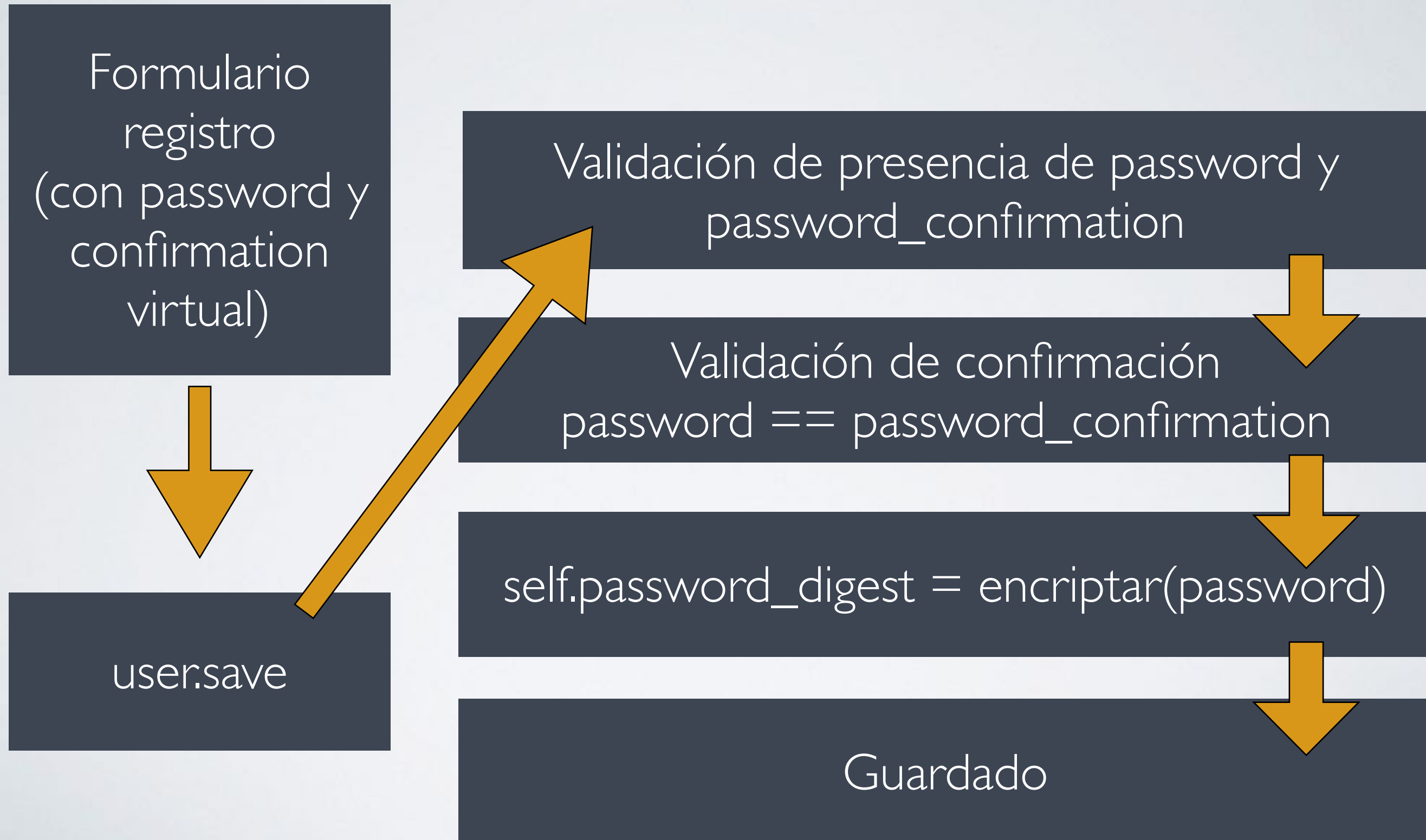
**PARA HACER ESO,
INTRODUCIREMOS EL CONCEPTO
DE ATRIBUTO VIRTUAL**

ATRIBUTOS VIRTUAL

Un atributo virtual es un atributo que no persiste

NUESTRO PASSWORD Y
PASSWORD_CONFIRMATION
SERÁN ATRIBUTOS VIRTUALES

PASOS IMPORTANTES



**PARA ENCRYPTAR
UTILIZAREMOS BCrypt**

PARA ENCRIPtar EL PASSWORD UTILIZAREMOS BCrypt

```
gem 'bcrypt', '~> 3.1.7'
```

ESTA GEMA NOS AYUDARA PARA LA ENCRIPtACIÓN DE
LAS CLAVES DE AUTENTICACIÓN DEL USUARIO

PARA ENCRIPtar EL PASSWORD UTILIZAREMOS BCrypt

```
gem 'bcrypt', '~> 3.1.7'
```

ESTA GEMA NOS AYUDARA PARA LA ENCRIPtACIÓN DE
LAS CLAVES DE AUTENTICACIÓN DEL USUARIO

```
bundle
```

BCRYPT PUEDE HACER TODO ESTO AUTOMÁTICAMENTE

Basta con agregar `has_secure_password` en el modelo.

Validación de presencia de password y password_confirmation



Validación de confirmación
`password == password_confirmation`



`self.password_digest = encriptar(password)`



**COMO SOMOS MALVADOS LO
VEREMOS DE FORMA MANUAL**

COMO SOMOS MALVADOS LO
VEREMOS DE FORMA MANUAL

PRIMERO AGREGAREMOS LAS VALIDACIONES EN EL MODELO

```
attr_accessor :password, :password_confirmation  
validates :password, confirmation: true  
validates :password, presence: true
```

CREAREMOS UN MÉTODO PARA GUARDAR EL PASSWORD

```
class User < ApplicationRecord

  def bcrypt_password
    self.password_digest = BCrypt::Password.create password
  end
end
```

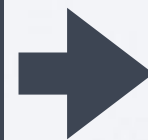
TAMBIÉN TENEMOS QUE CREAR UN MÉTODO PARA LEER EL PASSWORD

```
BCrypt::Password.create("hola")
```



Convierte un
string en hash

```
BCrypt::Password.new(valor  
guardado)
```



Utilizamos Password.new
para volver a convertirlo
en hash

TAMBIÉN TENEMOS QUE CREAR UN MÉTODO PARA LEER EL PASSWORD

```
BCrypt::Password.create("hola")
```



Convierte un
string en hash

Al guardarlo el hash se convierte en
un string



Por lo que no
podemos compararlo
con el original

```
BCrypt::Password.new(valor  
guardado)
```



Utilizamos Password.new
para volver a convertirlo
en hash

PARA HACERLO SENCILLO AGREGAREMOS UN MÉTODO AL MODELO

```
class User < ApplicationRecord
  attr_accessor :password, :password_confirmation
  validates :password, confirmation: true
  validates :password, presence: true
  before_save :bcrypt_password

  def bcrypt_password
    self.password_digest = BCrypt::Password.create password
  end
end
```

PODEMOS REVISAR EN LA DOCUMENTACIÓN
QUE HAS_SECURE_PASSWORD FUNCIONA
DE MANERA SIMILAR

**AHORA PODEMOS REMPLAZAR TODO
NUESTRO CÓDIGO EN EL MODELO
POR HAS_SECURE_PASSWORD**

DEVISE

Autenticación

PROS Y CONTRA DE DEVISE

- Pros:
 - Lo hace todo por nosotros
 - Implementar funcionalidades comunes es muy sencillo
 - Se encuentra muy bien documentada
- Contras:
 - Es una gema muy compleja que trae muchas cosas
 - Implementar algo fuera de lo común puede ser complejo

PASOS PARA LA INSTALACIÓN DE DEVISE

- Agregar la gema y hacer bundle (reiniciar el servidor)
- Correr el generador para generar los archivos de devise
- Correr el generador para generar el modelo deviseado (o sea, como se llamará el modelo que tenga autenticación, normalmente es user)
- Opcional:
 - Correr el generador para generar vistas custom del login (y otras) y poder modificarlas
 - Correr el generador para generar los controles custom de devise y poder modificar cosas como las redirecciones y strong params

LA DOCUMENTACIÓN

A screenshot of a web browser showing the GitHub repository page for 'devise'. The browser's address bar shows the URL 'https://github.com/plataformatec/devise'. The page features the 'devise' logo, which consists of the word 'devise' in a large, lowercase, sans-serif font, with four vertical bars of varying heights underneath. Below the logo, it says 'By Plataformatec.' and shows two status badges: 'build passing' and 'code climate 3.4'. The main text of the page reads: 'This README is also available in a friendly navigable format.' followed by 'Devise is a flexible authentication solution for Rails based on Warden. It:'. A bulleted list follows: '• Is Rack based;', '• Is a complete MVC solution based on Rails engines;', '• Allows you to have multiple models signed in at the same time;', and '• Is based on a modularity concept: use only what you really need.'. Below this, it says 'It's composed of 10 modules:'. A final bulleted list item is visible: '• Database Authenticatable: hashes and stores a password in the database to validate the authenticity of a user while signing in. The authentication can be done both through POST requests or HTTP Basic Authentication.'

INSTALANDO DEVISE

Agregar al gemfile

```
gem "devise", git: 'https://github.com/plataformatec/devise.git'
```

Luego en el bash

```
bundle install
```

```
rails g devise:install
```

```
rails g devise user
```


RAILS G DEVISE USER

```
>> rails g devise user
Running via Spring preloader in process 1893
  invoke  active_record
  create  db/migrate/20170615175752_devise_create_users.rb
  create  app/models/user.rb
  invoke  test_unit
  create  test/models/user_test.rb
  create  test/fixtures/users.yml
  insert  app/models/user.rb
  route  devise_for :users
```



Agrega modelo, migración
y rutas para el login

```
rake db:migrate
```

¿CÓMO HACEMOS LOGIN O SIGN_UP?

Revisemos rake routes

	Prefix	Verb	URI Pattern	Controller#Action
	new_user_session	GET	/users/sign_in(:format)	devise/sessions#new
	user_session	POST	/users/sign_in(:format)	devise/sessions#create
	destroy_user_session	DELETE	/users/sign_out(:format)	devise/sessions#destroy
	new_user_password	GET	/users/password/new(:format)	devise/passwords#new
	edit_user_password	GET	/users/password/edit(:format)	devise/passwords#edit
	user_password	PATCH	/users/password(:format)	devise/passwords#update
		PUT	/users/password(:format)	devise/passwords#update
		POST	/users/password(:format)	devise/passwords#create
	cancel_user_registration	GET	/users/cancel(:format)	devise/registrations#cancel
	new_user_registration	GET	/users/sign_up(:format)	devise/registrations#new
	edit_user_registration	GET	/users/edit(:format)	devise/registrations#edit
	user_registration	PATCH	/users(:format)	devise/registrations#update
		PUT	/users(:format)	devise/registrations#update
		DELETE	/users(:format)	devise/registrations#destroy
		POST	/users(:format)	devise/registrations#create

UNA VEZ QUE HACEMOS LOGIN ¿CÓMO SALIMOS?

No podemos salir directamente a través de una URL
porque se requiere llamar a través del verbo delete
pero podemos agregar el link

**AGREGUEMOS LOS LINKS PARA ENTRAR,
REGISTRARNOS
Y SALIR**

¿CÓMO SABER SI UN USUARIO ESTÁ LOGEADO?

`unless current_user.nil?`

El usuario al ingresarse se guarda en el objeto
`current_user`

`if user_signed_in?`

El método `user_signed_in?` devuelve true
si el usuario se encuentra ingresado

OCUPANDO UN IF PODEMOS MOSTRAR
EL LOGIN O SIGN_UP SI NO ESTÁ
INGRESADO Y UN SALIR SI LO ESTÁ

AHORA PARA QUE QUEDE BIEN PONGÁMOSLO SOBRE LA BARRA DE BOOTSTRAP

```
<ul class="nav navbar-nav navbar-right">
  <% if user_signed_in? %>
    <li> <a href="<%= destroy_user_session_path %>" data-method='delete'>
      Salir
    </a></li>
  <% else %>
    <li><a href="<%= new_user_session_path %>"> Login </a></li>
    <li><a href="<%= new_user_registration_path %>"> Registro </a></li>
  <% end %>
</ul>
```

OBLIGANDO AL USUARIO A AUTENTICARSE

Para eso ocuparemos un callback

PERSONALIZANDO LAS VISTAS

```
rails g devise:views
```

```
invoke Devise::Generators::SharedViewsGenerator
create  app/views/devise/shared
create  app/views/devise/shared/_links.html.erb
invoke  form_for
create  app/views/devise/confirmations
create  app/views/devise/confirmations/new.html.erb
create  app/views/devise/passwords
create  app/views/devise/passwords/edit.html.erb
create  app/views/devise/passwords/new.html.erb
create  app/views/devise/registrations
create  app/views/devise/registrations/edit.html.erb
create  app/views/devise/registrations/new.html.erb
create  app/views/devise/sessions
create  app/views/devise/sessions/new.html.erb
create  app/views/devise/unlocks
create  app/views/devise/unlocks/new.html.erb
invoke  erb
create  app/views/devise/mailer
create  app/views/devise/mailer/confirmation_instructions.html.erb
create  app/views/devise/mailer/password_change.html.erb
create  app/views/devise/mailer/reset_password_instructions.html.erb
create  app/views/devise/mailer/unlock_instructions.html.erb
```

REVISEMOS LAS VISTA DE LOGIN

```
invoke Devise::Generators::SharedViewsGenerator
create  app/views/devise/shared
create  app/views/devise/shared/_links.html.erb
invoke form_for
create  app/views/devise/confirmations
create  app/views/devise/confirmations/new.html.erb
create  app/views/devise/passwords
create  app/views/devise/passwords/edit.html.erb
create  app/views/devise/passwords/new.html.erb
create  app/views/devise/registrations
create  app/views/devise/registrations/edit.html.erb
create  app/views/devise/registrations/new.html.erb
create  app/views/devise/sessions
create  app/views/devise/sessions/new.html.erb
create  app/views/devise/unlocks
create  app/views/devise/unlocks/new.html.erb
invoke erb
create  app/views/devise/mailer
create  app/views/devise/mailer/confirmation_instructions.html.erb
create  app/views/devise/mailer/password_change.html.erb
create  app/views/devise/mailer/reset_password_instructions.html.erb
create  app/views/devise/mailer/unlock_instructions.html.erb
```

LA VISTA DEL LOGIN

```
<%= form_for(resource, as: resource_name, url: session_path(resource_name)) do |f| %>
  <div class="field">
    <%= f.label :email %><br />
    <%= f.email_field :email, autofocus: true %>
  </div>
```

```
  <div class="field">
    <%= f.label :password %><br />
    <%= f.password_field :password, autocomplete: "off" %>
  </div>
```

```
<% if devise_mapping.rememberable? -%>
  <div class="field">
    <%= f.check_box :remember_me %>
    <%= f.label :remember_me %>
  </div>
<% end -%>
```

```
<div class="actions">
  <%= f.submit "Log in" %>
</div>
<% end %>
```

¿Qué es esto?

¿QUÉ ES RESOURCE?

```
helper_method :resource_name, :resource, :devise_mapping

def resource_name
  :user
end

def resource
  @resource ||= User.new
end

def devise_mapping
  @devise_mapping ||= Devise.mappings[:user]
end
```

Nos ayuda a trabajar con el mismo formulario en caso de múltiples recursos deviseables, este código solo lo ocuparemos si queremos hacer otro formulario de ingreso, **lo podemos encontrar en la documentación de Devise**

FILTRANDO EL ACCESO DE UN USUARIO

Para poder filtrar y redirigir automáticamente a un usuario que no está ingresado (logeado)

En el controller

```
before_action :authenticate_user!
```

Esto filtrará el acceso a todas las páginas dentro de este controller

FILTRANDO A UNA O SÓLO ALGUNAS PÁGINAS

```
before_action :authenticate_user, only: [:pag1, :pag2]
```

```
before_action :authenticate_user, except: [:pag3, :pag4]
```

CREANDO NUESTRO PROPIO MÉTODO DE FILTRADO

Vamos a filtrar a los usuarios que no sean admin

Creamos una migración agregando admin a los usuarios

```
rails g migration addAdminToUser admin:boolean
```

Antes de correr la migración modifiquemos para que por defecto los usuarios no sean admin

```
def change
  add_column :users, :admin, :boolean, default: false
end
```


AHORA FILTRAMOS EL ACCESO A TODO AQUEL QUE NO SEA ADMIN

En el tag controller

```
before_action :filter_admin!
```

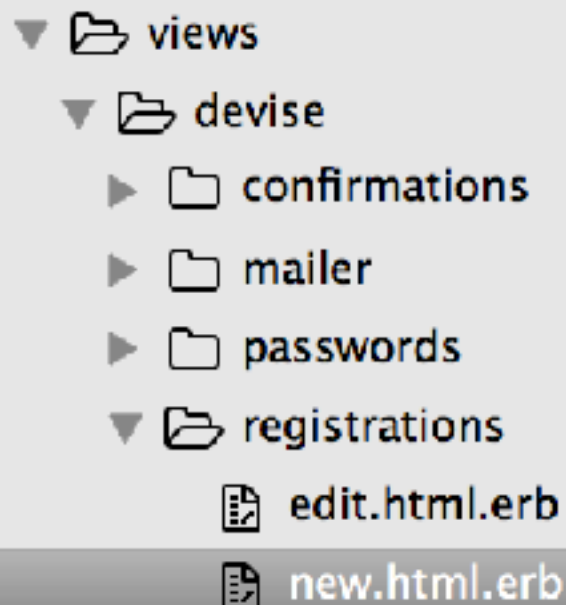
Luego creamos un método private dentro del mismo controller

```
def filter_admin!  
  authenticate_user!  
  redirect_to root_path, alert: "No tienes acceso" unless current_user.admin?  
end
```


AGREGUEMOS EL NOMBRE DE USUARIO

```
rails g migration addNameToUser name:string
```

Agregamos ahora el campo en el formulario de
sign_up

A screenshot of a file explorer showing the directory structure of a Rails application. The 'views' directory is expanded, showing subdirectories for 'devise', 'confirmations', 'mailer', 'passwords', and 'registrations'. The 'registrations' directory is further expanded, showing files 'edit.html.erb' and 'new.html.erb'.

```
▼ views
  ▼ devise
    ► confirmations
    ► mailer
    ► passwords
  ▼ registrations
    edit.html.erb
    new.html.erb
```

```
<div class="field">
  <%= f.label :name %><br />
  <%= f.text_field :name, autofocus: true %>
</div>
```

PERO VEREMOS QUE AL REVISAR EL ÚLTIMO USUARIO INGRESADO NO TIENE NOMBRE

Revisemos la consola

```
Started POST "/users" for ::1 at 2016-01-29 19:13:10 -0300
Processing by Devise::RegistrationsController#create as HTML
Parameters: {"utf8"=>"✓",
"authenticity_token"=>"r1xSxcVBP8TpxehdYvp4ACW99kEQ4Fk5Y0DQ3fiDXLmIi/snqNKGHhUC52G/
UlbuqymgEk2Z+KboZgD+gaCYRg==", "user"=>{"name"=>"Gonzalo", "email"=>"gonzalo@desafiolatam.com",
"password"=>"[FILTERED]", "password_confirmation"=>"[FILTERED]"}, "commit"=>"Sign up"}
Unpermitted parameter: name
```

DEVISE NO TIENE CONTROLLERS, ¿DÓNDE PODEMOS AGREGAR LOS STRONG PARAMS?

2 Opciones, podemos generar los controllers de devise, o agregarlos al application controller.

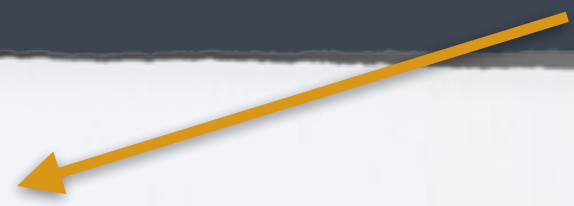
Ambos procesos se encuentran documentados en la documentación de la gema en **Github**.

CREANDO LOS CONTROLLERS DE DEVISE

```
rails generate devise:controllers [scope]
```

El scope es un supergrupo para todos las diversos groups de devise, como confirmation, registration, session, etc.

```
rails generate devise:controllers users
```



```
create app/controllers/users/confirmations_controller.rb  
create app/controllers/users/passwords_controller.rb  
create app/controllers/users/registrations_controller.rb  
create app/controllers/users/sessions_controller.rb  
create app/controllers/users/unlocks_controller.rb  
create app/controllers/users/omniauth_callbacks_controller.rb
```

Después de generar un controller para devise debemos especificar a las rutas que debe utilizarlo

```
devise_for :users, controllers: { registrations: "users/registrations" }
```

En el controller users/registrations

```
before_filter :configure_sign_up_params, only: [:create]  
protected  
  def configure_sign_up_params  
    devise_parameter_sanitizer.permit(:sign_up, keys: [:name])  
  end
```

CREEMOS UN PANEL DE CONTROL PARA LOS USUARIOS, LO PRIMERO ES PODER VERLOS

Para crear el index agregaremos

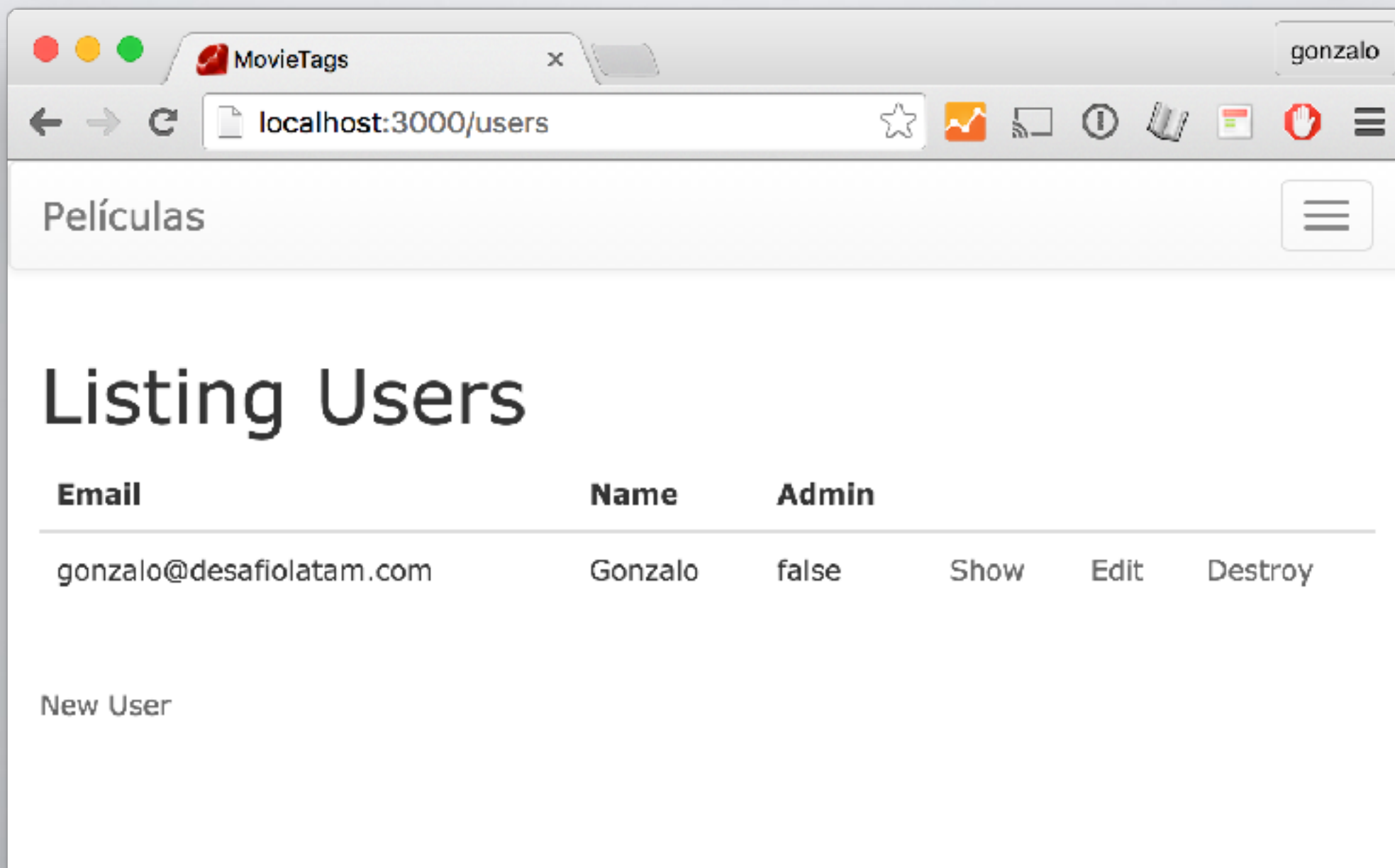
```
rails g scaffold_controller User name:string email:string admin:boolean
```

Revisamos que dentro del archivo de rutas se
encuentre

```
resources :users
```

YA TENEMOS PANEL DE CONTROL

SOLO NOS FALTA LIMITAR EL ACCESO

A screenshot of a web browser window. The browser has a single tab titled "MovieTags" with a red icon. The address bar shows "localhost:3000/users". The page content includes a header with "Películas" and a hamburger menu icon. Below this is a section titled "Listing Users". It contains a table with columns "Email", "Name", "Admin", and three action buttons: "Show", "Edit", and "Destroy". The table has one row with the data: "gonzalo@desafiolatam.com", "Gonzalo", "false". Below the table is a link labeled "New User".

Películas

Listing Users

Email	Name	Admin			
gonzalo@desafiolatam.com	Gonzalo	false	Show	Edit	Destroy

New User

YA HABÍAMOS CREADO EL MÉTODO



```
def filter_admin!  
  authenticate_user!  
  redirect_to root_path, alert: "No tienes acceso" unless current_user.admin?  
end
```

Moveremos este código al Application Controller.

Como todos los controllers heredan del Application Controller, por ende, podremos usar este método en el controller que queramos.

AHORA SIMPLEMENTE UTILIZAMOS EL MÉTODO DENTRO DEL CALLBACK BEFORE_ACTION

En el controller de users

```
before_action :filter_admin!
```