Laboratorio de Computación IV

Clase 20

Andrés Fortier

Repaso

- Repasamos algunas tareas de rake.
- Modificar migrations.
- Reemplazar un *join table* por un *join model*
  - *Join table* es “invisible” como modelo
    - No tiene id.
  - *Join model* es un modelo AR “normal”.

Repaso - Relaciones M:N

articles

id	title
1	Post number 0
2	Post number 1

article_categorization

article_id	category_id	position
1	1	1
1	2	2
2	1	1
2	3	2

categories

id	name
1	OOP
2	Programming
3	Smalltalk

Repaso

- Importancia de no perder los datos ante un cambio de esquema
  - Crear el *join model* y ejecutar la migration.
  - Configurar las relaciones.
  - Migrar los datos.
  - Crear un migration para eliminar la tabla actual.

Relaciones M:N

- Dejamos en validar el par artículo - categoría

/app/models/article_categorization.rb

```
class ArticleCategorization < ActiveRecord::Base
 belongs_to :article
 belongs_to :category

 validates_presence_of :article, :category, :position

 validates_uniqueness_of :position, :scope => :article_id
 validates_uniqueness_of :category_id, :scope =>
:article_id
end
```

Relaciones M:N

- `db:reset` nuevamente.
- En una consola

```
$ bin/rails console
> ArticleCategorization.create!(article_id: 17, category_id:
3, position: 2)
> ArticleCategorization.create!(article_id: 17, category_id:
3, position: 3)
...
```

Relaciones M:N

- Finalmente nos gustaría que la relación \`categories\` esté ordenada por posición

```
/app/models/article.rb
```

```ruby
class Article < ActiveRecord::Base

  ...
  has_many :categories, -> { order 'position ASC' },
:through => :article_categorizations

  ...
end
```

- Prueben modificar \`ASC\` por \`DESC\`.

Relaciones M:N

- Cosas pendientes
  - `add_category` sólo funciona para posiciones consecutivas.
    - No lo validamos en `ArticleCategorization`.
  - Deberíamos agregar al menos un `remove_category` que reorganize los índices.
  - Manejo del orden de las categorías.
  - Nota sobre encapsulamiento en general.
- Ver gema “acts_as_list”.

Relaciones M:N

- Dejen el branch, pero no mergeen en master.

Inicio

- Vuelvan a master.
- `db:reset`.
- Extiendan la clase User

/app/models/user.rb

```ruby
  ...
  def titles_in_category(category_name)
    matching_articles = articles.select do |article|
      article.categories.any? do |category|
        category.name == category_name
      end
    end
    matching_articles.map(&:title)
  end
  ...
```

rspec y Rails

- rspec – framework de test para ruby
- rspec-rails – extiende rspec para manejar tipos de tests particulares
  - Modelo (generalmente ActiveRecord).
  - Controllers.
  - Vistas.
  - etc.
- http://www.relishapp.com/rspec/rspec-rails/v/3-2/docs

rspec y Rails

/Gemfile

```
...
group :development, :test do
  gem 'rspec-rails', '~> 3.0'
end
...
```

```
$ bundle install
```

```
$ rails generate rspec:install
```

```
$ bundle exec rspec
No examples found.

Finished in 0.00021 seconds (files took 0.10992 seconds to
load)
0 examples, 0 failures
```

rspec - policies

- Comencemos con un test para `ArticlePolicy`
  - pundit provee algunos helpers para usar en rspec

```
/spec/spec_helper.rb
```

```ruby
...

require "pundit/rspec"

RSpec.configure do |config|
  # rspec-expectations config goes here. You can use an
  # alternate assertion/expectation library such as wrong
  # or the stdlib/minitest assertions if you prefer.
  config.expect_with :rspec do |expectations|
  ...
end
```

rspec - policies

- Creen la carpeta `/spec/policies`

```
/spec/policies/article_policy_spec.rb
```

```ruby
require "rails_helper"

RSpec.describe ArticlePolicy, :type => :model do
  subject { ArticlePolicy }

  permissions :new? do
    it "is denied to non-logged users" do
      expect(subject).not_to permit(nil, Article)
    end

    it "is allowed to any logged in user" do
      expect(subject).to permit(User.new, Article)
    end
  end
end
```

rspec - policies

- Creemos algunos fixtures y extendamos el test

```
/spec/policies/article_policy_spec.rb
```

```
require "rails_helper"

RSpec.describe ArticlePolicy, :type => :model do
  subject { ArticlePolicy }

  let(:user)   {User.new(email: "user@example.com",
password: "12345678", password_confirmation: "12345678")}
  let(:author) {User.new(email: "author@example.com",
password: "12345678", password_confirmation: "12345678")}
  let(:admin) do
    user = User.new(email: "admin@example.com", password:
"12345678", password_confirmation: "12345678")
    user.add_role :admin
    user
  end
```

rspec - policies

```
/spec/policies/article_policy_spec.rb
```

```ruby
  let(:article) {Article.new(title: "The title", text: 
"The body", author: author)}

  permissions :new? do
    it "is denied to non-logged users" do
      expect(subject).not_to permit(nil, Article)
    end

    it "is allowed to any logged in user" do
      expect(subject).to permit(user, Article)
    end
  end
```

rspec - policies

/spec/policies/article_policy_spec.rb

```
  permissions :destroy? do
    it "is denied to non-logged users" do
      expect(subject).not_to permit(nil, article)
    end

    it "is denied if the user is not the author" do
      expect(subject).not_to permit(user, article)
    end

    it "is allowed if the user is the article author" do
      expect(subject).to permit(author, article)
    end

    it "is allowed if the user is an admin" do
      expect(subject).to permit(admin, article)
    end
  end
```

rspec - models

- Ahora debemos testear un modelo AR
- ¿Qué se testea?
  - ¿Relaciones y comportamiento de AR?
  - ¿Validaciones?
  - ¿Mensajes que agregan las librerías?
  - ¿Mensajes que agregamos nosotros?
- Importante: persistir los modelos para que las relaciones funcionen.

rspec - user

/spec/model/user_spec.rb

```
require "rails_helper"

RSpec.describe User, :type => :model do

  let(:user)    {User.create!(email: "user@example.com",
password: "12345678", password_confirmation: "12345678")}
  let(:author)  {User.create!(email: "author@example.com",
password: "12345678", password_confirmation: "12345678")}
  let(:article) {Article.create!(title: "The title", text:
"The body", author: author)}
  let(:test_category_name) {"Test Category"}

  describe "::titles_in_category" do
```

rspec - user

/spec/model/user_spec.rb

```
...
it "return an empty array if the user has no associated
articles" do

  expect(user.titles_in_category(test_category_name))
    .to be_empty

end
...
```

rspec - user

/spec/model/user_spec.rb

```
...
it "returns an empty array if the user has an article with
no categories" do

  expect(author.titles_in_category(test_category_name))
    .to be_empty

end
...
```

rspec - user

```
/spec/model/user_spec.rb
```

```
...
it "returns an empty array if the user has a categorized
article but the categories do not match" do

  new_category = Category.create!(name: "New category")
  article.categories << new_category
  expect(author.titles_in_category(test_category_name))
    .to be_empty

end
...
```

rspec - user

```
/spec/model/user_spec.rb
```

```
...
it "returns the article title if the user has an article
with the requested category name" do

  new_category=Category.create!(name: test_category_name)
  article.categories << new_category
  expect(author.titles_in_category(test_category_name))
    .to eq([article.title])

end
...
```

rspec - article

- Validaciones

```
/spec/model/article_spec.rb
```

```
require "rails_helper"

RSpec.describe Article, :type => :model do

  let(:author) {User.new(email: "author@example.com",
password: "12345678", password_confirmation: "12345678")}

  describe "Validations" do
```

rspec - article

```
/spec/model/article_spec.rb
```

```ruby
...
it "is not valid if title is absent" do
  expect(Article.new(author: author)).not_to be_valid
end
...
```

rspec - article

```
/spec/model/article_spec.rb
```

```
...
it "is not valid if the title's length is less than 5
characters" do

  expect(Article.new(author: author, title: "Test"))
    .not_to be_valid

end
...
```

rspec - article

```
/spec/model/article_spec.rb
```

```
...
it "is valid if the title's length is 5 characters or
more" do

  expect(Article.new(author: author, title: "Tests"))
    .to be_valid
  expect(Article.new(author: author, title: "Test #2"))
    .to be_valid

end
...
```

rspec - controllers

- Debemos incluir los helpers de devise

```
/spec/spec_helper.rb
```

```
...
require "pundit/rspec"
require "devise"

RSpec.configure do |config|
...

 config.include Devise::TestHelpers, :type => :controller

...
end
```

rspec - article

```
/spec/controllers/articles_controller_spec.rb
```

```ruby
require "rails_helper"

RSpec.describe ArticlesController, :type => :controller do

  let(:user) {
    User.create!(email: "author@example.com", password: 
"12345678", password_confirmation: "12345678")
  }

  before(:each) do
    sign_in user
  end
```

rspec - article

/spec/controllers/articles_controller_spec.rb

```
...
describe 'GET index' do

  it "returns 200 (ok) response code" do
    get :index
    expect(response).to have_http_status(:ok)
  end

  it "renders the index template" do
    get :index
    expect(response).to render_template("index")
  end
  ...
```

rspec - article

```
/spec/controllers/articles_controller_spec.rb
```

```
...
it "leaves an empty relationship on @articles if there are
no articles" do

  get :index
  expect(assigns(:articles)).to be_empty

end
...
```

rspec - article

/spec/controllers/articles_controller_spec.rb

```
...
it "assigns the latest 10 posts to @articles" do

  Article.create!(title: "Post number 1", text: "My first
post!", author: user)
  last_articles = (2..11).map do |i|
    Article.create!(title: "Post number #{i}", text: "My
#{i} post!", author: user)
  end

  get :index

  expect(assigns(:articles))
   .to eq(last_articles.reverse)

end
```

rspec – tipos de test

- Model.
- Controller.
- View.
- Routes.
- Helpers.
- Requests.
- Features.

rspec – tipos de test

Tipo de test	Unidad	Integración
Model – No ActiveRecord	Si	No
Model - ActiveRecord	Depende	Depende
Controller	Depende	Depende
View	Depende	Depende
Routes	Depende	Depende
Helpers	Depende	Depende
Requests	No	Si
Features	No	Si

Tarea para el hogar

- Vayan a http://www.relishapp.com/rspec/rspec-rails/v/3-2/docs/
 y miren los distintos tipos de tests y sus ejemplos.

Tests y entrega final

- Al menos deberían tener
  - Una clase del modelo (no AR) testeada.
  - Una clase del modelo (AR) testeada.
  - Una clase de controlador testeada.
  - Un test de request que ejercite un POST a un form.

Pasos siguientes

- Vamos a ir pasando a una modalidad mas “taller”
- Lo que nos queda
  - Un poco de javascript.
  - Edición de modelos anidados.
  - Búsquedas.
- Tenían que traer un listado de cosas que quieren hacer pero no saben cómo.