RSpec

Devpoint Labs - Dave Jungst / Jake Sorce

What is unit testing?

- Unit testing is testing the smallest parts of code (units), independently.
- Unit test helps to automate testing by making sure each function/method/unit is performing its desired task as expected.
- Unit testing aids in continuous integration by allowing us to build our projects by first running the test suite to ensure new code didn’t break existing functionality.
- Unit test aids in test driven development by allowing us to create tests for expected behavior prior to writing code.

TDD

TDD stands for test driven development

This means that tests for methods are written before methods and then the method is developed until it passes the test

TDD gives a clear picture of what a method is supposed to do and a better understanding of its functionality

TDD

First a test is written by stubbing some data and executing the method

```
it ‘doubles my number’ do
  x = 2
  expect(double_number(x)).to eq(4)
end
```

When the test is first executed it will fail because the method doesn’t exist. Then the method is written.

```
def double_number(num)
  num * 2
end
```

If the test passes then the method is done if it fails the method needs to be modified until it meets the criteria of the test.

Unit testing

By testing a single function the app can be broken down to its most basic functionality.

Remember a method should do only 1 thing and if that holds up then writing unit tests for all methods will help ensure the app is working correctly.

Unit tests should be written using “black box testing”

Black box testing means that we know only what the method produces or does and not how it works

Behavior Driven Development

Behavior driven development is a process that was developed based on Test Driven Development

BDD takes test driven development a step further by allowing you to describe your application.

Using a DSL built on top of the framework BDD gives a clear way through specifications or “specs” to explain the application and assert that it is executing properly.

RSpec

RSpec is a behavior driven framework for Ruby

It comes with its own mocking framework that is fully integrated into Ruby / Rails

RSpec can be used to test our models and controllers in isolation and scoped to their context

Like Ruby RSpec tests are written very closely to the english language to aid in readability and maintainability. You can look at an RSpec test and easily understand what it is testing and what the output should be.

RSpec on Rails

RSpec is not the only testing suite for Rails but it has become the most popular and most widely used.

Because of its popularity many other libraries have come along to improve the process of writing RSpec tests.

There are several Gems that are commonly used in conjunction with RSpec

Gemfile

Testing gems should go into a :development and :test group so that they are only loaded when we are developing or testing.

Gemfile.rb

```
#a bunch of gems

group :test, :development do

  gem ‘rspec-rails’

  gem ‘shoulda-matchers’

  gem ‘database_cleaner’

end
```

Install and set up

bundle

rails g rspec:install

From this point forward when resources are generated the spec files will be included. In existing apps specs will need to be added for models and controllers that already exist.

RSpec comes with a spec/rails_helper.rb file which can be used to configure RSpec although this is beyond the scope of this lecture you should know it exists and also know that it needs to be included in your specs

Testing Models

Testing Models with RSpec is the best place to start learning the RSpec syntax. Models are the easiest to test and also provide the most core functionality to a Rails app.

In order to put more focus on RSpec syntax and fat models we are going to put BDD aside and build a project and then write specs for the project.

RSpec Rails Project

```
rails new cars -d postgresql -T

cd cars

Gemfile.rb
#a bunch of gems
group :test, :development do
  gem 'rspec-rails'
  gem 'shoulda-matchers'
  gem 'database_cleaner'
  gem 'simplecov'
end

bundle

rails g rspec:install

rails g model car make:string model:string age:integer price:float mileage:integer color:string interior:string

bundle exec rake db:create db:migrate
```

Setting up simplecov

spec/spec_helper.rb

require ‘simplecov’

SimpleCov.start

DON’T FORGET TO PUT /coverage into your .gitignore

```ruby
class Car < ActiveRecord::Base
  def self.by_model
  end

  def self.by_make
  end

  def self.by_price(high = false)
  end

  def paint(color)
  end

  def info
  end

  def honk
  end
end
```

Logic that takes place on the car model should be placed inside of the model and removed from the controller.

This helps create skinny controllers and fat models.

Methods that start with self. are called directly on the class for example Car.by_make.

Methods without self. are called on an instance of the class for example @car.info

Validating attributes

spec/models/car_spec.rb

```
require ‘rails_helper’

RSpec.describe Car, type: :model do
  describe ‘attributes’ do
    it ‘has a make’ do
      make = ‘Toyota’
      car = Car.create(make: make)
      expect(car.make).to eq(make)
    end
  end
end
```

To run specs

To run all specs:

```
bundle exec rspec spec/
```

To run all specs in 1 file:

```
bundle exec rspec spec/models/car_spec.rb
```

To run a block:

```
bundle exec rspec spec/models/car_spec.rb:4
```

Cleaning up with shoulda-matchers

spec/models/car_spec.rb

```
require ‘rails_helper’

RSpec.describe Car, type: :model do
  describe ‘attributes’ do
    it ‘has a make’ do
      make = ‘Toyota’
      car = Car.create(make: make)
      expect(car.make).to eq(make)
    end
  end
end
```

spec/models/car_spec.rb

```
require ‘rails_helper’

RSpec.describe Car, type: :model do
  describe ‘attributes’ do
    it { should respond_to :make }
  end
end
```

Validation Testing

In models it is common to see validations for example

validates_presence_of :name

validates_uniqueness_of :name

Or even better

validates :name, uniqueness: true, presence: true

http://apidock.com/rails/ActiveModel/Validations/ClassMethods/validates

Method Testing

Let’s write specs for all of our methods!!

.rspec

--format progress

--format documentation