Testing Sinatra

So much work around Ruby testing...
(not that much, actually)
it's time to give Sinatra some love!

IRON HACK

We already have a very good friend we've been using for testing...

IRON HACK

But first, we have to install some gems using our Gemfile

```
source 'https://rubygems.org'

gem 'rspec'
gem 'sinatra'
gem 'rack'
gem 'rack-test'
```

(Yeah, you probably already have the first two installed, so just add the other two)

Now, let's call our dear friend

```
$ bundle install
```

As we did before, we will create a new _spec file to test our server.rb

server_spec.rb

Ladies and gentlemen, please remember to save this file under your spec folder... yes, where your other spec files are

in your server_spec.rb file...

We have a few things to do here, don't we?

- require the sinatra server file
- set the right environment for testing

in your server_spec.rb file...

```ruby
require_relative '../server.rb'
require 'rspec'
require 'rack/test'

describe 'Server Service' do
  include Rack::Test::Methods

  def app
    Sinatra::Application
  end
end
```

This is your typical sentence to describe the class you're testing

The Rack::Test::Methods module includes a variety of helper methods for simulating requests against an application and asserting expectations about the response. It's typically included directly within the test context and makes a few helper methods and attributes available.

in your server_spec.rb file...

We have a few things to do here, don't we? (part II)

- add some tests to your file

(the amusing part)

in your server_spec.rb file...

```ruby
it "should load the home page" do
  get '/'
  expect(last_response).to be_ok
end

it "should not load the home page" do
  get '/home'
  expect(last_response).to_not be_ok
end
```

```ruby
it "should load the other page" do
  get '/real_page'
  expect(last_response).to be_ok
end

it "should redirect to the real_page" do
  get '/hi'
  expect(last_response.redirect?).to be(true)
  follow_redirect!
  expect(last_request.path).to eq('/real_page')
end
```

By now, we have a lot of server.rb files in our computer... but let’s not be lazy and let's create a new one that matches our spec file.

Your server.rb file should be something like this

```ruby
require 'sinatra'

get '/' do
  'Hello, World!'
end

get '/real_page' do
  'The other page'
end

get '/hi' do
  redirect 'real_page'
end
```

Bring up a terminal, redirect yourself to your project folder and start

Our terminal output should be something like this...

```
Server Service
  should load the home page
  should not load the home page
  should load the other page
  should redirect to the real_page

Finished in 0.02917 seconds (files took 0.4077 seconds to load)
4 examples, 0 failures
```

And if you got some failures, now is time to fix them :)

Now we know how to perform automated tests for Sinatra...

... What about automatize the execution of automated tests?

What?!?!?!!?!

Let me introduce you a new gem...

Guard will help you to handle events on file system modifications

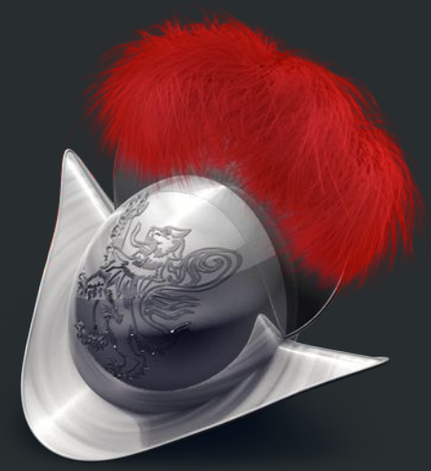
In english, please?

Guard is a command line tool, so we'll have to execute it from our terminal.

It will enable us to keep making changes in our code. Each time we type a change, Guard will execute all our tests

Automatically!!!

Guard documentation



First of all, let's go to:

http://guardgem.org/

The simplest way to install Guard is to use Bundler.

Add Guard (and any other dependencies) to our `Gemfile` in your project's root:

```
gem 'guard'
gem 'guard-rspec'
```

We will be using Guard with rspec

then install it by running Bundler:

```
$ bundle install
```

IRON HACK

Generate a `Guardfile` with this code:

```
guard :rspec, cmd: 'bundle exec rspec --color' do
  watch(%r{^spec/.+_spec\.rb$})
  watch(%r{^lib/(.+)\.rb$}) { |m| "spec/#{m[1]}_spec.rb" }
  watch("server.rb") {|m| "spec/server_spec.rb"}
end
```

Run Guard through Bundler with:

```
$ guard -i
```

IRON HACK