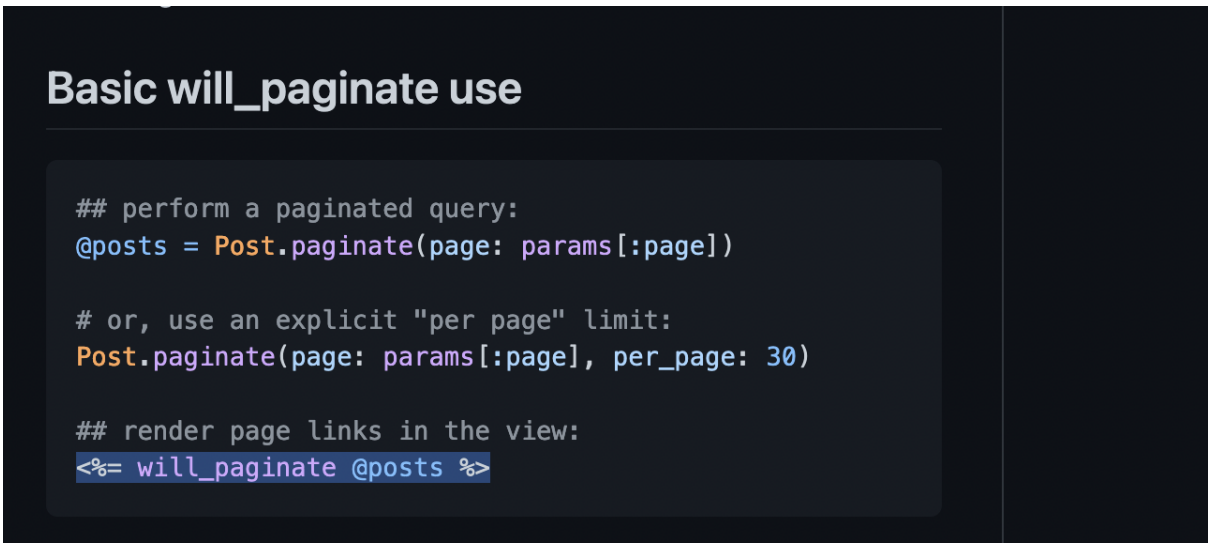
Pagination

will_paginate gem will help you with pagination

gem 'will_paginate', '~> 3.3'



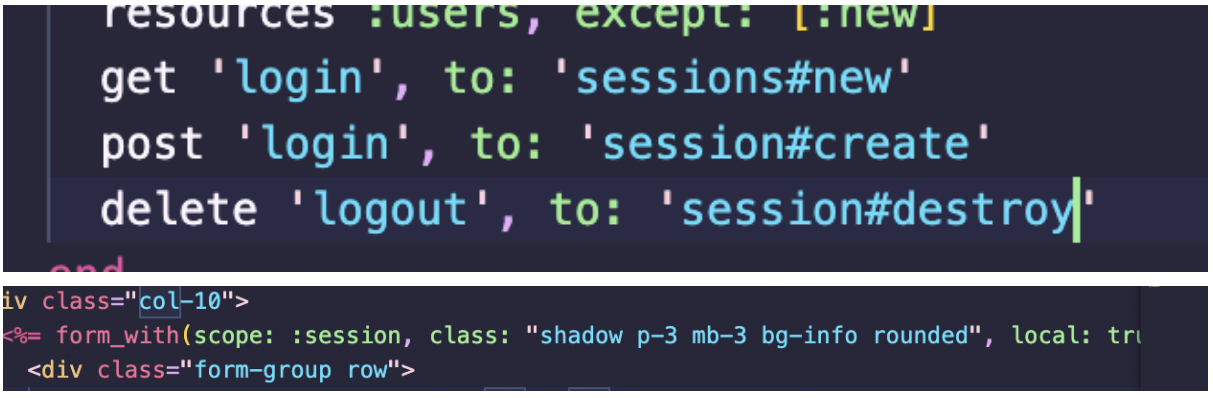
Estilos

http://mislav.github.io/will_paginate/

152. Add login form

usaremos sesiones para simular que un usuario esta logueado..

Crearemos un controlador nuevo para manejar esto.



154. Create and destroy user sessions

Mensajes flash

Flas sin now cuando haremos un redirect

```
def update
  if @article.update(article_params)
    flash[:notice] = "Article was updated successfully."
    redirect_to @article
  else
```

Flash now cuando no redirigimos

```
  render 'new'
  flash.now[:alert] = "There was something wrong with your login details"
```

Cuando no usamos now, flash persiste en la siguiente petición http más.
Si no lo hubiéramos usado el new.

Sessions

Las sesiones permiten tener un estado del usuario. En este caso seria si el usuario fue autentificado correctamente.

```
if user && user.authenticate(params[:ses
  session[:user_id] = user.id
```

Esto hara que se recuerde el id cada vez que el usuario hace una petición.

Lo que hacen las sesiones es almacenar un id para cada sesión.

5 Session

Your application has a session for each user in which you can store small amounts of data that will be persisted between requests. The session is only available in the controller and the view and can use one of a number of different storage mechanisms:

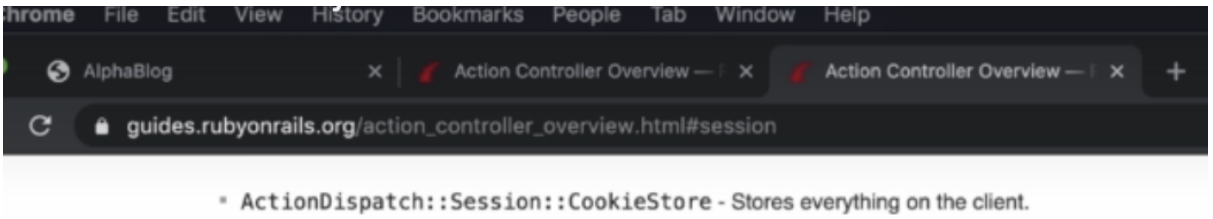
- `ActionDispatch::Session::CookieStore` - Stores everything on the client.
- `ActionDispatch::Session::CacheStore` - Stores the data in the Rails cache.
- `ActionDispatch::Session::ActiveRecordStore` - Stores the data in a database using Active Record. (require `activerecord-session_store` gem).
- `ActionDispatch::Session::MemCacheStore` - Stores the data in a memcached cluster (this is a legacy implementation; consider using CacheStore instead).

All session stores use a cookie to store a unique ID for each session (you must use a cookie, Rails will not allow you to pass the session ID in the URL as this is less secure).

Nosotros usaremos la primera opción que almacena toda la info en la misma cookie.

Esta sesión está encriptada en el lado del cliente y firmada para que nadie pueda modificarlo

Mas info en:



Para quitar la sesion simplemente

```
def destroy
  session[:user_id] = nil
```

More info
https://www.justinweiss.com/articles/how-rails-sessions-work/

156. Authentication helper methods

Memorización

Vamos a usar un método para obtener el user con el id de la session.

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

El problema es que esto es una llamada a la bd a cada rato. Asi que podemos usar algo llamado **memorización**
Eso es que si ya hemos hecho una referencia a ese usuario en el pasado, pues queremos regresar ese usuario en lugar de hacer otra llamada.

gira alrededor de @current_user, no se explica mas a fondo en el video

Yes, `current_user` uses `session`. You can do something similar in your application controller if you want to roll your own authentication:

```
def current_user
  return unless session[:user_id]
  @current_user ||= User.find(session[:user_id])
end
```

```ruby
def current_user
    user_id = session[:user_id]
    if user_id
        @current_user ||= User.find(user_id)
    end
end
```

Checar si esta logueado

```ruby
def logged_in?
    return !!@current_user
end
```

Checamos esa variable @current_user
El !! convierte lo de la derecha a booleano

Double bang operator

In Ruby (and many other languages) there are many values that evaluate to `true` in a boolean context, and a handful that will evaluate to false. In Ruby, the only two things that evaluate to `false` are `false` (itself) and `nil`.

If you negate something, that forces a boolean context. Of course, it also negates it. If you double-negate it, it forces the boolean context, but returns the proper boolean value.

For example:

```ruby
"hello"   #-> this is a string; it is not in a boolean context
!"hello"  #-> this is a string that is forced into a boolean
          #   context (true), and then negated (false)
!!"hello" #-> this is a string that is forced into a boolean
          #   context (true), and then negated (false), and then
          #   negated again (true)
!!nil     #-> this is a false-y value that is forced into a boolean
          #   context (false), and then negated (true), and then
          #   negated again (false)
```

158. Controller methods as helper methods

Learn how o use the methods of the controllers

NO podemos usar metodos del helper en el controlador

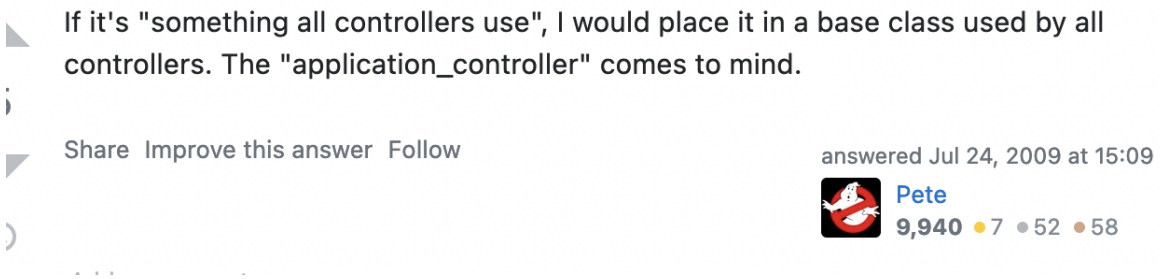
Podemos mover el método de get_user hacia la clase Application controller para que todos los controladores lo tengan. **El problema es que no funcionara en nuestras vistas.**

Para solucionarlo hay que decirle que también sea un método helper.

AHORA PODEMOS OBTENER EL USER CON EL MISMO CÓDIGO EN EL CONTROLADOR Y VISTA

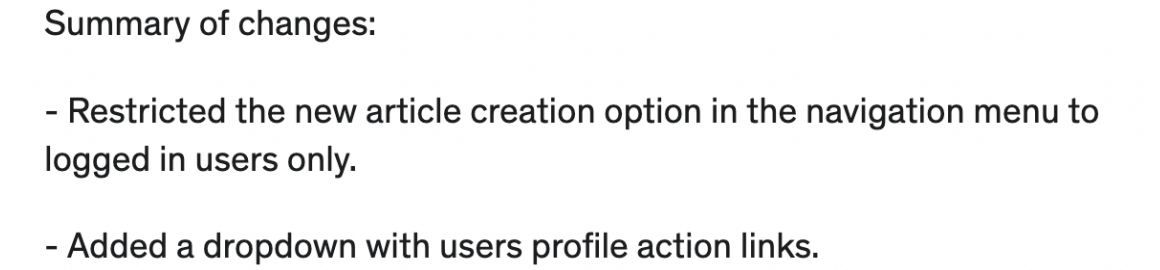
Ahora queremos que como facebook y otras apps, que si el usuario está logueado, se redireccione a otra página al tratar de ir a la home page.

HEMOS APRENDIDO A USAR MÉTODOS QUE FUNCIONEN COMO HELPER METHODS EN NUESTRAS VISTAS Y EN NUESTROS CONTROLADORES



160. Restrict actions from UI

Pusimos restricciones en cuanto a botones, pero aun así el usuario puede saltarse estas restricciones e ir a la página a través de la url.



164. Restrict actions at controller level - articles

Bloquear acceso a una página

Pones un método para verificar si el user esta logueado en la clase en que todos los controles heredan y poner el sig before_Action

```
> controllers > articles_controller.rb
  class ArticlesController < ApplicationController

      before_action :set_article, only: [:show, :edit, :update, :destroy]
      before_action :require_user, exept: [:show, :index]
```

```
def require_user
    if !logged_in?
        flash[:alert] = "You must be logged in to perform that action"
        redirect_to login_path
    end
end
```

Tambien en otras

```
1   class ArticlesController < ApplicationController
2
3       before_action :set_article, only: [:show, :edit, :update, :destroy]
4       before_action :require_user, exept: [:show, :index]
5       before_action :require_same_user, only: [:edit, :update, :destroy]
```

El require_same_user definirlo despues que require user

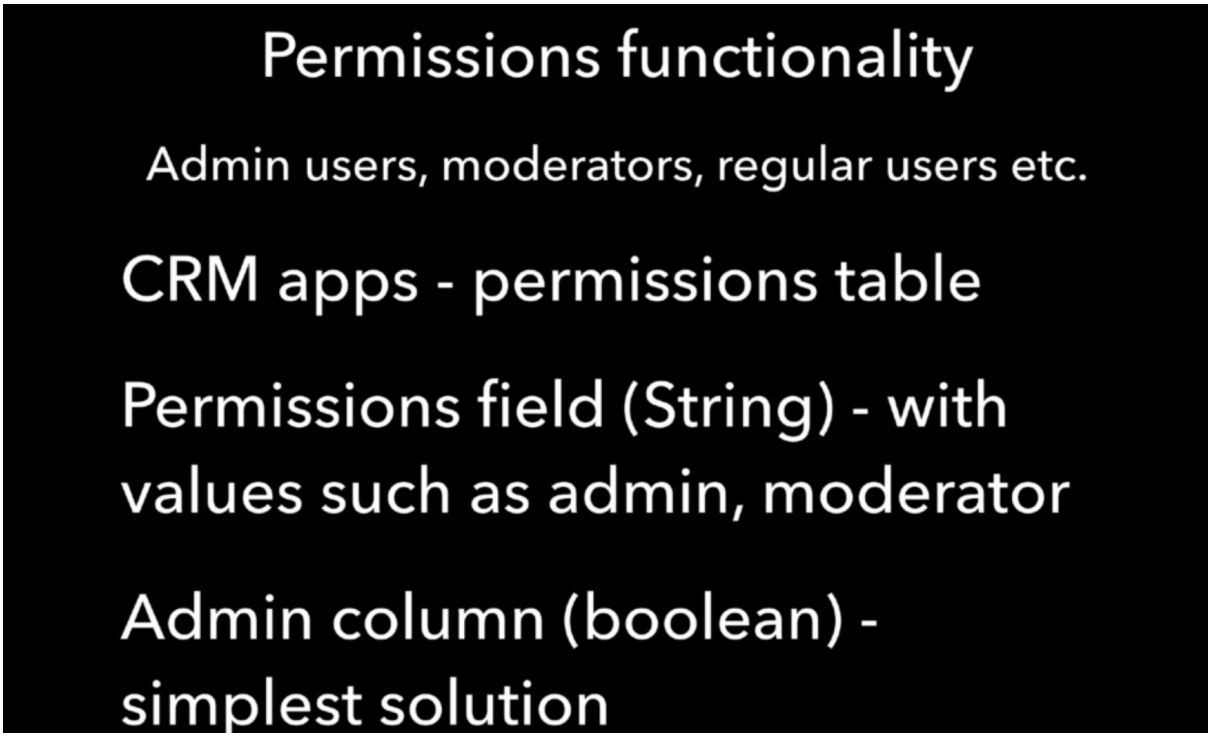
168. Delete user

Eliminar artículos al eliminar usuario

```ruby
class User < ApplicationRecord
  before_save { self.email = email.downcase }
  has_many :articles, dependent: :destroy
  validates :username, presence: true,
                       uniqueness: { case_sensitive: false },
                       length: { minimum: 3, maximum: 25 }
  VALID_EMAIL_REGEX = /\A[\w+\-.]+@[a-z\d\-.]+\.[a-z]+\z/i
  validates :email, presence: true,
                       uniqueness: { case_sensitive: false },
                       length: { maximum: 105 },
                       format: { with: VALID_EMAIL_REGEX }
  has_secure_password
end
```

Agregar en el modelo el dependent: destroy

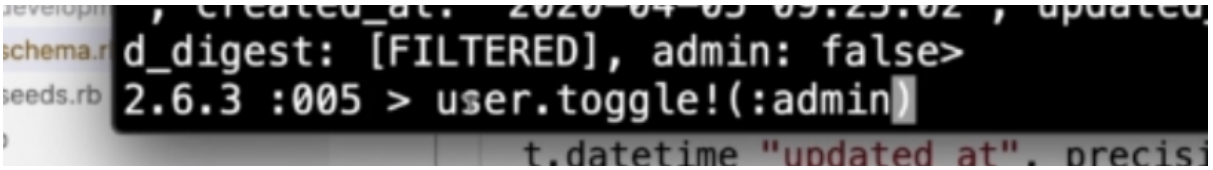
170. Add admin user functionality - intro



Varias maneras de manejar los permisos

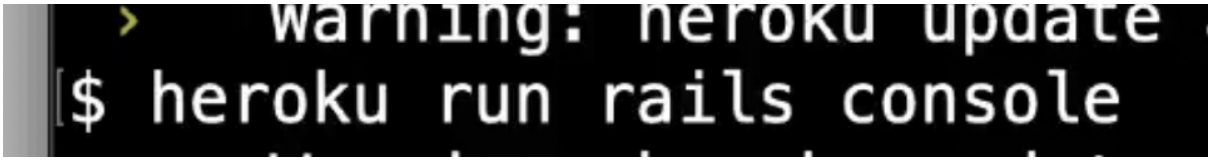
```
db > migrate > 20220816003657_add_admin_field_on_users.rb
1   class AddAdminFieldOnUsers < ActiveRecord::Migration[6.1]
2     def change
3       add_column :users, :admin, :boolean, default: false
4     end
5   end
6
```

Cambiar usuario hacia admin

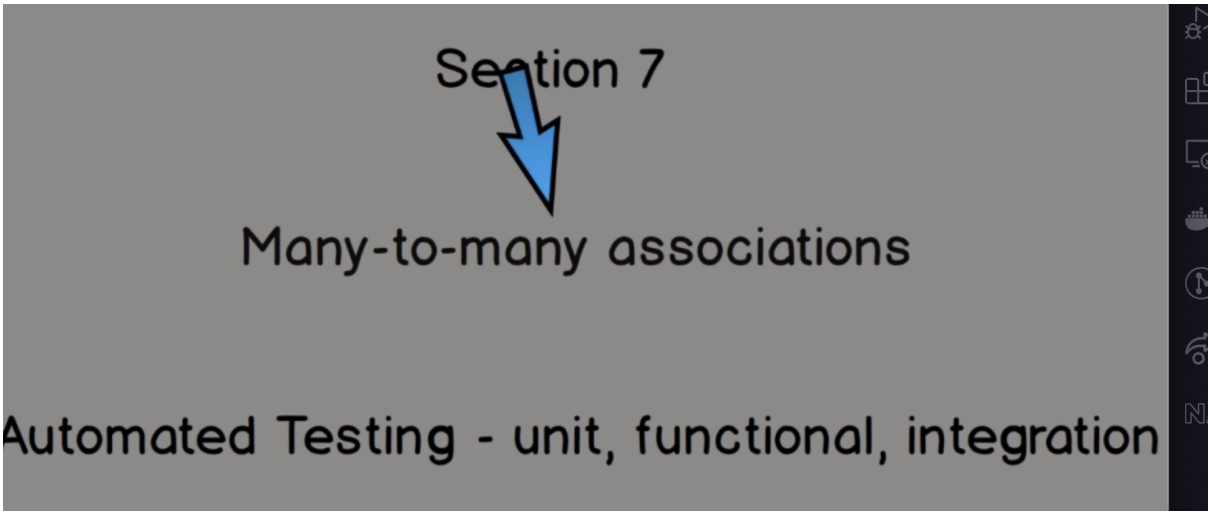



```
b > migrate > 20220816003657_add_admin_field_on_users.rb
1   class AddAdminFieldOnUsers < ActiveRecord::Migration[6.1]
2     def change
3       add_column :users, :admin, :boolean, default: false
4     end
5   end
6
```

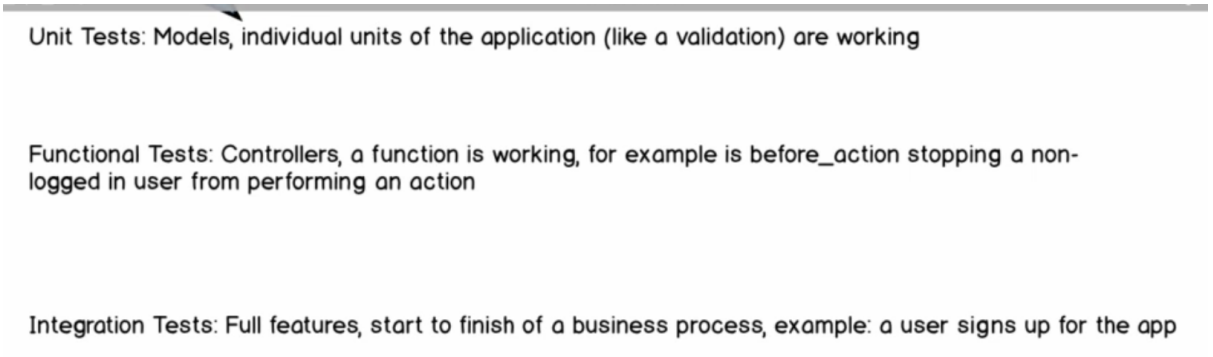
176. Production deploy and wrap up section 6



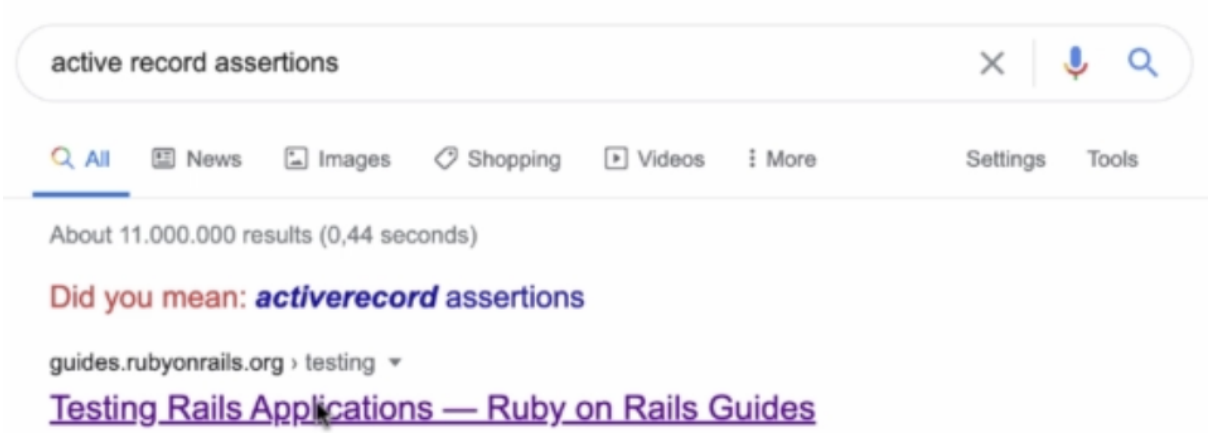
177. Introduction to Section 7



Unit, functional and integration test



178. Category model and testing



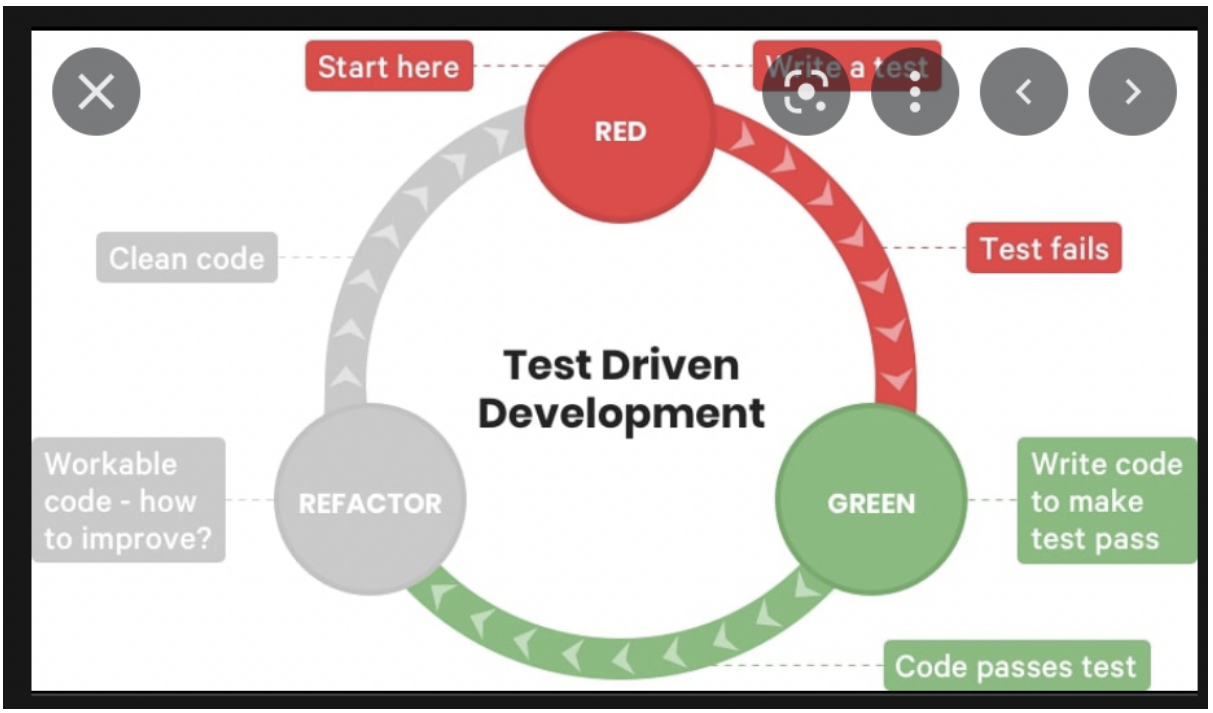
The default rails library for testing is mini-test, tienen assertions

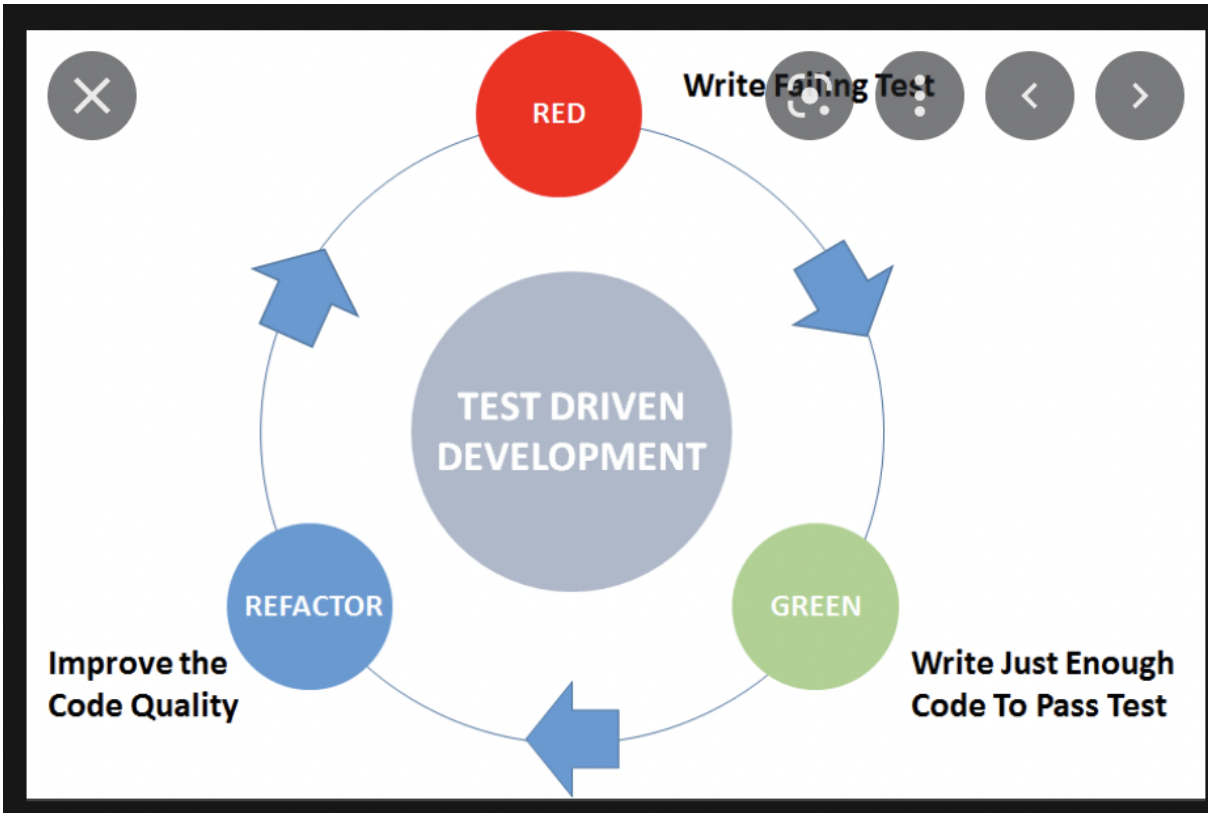
HAREMOS TESTING DE MODELOS
Como queremos hacer un test de modelo de la categoria creamos
category_test.rb

Para correrlo usamos rails test (corre todos los tests)

Test driven development

Escribiremos la cantidad de código suficiente para hacer primero el test.


Start here
RED
Write a test
Test fails
Clean code
Test Driven Development
Workable code - how to improve?
REFACTOR
GREEN
Write code to make test pass
Code passes test


RED
Write Failing Test
TEST DRIVEN DEVELOPMENT
REFACTOR
GREEN
Improve the Code Quality
Write Just Enough Code To Pass Test

Different types of testing 0

Sarah · Clase 177 · hace 4 años

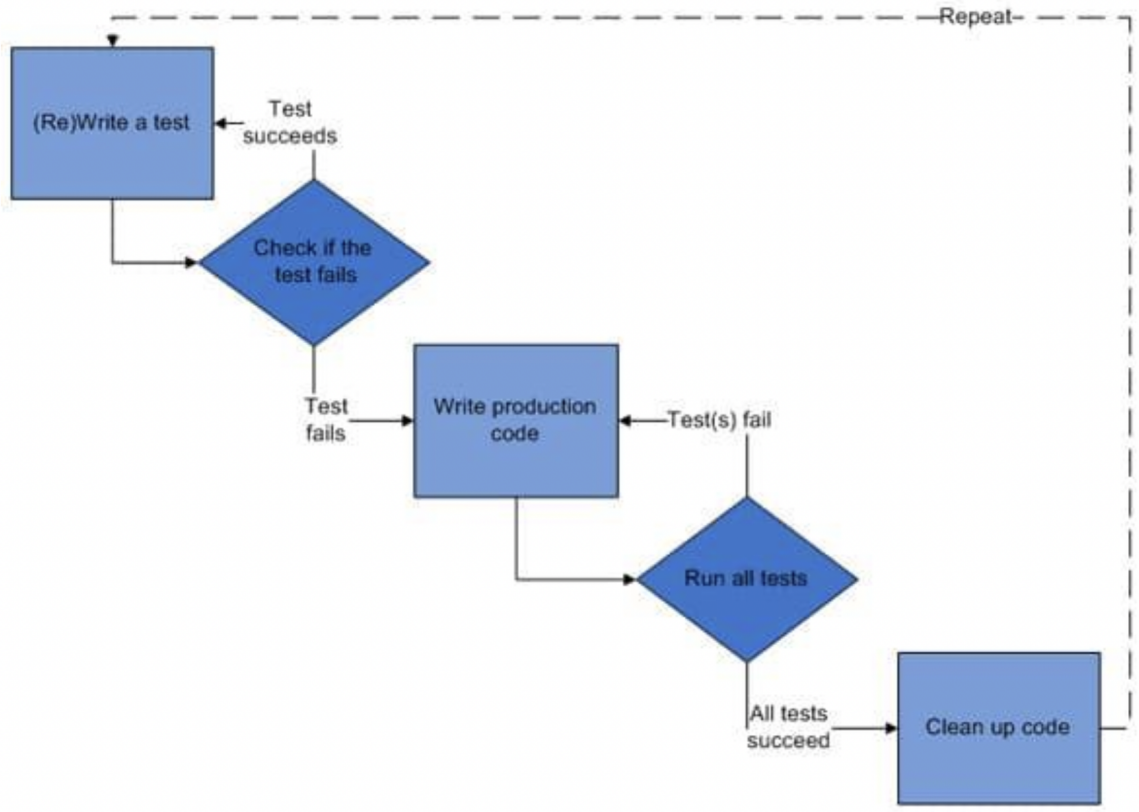
My company does TDD and other testing so I've been spending a lot of time studying testing and working with senior engineers on my team on it, and I think this definition of functional tests is confusing. The Rails documentation says this about functional tests: "When writing functional tests, you are testing how your actions handle the requests and the expected result or response, in some cases an HTML view."

1 respuesta **Seguir las respuestas**

SR Sasikala — Profesor asistente 2

hace 4 años

Hi Sarah, What Mashrur says is pretty much same as what is in Rails documentation. It means when you say `"you are testing how your actions handle the requests"` , it is nothing but controller actions of the app which handles the requests and `"the expected result or response, in some cases an HTML view"` means that the controller renders the view as response in case of index or show actions. Also, Mashrur was mentioning that in case of using before_action callback, we can even check whether controller action has been using the callback or not. Hope this helps you to make the connection between the two definitions.



Queremos probar que la categoría sea válida

1. Escribimos el test
2. El test falla porque no existe la categoría
3. Escribimos la cantidad mínima para hacer que esa prueba funcione (creamos el modelo categoría)
4. Probamos otra vez
    a. Error, nos falta la tabla categoria, asi que la agregamos
    b. Esto se repite hasta el éxito
5. Probamos otra vez y da éxito

Los tests corren en una base de datos de testeo. Cada vez que corres un test la bd es limpiada.

Esto es un test automatizado

segun esto en algunas versiones daba error el rails test porque una gema no se agregaba por default

Por si acaso nosotros la pondremos

```
gem 'rexml', require: false
```

180. Validations using unit tests

En cada test, los objetos y variables se van a limpiar, asi que @category no existira en el proximo test

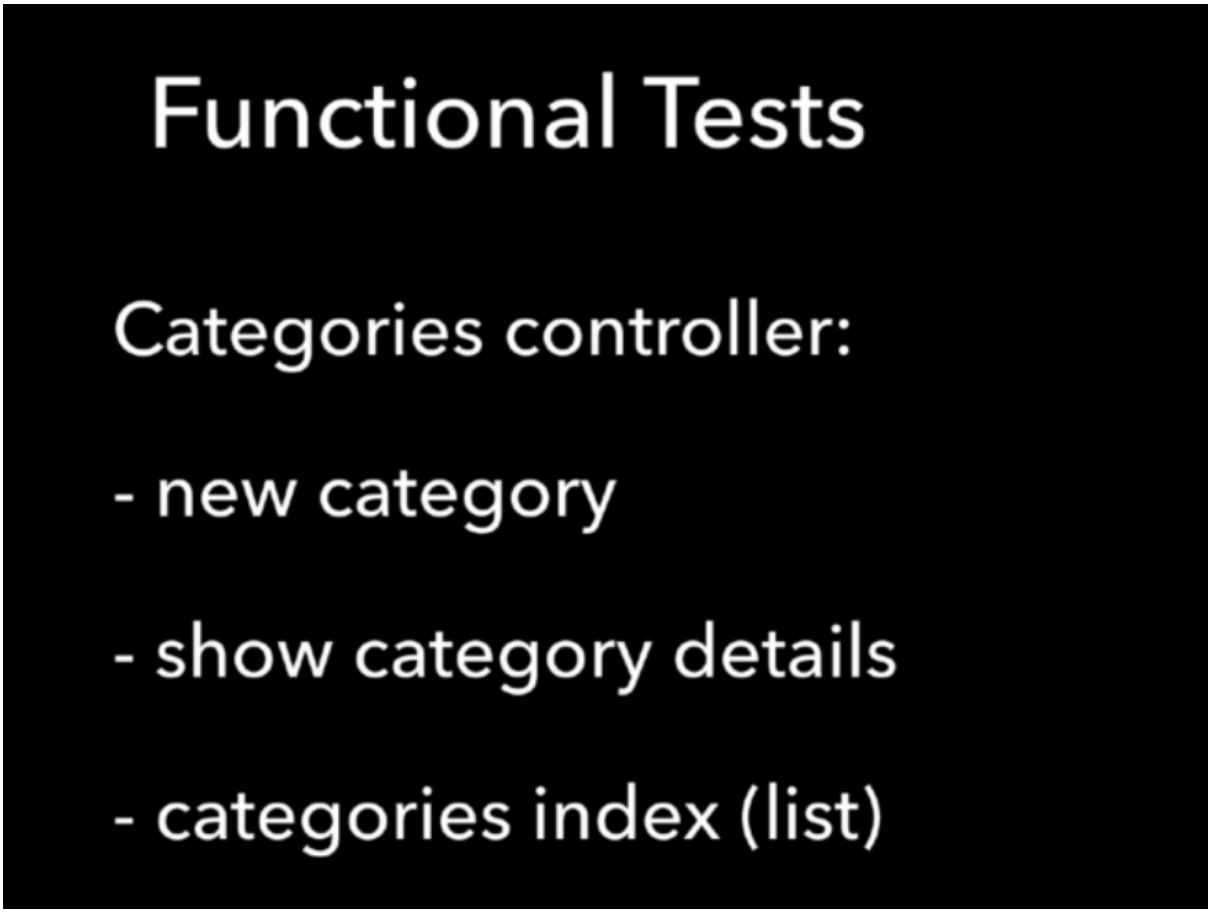
```
test "category should be valid" do
  @category = Category.new(name: "Sports")
  assert @category.valid?
end

test "name should be present" do
  @category = Category.new(name: " ")
  assert_not @category.valid?
end
```

Si no queremos crear el category en cada test podemos usar setup. **Este corre algo al comienzo de cada test**

```
def setup
    @category = Category.new("Fantasy")
end
```

182. Categories controller and tests

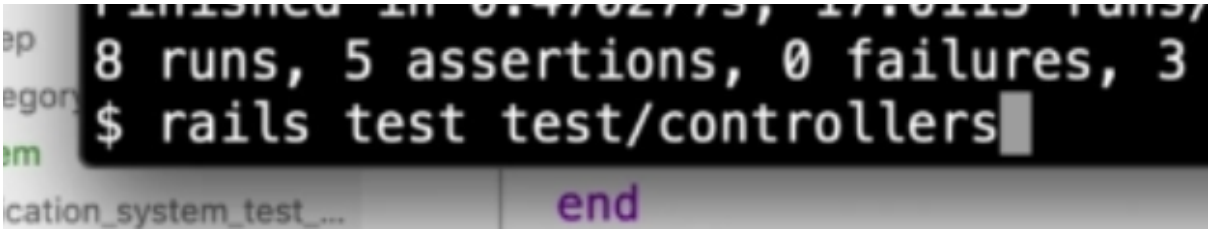


show category details -> show all movies on that category

Generador

```
~/rails_projects_2020/rails_6_projects/alpha-blog — -bash
$ rails generate test_unit:scaffold category
```

Solo testear solo controladores



o solo un controlador
rails test test/controllers/categories_controller_test.rb

Esos test de módulos y controlador están testeando cosas individuales (Test funcionales)
Queremos hacer un test de integración para ver que todo funcione en conjunto

186. Integration test: Create category business process

Test funcional vs integración

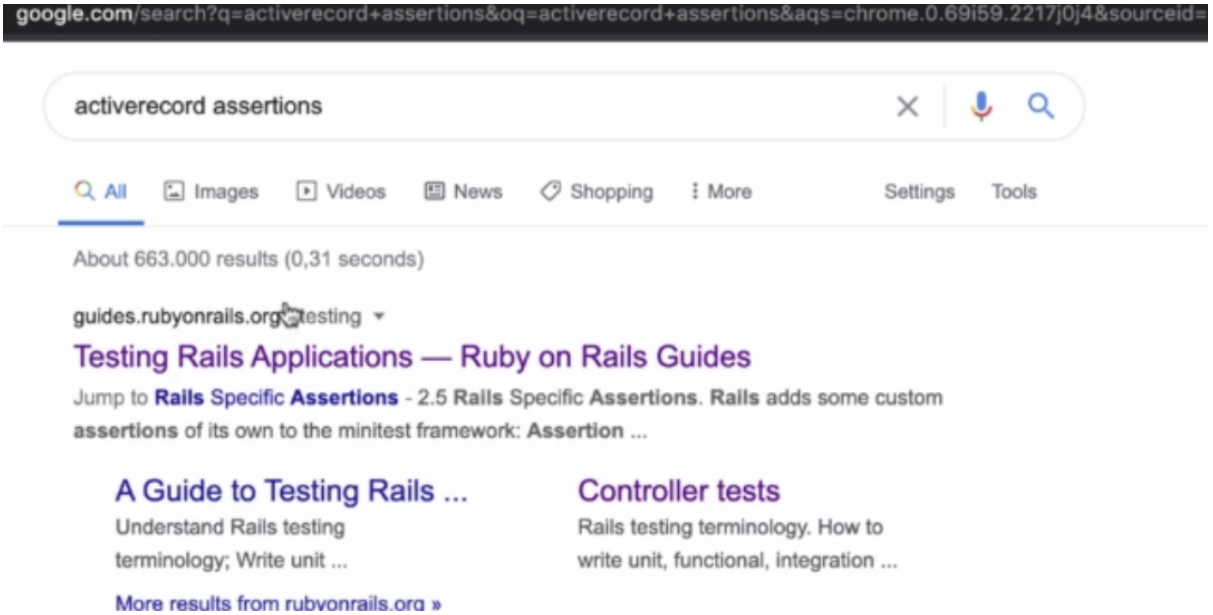
Funcional o test de controlador

Es para checar que las funciones individuales del controlador funcionen

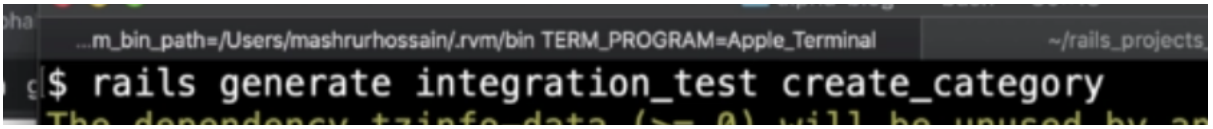
Integración

Testea el full business process y puede incluir múltiples funciones. Checa que estas funcionen juntas

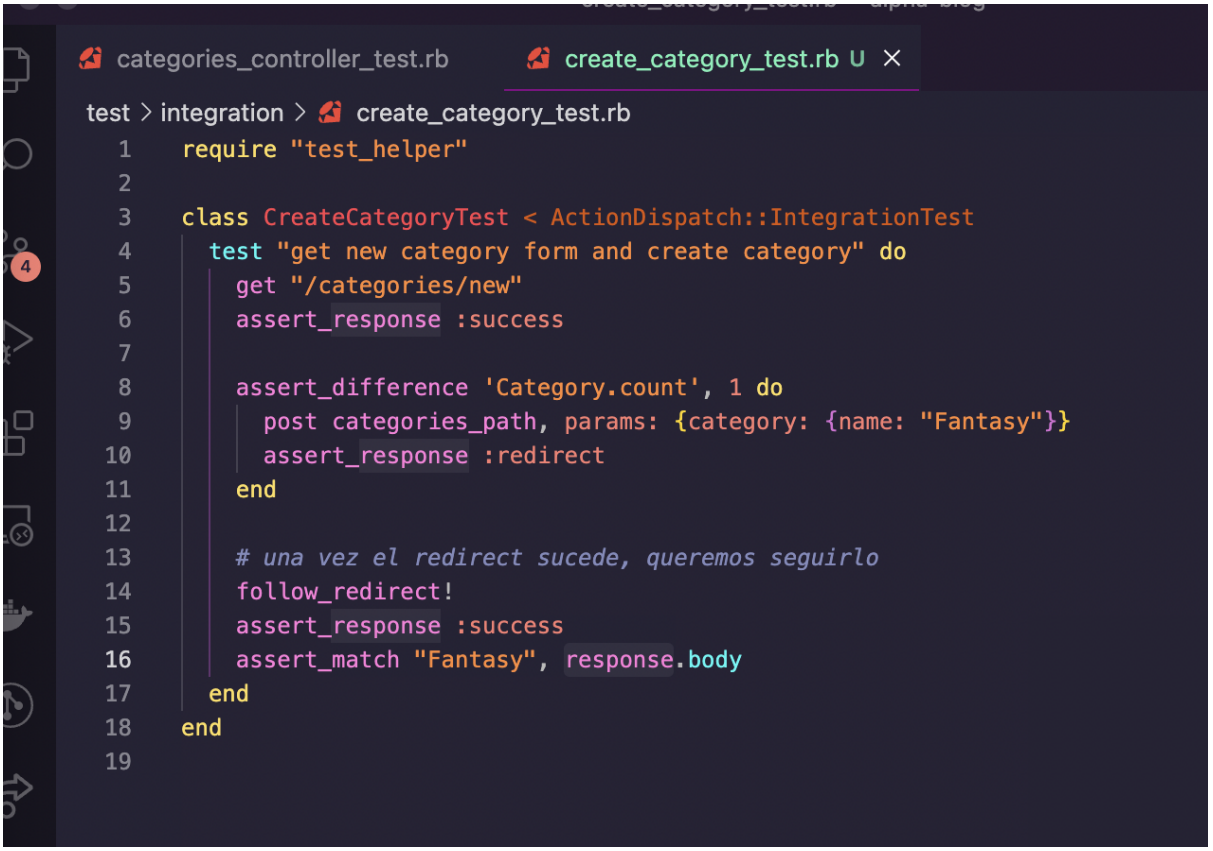
Pagina para saber más de testing



Generar integration test



Haremos un test de integración de todo el proceso de crear una categoría



Eso fue integración pues fue todo un flow, desde entrar a index, ir hacia la página de new, hacer el post, ser redirigido hasta ver la página de la categoría.

188. Integration test for invalid category

190. Integration test and feature: listing categories

The integration test now will be create two categories and check if these two categories are shown on the index page

192. Admin user requirement and test

Podemos poner el método de login en el archivo test_helper

197. Many-to-many association - back-end implementation

models > category.rb

```
class Category < ApplicationRecord
    validates :name, presence: true, length: {minimum: 3, ma
    validates_uniqueness_of :name
    has_many :article_categories
    has_many :articles, through: :article_categories
end
```

```
class Article < ApplicationRecord
    belongs_to :user
    has_many :article_categories
    has_many :categories, through: :article_categories
    validates :title, presence: true, length: {minimum: 6, m
    validates :description, presence: true, length: {minimum
end
```

models > article_category.rb

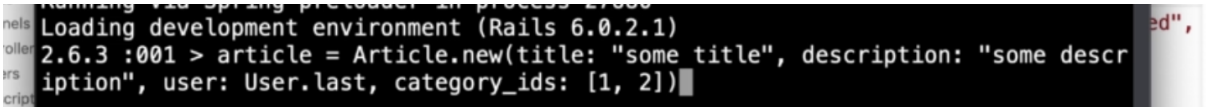
```
class ArticleCategory < ApplicationRecord
    belongs_to :article
    belongs_to :category
end
```

Agregar a la tabla article_category

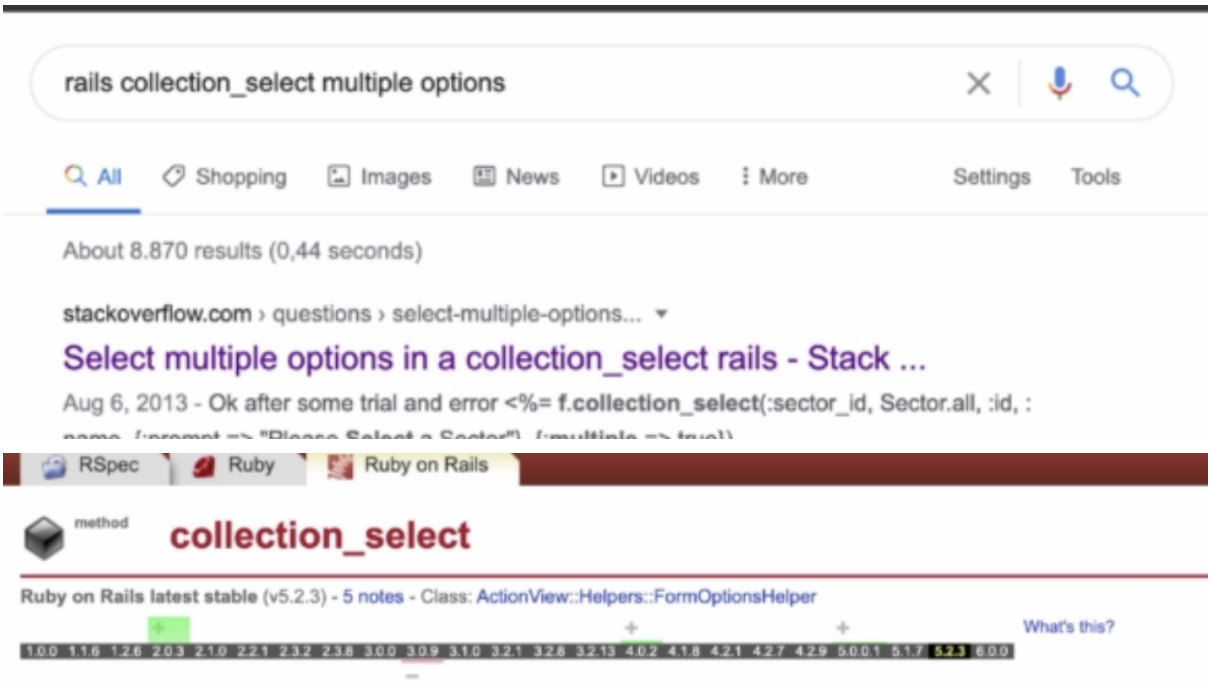
Se puede desde article.categories << article o
category.articles << category

199. Add association from UI

Create article with category

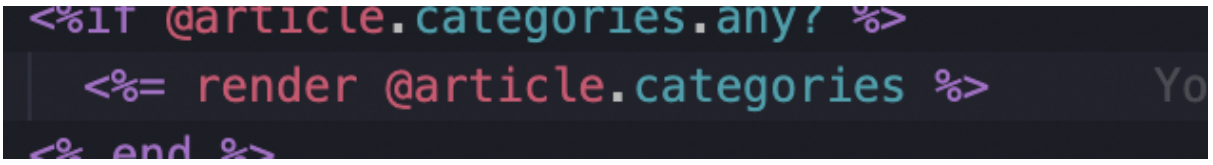


Since they are multiple ids you add an s to “category_ids”



201. Update article views to display categories

Render partial of each category inside article



203. Complete category index and show views

207. Deploy to production, homework, wrap up section 7