Integrallis

Ruby on Rails

by Chris Irish

Part 3 of 5

http://www.integrallis.com

Building an App

Rails Development

Glob
Our Blog App

Why a blog?

Starting a New App

```
$ rails new glob -T
      create
      create  README.rdoc
      create  Rakefile
      create  config.ru
      create  .gitignore
      create  Gemfile
      ...
$ cd glob/
```

Setup RVM Gemset

Isolated Environment

```
$ echo "rvm use 1.9.3@glob --create" > .rvmrc
$ cd .
# RVM will find the .rvmrc file and prompt you to verify it
```

Quick Cleanup
Remove Cruft

```
$ rm public/index.html
$ rm app/assets/images/rails.png
```

```
# Pick the frameworks you want:
require "active_record/railtie"
require "action_controller/railtie"
require "action_mailer/railtie"
# require "active_resource/railtie"
require "sprockets/railtie"
```

Initial Commit

```
$ git init
$ git add .
$ git commit -m "Initial commit"
```

RSpec Setup

```
# Gemfile
group :test, :development do
  gem 'rspec-rails', '~> 2.12.2'
end

# dev will give us the spec generators
```

```
$ bundle # or bundle install
```

RSpec Install

```
$ rails g rspec:install
      create  .rspec
      create  spec
      create  spec/spec_helper.rb
```

spec_helper.rb
Test Configuration

We'll require `spec_helper.rb` in our test files

The default setup is fine for now

Running RSpec

```
$ rspec spec/
No examples found.

Finished in 0.00007 seconds
0 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Install RSpec"
```

Capybara
Speak Browser

Capybara is an "acceptance test framework for web applications"

Drive out application from the UI level

Runs specs in a browser if we need a JS runtime

Installing Capybara

Speak Browser

```
# Gemfile
group :test do
  ...
  gem 'capybara', '~> 2.0.1'
end
```

```
$ bundle install
```

```
# spec/spec_helper.rb
require 'capybara/rspec' # integrate with RSpec
require 'capybara/rails' # integrate with Rails
```

Make Sure it Works

```
$ rspec spec/
No examples found.

Finished in 0.00007 seconds
0 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Install Capybara"
```

Our First Test

Tests First, Questions Later

First test is UI-level to make sure we have a home page

We'll write a Rails integration test with RSpec

Our First Test

Tests First, Questions Later

```
# test the home page
$ rails g rspec:integration home_page
      create  spec/requests/home_pages_spec.rb
```

GOTCHA
Capybara and RSpec

```
$ mkdir spec/features
$ mv spec/requests/home_pages_spec.rb \
     spec/features/home_page_spec.rb
```

The Test

Do we have a home page?

```
describe "Home Page" do
  describe "GET '/'" do
    it "is successful" do
      visit root_path
      expect(page).to have_content("Glob")
    end
  end
end
```

Run The Test

Do we have a home page?

```
$ rspec spec
...
Failure/Error: visit root_path
NameError:
  undefined local variable or method `root_path' for
#<RSpec::Core::ExampleGroup::Nested_1::Nested_1:0x007fc1a36abf00>
...
```

Declaring Root Route

Respond to /

```
# config/routes.rb
Glob::Application.routes.draw do
  root :to => 'main#index'
end
```

Run The Test

Do we have a home page?

```
$ rspec spec
...
Failure/Error: visit root_path
ActionController::RoutingError:
  uninitialized constant MainController
...
```

Our First Controller

Code we're routing to

```
$ touch app/controllers/main_controller.rb
```

```
# app/controllers/main_controller.rb
class MainController < ApplicationController
end
```

Run The Test

Do we have a home page?

```
$ rspec spec
...
AbstractController::ActionNotFound:
  The action 'index' could not be found for MainController
...
```

Run The Test

Do we have a home page?

```
class MainController < ApplicationController
  def index
  end
end
```

Run The Test

Do we have a home page?

```
$ rspec spec
```

```
...
ActionView::MissingTemplate:
  Missing template main/index, application/index with
{:locale=>[:en], :formats=>[:html], :handlers=>[:erb, :builder, :coffee]}.
Searched in:
...
```

Our First Template
HTML Output

```
$ mkdir app/views/main
$ touch app/views/main/index.html.erb
```

Run The Test

Do we have a home page?

```
$ rspec spec
...
Failure/Error: expect(page).to
have_content("Glob")
   expected there to be text "Glob" in ""
...
```

Add Content

Templates neeeed content

```
<!-- we're in app/views/main/index.html.erb -->
<h1>Glob</h1>
```

Run The Test

Do we have a home page?

```
$ rspec spec
Finished in 0.19433 seconds
1 example, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Working home page"
```

Fire Up a Browser

To the victors belong the spoils

Twitter Bootstrap

Design for Devs

Faster, easier web development

We want out app to look good

Focus on Rails not design

Twitter Bootstrap

Design for Devs

```
# Gemfile
group :assets do
  #...
  gem 'bootstrap-sass'
end
```

```
$ bundle install
```

Asset Pipeline

Making JS/CSS first class citizens

app/assets/stylesheets/application.css

```
*= require bootstrap
*= require_self
*= require_tree .
*/
```

Twitter Bootstrap
Design for Devs

app/views/layouts/application.html.erb

```
<body>
  <div class="container">
    <div class="navbar">
      <div class="navbar-inner">
        <%= link_to 'Glob', root_path, :class => 'brand' %>
      </div>
    </div>
  </div>
  <div class="container">
    <%= yield %>
  </div>
</body>
```

Twitter Bootstrap

Design for Devs

app/assets/stylesheets/styles.scss.css

```
body  {
   margin-top: 20px;
}
```

Run The Tests

```
$ rspec spec
Finished in 0.19433 seconds
1 example, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Add bootstrap-sass and layout structure"
```

Our First Model

Tests first please!

```
$ mkdir spec/models
$ touch spec/models/article_spec.rb
```

Our First Model

Tests first please!

```
# spec/models/article_spec.rb
require 'spec_helper.rb'

describe Article do
end
```

Run The Tests

```
$ rspec spec
uninitialized constant Article (NameError)
...
```

Generating a Model

Every Blog Needs Posts

```
$ rails g model article title body:text
      invoke  active_record
      create    db/migrate/2012XXXXXXX_create_articles.rb
      create    app/models/article.rb
```

Write a Test

Can we persist to the DB?

```
describe Article do
  it "creates an article" do
    article = Article.new(:title => "Blah", :body => "lorem ipsum")
    article.save
    expect(article).to be_persisted
  end
end
```

Run the Tests

Can we persist to the DB?

```
$ rspec spec
...
ActiveRecord::StatementInvalid:
  Could not find table 'articles'
...
```

Run the Migration

Create the DB table

```
$ rake db:migrate
==  CreateArticles: migrating
=============================================================
-- create_table(:articles)
   -> 0.0012s
==  CreateArticles: migrated (0.0013s)
=============================================================
```

Run the Tests

Can we persist to the DB?

We need to explicitly migrate the test database

```
$ rspec spec
...
ActiveRecord::StatementInvalid:
  Could not find table 'articles'
...
```

Migrate the Test DB

Tests have their own database

```
$ rake db:test:prepare
```

Run the Tests

Can we persist to the DB?

```
$ rspec spec
Finished in 0.23919 seconds
2 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Working article model"
```

Validations
Data Integrity

Data integrity is critical

ActiveRecord provides a validations framework

Data integrity is critical

ActiveRecord provides a validations framework

So what next?

Write a Test

Is the data being validated?

```ruby
# spec/models/article_spec.rb
it "requires a title" do
  article = Article.new(:body => "World")
  expect(article).to_not be_valid
end
```

Run the Tests

Is the data being validated?

```
$ rspec spec
Failure/Error: expect(post).to_not be_valid
  expected valid? to return false, got true
```

Add Validation

Title Required!

```
class Article < ActiveRecord::Base
  ...
  validates_presence_of :title
end
```

Run the Tests

Is the data being validated?

```
$ rspec spec
Finished in 0.24452 seconds
3 examples, 0 failures
```

Article's require a Body

Tests first please!

Article Title Must Be Unique

Tests first please!

Hint: validates_uniqueness_of

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Add validations to Article model"
```

FactoryGirl
Object Factories

Less work for us

More readable for other devs

FactoryGirl
Object Factories

```
# Gemfile
group :test do
  # ...
  gem 'factory_girl_rails', '~> 4.1.0'
end
```

```
$ bundle
```

Fixture Path
Proper Configuration

```
# spec/spec_helper.rb
#...
#   config.fixture_path = "#{::Rails.root}/spec/fixtures"
```

Defining Factories

Object Factories

```
# spec/factories.rb
FactoryGirl.define do
  factory(:article) do
    sequence(:title) { |n| "Title-#{n}" }
    body "Lorem ipsum dolor sit amet, consectetur adipisicing"
  end
end
```

Using Factories

Object Factories

```
FactoryGirl.create(:post) #=> saves it to the database
FactoryGirl.build(:post) #=> Just initializes the model object
FactoryGirl.create(:post, :title => "New Title") #=> provide custom attributes
```

Testing Factories

Object Factories

```ruby
# spec/models/factories_spec.rb
require 'spec_helper'

describe 'Factories' do
  FactoryGirl.factories.each do |factory|
    it "#{factory.name} returns a valid object" do
      obj = FactoryGirl.build(factory.name)
      expect(obj).to be_valid
    end
  end
end
```

Shorter Factory Syntax

Object Factories

```
# spec/spec_helper.rb
RSpec.configure do |config|
  config.include FactoryGirl::Syntax::Methods
  # ...
```

```
create(:article)
build(:article, :title => "Foo Post")
```

Porting our specs to use FactoryGirl

Run the Tests
Factories Working Properly?

```
$ rspec spec
Finished in 0.3145 seconds
6 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Port specs to factory_girl"
```

Glob
The Plan

- Anyone can read articles
- Registered users can create articles
- Anyone can comment on articles

User Authentication
Devise Gem

Devise is feature-full, but modular Rails authentication framework

Much of the complexity of authentication is abstracted away from us by Devise (for better and worse)

We'll step through a reasonable amount of Devise's functionality to get a better understanding

User Authentication
Devise Gem

We'll write acceptance tests for the core behavior Devise gives us.

Writing these tests will illuminate much of what Devise is doing behind the scenes as well as give us confidence that our authentication system is functioning properly.

Quick Detour
simple_form

Rails default form-builder library is great - it let's us customize forms to our hearts' content.

But the new kids on the block have stolen the show with cleaner DSL's and integration with CSS frameworks like Bootstrap

simple_form
Better Rails Form Builder

```
# Gemfile
gem 'simple_form', '~> 2.0.4'
```

```
$ bundle
```

simple_form
Better Rails Form Builder

```
$ rails g simple_form:install --bootstrap
      create  config/initializers/simple_form.rb
      create  config/initializers/simple_form_bootstrap.rb
       exist  config/locales
      create  config/locales/simple_form.en.yml
      create  lib/templates/erb/scaffold/_form.html.erb
      ...
```

Run the Tests

Still Working?

```
$ rspec spec
Finished in 0.3145 seconds
6 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Install simple_form"
```

Devise
User Authentication

```
# Gemfile
gem 'devise', '~> 2.2.3'
```

```
$ bundle
```

Devise
User Authentication

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

Devise
User Authentication

```
# config/application.rb
module Glob
  class Application < Rails::Application
    # Recommended by Devise install generator
    config.assets.initialize_on_precompile = false
    ...
```

Run the Tests

Still Working?

```
$ rspec spec
Finished in 0.3145 seconds
6 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Install devise"
```

Devise
Generating a user model

```
$ rails g devise user name
      invoke  active_record
      create    db/migrate/2013xxxxxx_devise_create_users.rb
      create    app/models/user.rb
      insert    app/models/user.rb
       route  devise_for :users
```

Devise
The User Model

```
# app/models/user.rb

# Include default devise modules. Others available are:
# :token_authenticatable, :confirmable, :trackable,
# :lockable, :timeoutable, :recoverable and :omniauthable
devise :database_authenticatable, :registerable,
       :rememberable, :validatable
```

Devise

Updating the generated migration

```
## Recoverable
# t.string   :reset_password_token
# t.datetime :reset_password_sent_at

## Trackable
# t.integer  :sign_in_count, :default => 0
# t.datetime :current_sign_in_at
# t.datetime :last_sign_in_at
# t.string   :current_sign_in_ip
# t.string   :last_sign_in_ip

# add_index :users, :reset_password_token, :unique => true
```

Devise
Running the Migrations

```
$ rake db:migrate
...
$ rake db:rollback
...
$ rake db:migrate
...
$ rake db:test:prepare
...
```

Run the Tests

Still Working?

```
$ rspec spec
Finished in 0.3145 seconds
6 examples, 0 failures
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Generate devise user model"
```

RSpec/Capybara

Behavior-Driven Development

Describe how the software should behave by
describing how a real user would
interact with the website in a browser

Cucumber Criticisms

"Gherkin is really just glorified comments"

"Cucumber is a process disguised as a tool"

"Given-When-Then form hides dependencies"

"Overhead when the whole team isn't involved"

"Don't use Cucumber unless you live in the magic kingdom of non-programmers-writing-tests (and send me a bottle of fairy dust if you're there!)" - DHH

RSpec/Capybara

Let's get started!

```
# Gemfile
group :test do
  # ...
  gem 'database_cleaner', '~> 0.9.1'
end
```

```
$ bundle
```

Cucumber
Run install generator

```
$ rails g cucumber:install
      create  config/cucumber.yml
      create  script/cucumber
       chmod  script/cucumber
       exist  features/step_definitions
      create  features/support
      ...
```

Run the Tests

Still Working?

```
$ rspec spec && cucumber
Finished in 0.3145 seconds
6 examples, 0 failures

...
Using the default profile...
0 scenarios
0 steps
0m0.000s
```

Commit It
Version Control FTW

```
$ git add .
$ git commit -m "Install cucumber-rails"
```

Cucumber
Our First Scenario

The cucumber-rails gem is setup to integrate with Capybara and our test framework of choice, RSpec.

The step definitions will be aware of our application's named routes, models, factories, etc.

We'll use capybara to drive the UI and RSpec to make expectations on the page object (provided by capybara)

Cucumber
User can sign in

We can categorize our first few features to correspond with the Devise modules we're trying to test.

Opinions vary as the best way to organize features and step definitions.

We'll use cucumber scenarios to help us stitch together our app's UI with what Devise gives us

Cucumber
Our First Scenario

```
# features/registration.feature
Feature: User registration

  Background:
    Given I visit the home page

  Scenario: Users can sign up
    Given I navigate to the sign up page
    When I fill in the sign up form with valid data
    And I submit the sign up form
    Then I should be signed in
    And I should see confirmation that I signed up successfully
```

State can be shared between step definitions using instance variables which persist for the duration of a scenario.

A balance must be struck when sharing state between step definitions. Things can quickly get out of hand, especially with step definitions which are reused by many scenarios.

Low maintenance tests are the goal. Brittle test suites can become a pain and discourage thorough testing.

Cucumber
User can sign in

Avoid passing complex objects between step definitions.

Try to share state at the parameter level,
which will typically mean strings or lists of strings.


```
# features/sessions.feature
Feature: User Sessions

  Background:
    Given I visit the home page

  Scenario: Registered users can sign in
    Given I am a registered user
    When I navigate to the sign in page
    And I fill in the sign in form with valid data
    And I submit the sign in form
    Then I should be signed in
    And I should be notified that I successfully signed in
```

Cucumber
User can sign out

```
# features/session.feature
Feature: User Sessions

  Background:
    Given I visit the home page

  Scenario: Registered users can sign in and out
    Given I am a registered user
    When I navigate to the sign in page
    And I fill in the sign in form with valid data
    And I submit the sign in form
    Then I should be signed in
    And I should be notified that I successfully signed in
    When I click to sign out
    Then I should be signed out
    And I should be notified that I successfully signed out
```

Cucumber

User can dismiss flash notices

By default cucumber-rails run the scenarios without a JavaScript runtime.

If we're not testing behavior that requires JavaScript, we can use the defaults.

Otherwise, we can tag a scenario with @javascript, and it will open up an instance of Firefox and execute the steps in live browser with the JS runtime.

Cucumber

User can dismiss flash notices

```
Feature: Flash Notices

  Background:
    Given I visit the home page

  @javascript
  Scenario: User can dismiss flash notices
    Given there is a flash notice
    When I click to dismiss the flash notice
    Then I should not see the flash notice
```

Cucumber
Viewing Articles

Occasionally we'll want to declare a list of items for our scenarios to use.

Cucumber tables allow us to pass lists of data to our step definitions. The syntax is very simple. Just setup rows and columns and give columns a header.

The table will be yielded to the body of the step definition so we can iterate over the rows and create objects or make expectations.

Cucumber
Viewing Articles

Cucumber scenarios typically cut across many different parts of our application. It is often prudent to put a scenario in a pending state while we test and implement controller actions and even models.

By tagging a scenario with @wip (work-in-progress) we can safely commit the scenario and leave a note to ourselves and the team that this scenario is still being worked on.

Cucumber
Viewing Articles

```
Feature: Viewing Articles

  Scenario: Users can view a list of articles on the home page
    Given the following articles exist:
      | title          |
      | First Article  |
      | Second Article |
      | Third Article  |
    When I visit the home page
    Then I should see the list of articles
```

Cucumber
Viewing Articles

Controller Testing

Invoke the controller action, make an expectation on the response object.

```
require 'spec_helper'

describe ArticlesController do
  describe "GET :index" do
    it "is successful" do
      get :index
      expect(response.code).to eq "200"
    end
  end
end
```

Controller Tests
Viewing Articles

```
require 'spec_helper'

describe ArticlesController do
  describe "GET :index" do
    ...
    it "assigns all articles to @articles" do
      first = create(:article)
      second = create(:article)
      third = create(:article)

      get :index
      expect(assigns[:articles]).to include(first, second, third)
    end
  end
end
```

Controller Action
Viewing Articles

```ruby
# app/controllers/articles_controller.rb
class ArticlesController < ApplicationController
  def index
    @articles = Article.all
  end
end
```

Cucumber
Viewing Articles

```
Feature: Viewing Articles

  @wip
  Scenario: Users can view a list of articles on the home page
    Given the following articles exist:
      | title          |
      | First Article  |
      | Second Article |
      | Third Article  |
    When I visit the home page
    Then I should see the list of articles
```

Cucumber
Viewing Articles

```
Feature: Viewing Articles
   ...

  @wip
  Scenario: Users can view a single article
    Given an article exists
    When I visit the home page
    And I click to view the article
    Then I should see the article
```

Controller Tests
Viewing Articles

```
require 'spec_helper'

describe ArticlesController do
  describe "GET :show" do
    before do
      @article = create(:article)
    end

    it "is successful" do
      get :show, :id => @article.to_param
      expect(response.code).to eq "200"
    end
  end
end
```

Controller Tests
Viewing Articles

We'll need to test the controller actions in several contexts.

What is the behavior with/without an authenticated user?

RSpec gives us a `context` method, which let's us scope tests to a specific area of concern. It's behaves roughly the same as `describe` but the terminology is more fitting.

Controller Tests
Viewing Articles

```
require 'spec_helper'

describe ArticlesController do
  describe "GET :show" do
    before do
      @article = create(:article)
    end
    ....
    it "assigns the article to @article" do
      get :show, :id => @article.to_param
      expect(assigns[:article]).to eq @article
    end
  end
end
```

Controller Actions

before_filter's

```
class ArticlesController < ApplicationController
  before_filter do
    # Run this code before all actions in this controller
  end
  ...
end
```

Controller Actions
before_filter's

```
class ArticlesController < ApplicationController
  before_filter :my_method
  ...
  private

  def my_method
    # Run this code before all controller actions
  end
end
```

Controller Actions

before_filter's

```
class ArticlesController < ApplicationController
  before_filter :my_method, :except => [:index, :show]
  ...
  private

  def my_method
    # Run this code before all actions except 'index' and 'show'
  end
end
```

Cucumber
Creating an Article

```
Feature: Managing Articles

  Scenario: Authenticated users can view the new article page
    Given I am signed in
    When I navigate to the new article page
    Then I should see the new article form
```

Controller Tests
Creating an Article

```
describe "GET :new" do
  context "without an authenticated user" do
    it "denies access" do
      get :new
      expect(response).to redirect_to new_user_session_path
    end
  end
  context "with an authenticated user" do
  end
end
```

Controller Tests
Creating an Article

Setting up controller tests for authenticated user is fairly simple to do with Devise::TestHelpers.

We'll need to manually set the 'devise.mapping' key on the request object environment, so the controller test has access to the Devise configuration for the user model. In a running application this happens automatically.

Controller Tests
Creating an Article

```
describe "GET :new" do
  ...
  context "with an authenticated user" do
    before do
      @request.env['devise.mapping'] = Devise.mappings[:user]
      sign_in create(:user)
    end

    it "is successful" do
      get :new
      expect(response.code).to eq "200"
    end
  end
end
```

Cucumber
Creating an Article

```
Feature: Managing Articles

  Scenario: Authenticated users can create articles
    Given I am signed in
    And I navigate to the new article page
    When I fill in the new article form with valid data
    And I submit the new article form
    Then I should see the article
    And I should be notified that I successfully created and article
```

Controller Tests
Creating an Article

```
describe "POST :create" do
  ...
  context "without and authenticated user" do
    it "denies access" do
      post :create, @params
      expect(response).to redirect_to new_user_session_path
    end
  end
end
```

Controller Tests
Creating an Article

```
describe "POST :create" do
  context "with an authenticated user" do
    ...
    context "given invalid parameters" do
      it "renders the new template" do
        invalid_params = { :article => {
            :title => "", :body => "I had an amazing trip to Alaska" } }
        post :create, invalid_params
        expect(response).to render_template(:new)
      end
    end
  end
end
```

Controller Tests
Creating an Article

RSpec's expect(&block) method is useful for asserting changes in state around some specific behavior.

We could do this manually, but using expect(&block) is much more elegant and readable.

Controller Tests
Creating an Article

```
describe "POST :create" do
  ...
  context "with an authenticated user" do
    ...
    context "given valid parameters" do
      it "creates an article" do
        expect { post :create, @params }.to change(Article, :count).by(+1)
      end
      it "redirects to the created article's show page" do
        post :create, @params
        expect(response).to redirect_to article_path(Article.last)
      end
    end
  end
end
```

Model Tests

Articles have an author

```
describe Article do
  ...
  it "belongs to an author" do
    user = create(:user)
    article = create(:article, :author => user)
    expect(article.author).to eq user
  end
end
```

Controller Tests

Articles have an author

It is often necessary to access the current_user from within a controller test.

The controller tests gives us a `controller` method which returns the context of that controller. We'll call `controller.current_user` to access the authenticated user.

Controller Tests

Articles have an author

```
describe "POST :create" do
  ...
  context "with an authenticated user" do
    ...
    context "given valid parameters" do
      ...
      it "assigns the current user as the author" do
        post :create, @params
        expect(Article.last.author).to eq controller.current_user
      end
    end
  end
end
```

Cucumber

Articles have an author

```
Scenario: Users can view a single article
  Given an article exists
  When I visit the home page
  And I click to view the article
  Then I should see the article
  And I should see the author's name # additional step
```

Commenting

Users can comment on articles

Users should be able to comment on articles

Articles should have many comments

Comments should belong to an article

Commenting

Users can comment on articles

has_many/belongs_to association between articles and comments

Nested routes

Forms with nested routes

Commenting

Users can comment on articles

```
require 'spec_helper'

describe Comment do
  it "require some content" do
    comment = build(:comment, :content => '')
    expect(comment).to_not be_valid
  end

  it "belongs to an article" do
    article = create(:article)
    comment = create(:comment, :article => article)
    expect(comment.article).to eq article
  end
end
```

Commenting

Users can comment on articles

```
require 'spec_helper'

describe Article do
  ...
  it "has any comments" do
    article = create(:article)
    comment = create(:comment, :article => article)
    expect(article.comments).to eq [comment]
  end
end
```

Commenting

Users can comment on articles

```
Feature: Commenting on Articles

  Scenario: User can comment on articles
    Given an article exists
    And I visit the home page
    And I click to view the article
    When I fill out the comment form with "Great post!"
    And I submit the comment form
    Then I should the comment "Great post!" in the list of comments
```

Nested Route Parameters

Users can comment on articles

```
# routes
# article_comments POST /articles/:article_id/comments(.:format) comments#create

# params hash
params = {
  # id of the article we're commenting on
  :article_id => 42,
  # nested hash of values for the comment
  :comment => {
    :nickname => "#1 Fan",
    :content => "Very articulate!"
  }
}
```

Commenting

Users can comment on articles

```
describe CommentsController do
  describe "POST :create" do
    ...
    context "given valid params" do
      it "redirects to the article page" do
        post :create, @params
        expect(response).to redirect_to article_path(@article)
      end
      it "creates a comment on the article" do
        expect {
          post :create, @params
        }.to change(@article.comments, :count).by(+1)
      end
    end
  end
end
```

Commenting

Users can comment on articles

```
describe CommentsController do
  describe "POST :create" do
    ...
    context "given invalid params" do
      it "redirects to the article page" do
        invalid_params = {
          :article_id => @article.to_param,
          :comment => {
            :content => "" } }
        post :create, @params
        expect(response).to redirect_to article_path(@article)
      end
    end
  end
end
```

Thanks

http://integrallis.com