Test-Driven Development (TDD) with Rails 3 and rspec :part 1

October 19, 2012 Ruby on Rails

In this post I will show you step by step instruction for Test-Driven Development (TDD) with Rails 3.

Here is my full project on github.

So what is Test-driven development or TDD ?

Test-driven development (TDD) is a software development process that relies on the repetition of a very short development cycle: first the developer writes a failing automated test case that defines a desired improvement or new function, then produces code to pass that test and finally refactors the new code to acceptable standards.

First you should have Ruby installed on your system. Please check my post to see if you don't have Ruby on your system.

Create a Rails project

We will create a sample Task application for this tutorial.

By default, Rails uses Test Unit for its testing framework. But we will use something called rspec. So we will not install Rails default testing framework.

So lets create a Rails project by typing

```
rails new tasks -T -d mysql
```

Add New Gem to Gemfile

Now go to the project directory and open the Gemfile.rb with a text editor.

Then add those gem for this project.

```
group :test, :development do
  gem 'turn'
  gem 'rspec-rails'
  gem 'capybara'
  gem 'guard-rspec'
  gem 'rb-inotify', '~> 0.8.8'
  gem 'launchy'
end
```

Here we have created a group for our test and development. Now lets get some info about those gem

gem 'turn': TURN is a new way to view test results. With longer running tests, it can be very frustrating to see a failure (...F...) and then have to wait till all the tests finish before you can see what the exact failure was. TURN displays each test on a separate line with failures being displayed immediately instead of at the end of the tests.

gem 'rspec-rails': RSpec is a behaviour-driven development (BDD) tool for Ruby programmers. BDD is an approach to software development that combines test-driven development (TDD), domain-driven design (DDD), and acceptance test-driven planning (ATDP). RSpec helps you do the TDD part of that equation, focusing on the documentation and design aspects of TDD.

gem 'capybara': Capybara helps you test web applications by simulating how a real user would interact with your app. It is agnostic about the driver running your tests and comes with Rack::Test and Selenium support built in.WebKit is supported through an external gem.

gem 'guard-rspec': RSpec guard allows to automatically & intelligently launch specs when files are modified.

gem 'rb-inotify': This is a simple wrapper over the inotify Linux kernel subsystem for monitoring changes to files and directories. It uses the FFI gem to avoid having to compile a C extension.

gem 'launchy': Launchy is helper class for launching cross-platform applications in a fire and forget manner.
There are application concepts (browser, email client, etc) that are common across all platforms, and they may be launched differently on each platform. Launchy is here to make a common approach to launching external application from within ruby programs.

Install Gem:

Now run the command for install the gem

```
bundle install
```

or

```
bundle
```

Now you have to install rspec. To install rspec first check available generator for your system.

To check generator run this command –

```
rails generator
```

or

```
rails g
```

Then run this command –

```
rails g rspec:install
```

This will create a .rspec file, spec folder and spec/spec_helper.rb file

Now initiate Guard with rspec. To do this run this command –

```
guard init rspec
```

Running Test:

Now if we run

```
guard
```

This will test our spec. We can see that there is 0 examples and 0 failures.

Now we will install integration test for this project.

To install integration test run this command –

```
rails g integration_test tasks
```

This will create a folder request and request/tasks_spec.rb file

Write some test code:

Now open the request/tasks_spec.rb file and paste this code.

```
require 'spec_helper'

describe "Tasks" do

  describe 'GET /tasks' do

    it "display some tasks" do
      visit tasks_path
      page.should have_content
"go to bed"
    end

  end
end
```

Now if you run

```
guard
```

you see some test failure. We have to create required routes.

Create routes:

To create routs add this line to your config/routes.rb file.

```
resources :tasks
```

Now run –

```
rake routes
```

This will create RESTfull routes for tasks project.

Now if you run the test you see some error for routing. This caused because we don't have routing directory in our spec folder. So make a folder called spec/routing in our spec folder.

Create Controller:

To create our first controller run this command –

```
rails g controller Tasks index
```

Just look at the controller name. It's plural. It's a rails naming convention.

This command creates our Task controller and index action and also creates a index view which located at app/views/tasks/index.html.erb

That's all for the first part.

Second part of this tutorial is here.