skillcrush

TESTING WITH RSPEC

cheatsheet

RSPEC

RSpec is a testing framework specifically designed for writing tests for Ruby and Rails. RSpec is designed to make **TDD** (test driven development) easy and fun!

Learn more here.

GETTING STARTED WITH RSPEC

In order to use RSpec with your Rails applications you will need to make sure to include the following in your Gemfile:

```
gem 'rspec'
gem 'rspec-rails'
```

And then make sure to run a `bundle install`!

You will want to organize your tests by folder, for example, controller tests should go inside of your `spec > controllers` folder.

TESTING WITH RSPEC

cheatsheet

Then you will need to have a `spec_helper.rb` file inside your spec folder that configures your testing suite:

```
ENV["RAILS_ENV"] ||= 'test'
require File.expand_path("../../config/environment", __FILE__)
require 'rspec/rails'

RSpec.configure do |config|
  config.infer_spec_type_from_file_location!

  config.before(:suite) do
    DatabaseCleaner.strategy = :transaction
    DatabaseCleaner.clean_with(:truncation)
    Rails.application.load_seed # loading seeds
  end
end
```

This is just sample code, your spec_helper.rb file will vary for each individual app.

TESTING WITH RSPEC

cheatsheet

Finally, each one of your test files should look like this:

```
require 'spec_helper' # makes sure your spec helper file is loaded
RSpec.describe PinsController do
   # your test code & expectations go here
end
```

To run your test you need to hop into your terminal and run:

```
rspec spec/controllers/pins_controller_spec.rb
```

And just change the file path and filename depending on the spec you want to run!

skillcrush

TESTING WITH RSPEC

cheatsheet

STARTING YOUR TEST

After your `RSpec.describe PinsController do` line you can begin to write your tests. Start by describing WHAT you are testing:

```
describe "GET index" do
    # here is where your test code will go
end
```

This code says "here I will describe what I expect by GET index controller action to do.

Then you can actually DESCRIBE what you expect:

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

The first line says what you expect in plain English. Then you ask RSpec to run whatever action you need done, and finally you write your expectation. The expectation line is the TRUE core of the RSpec testing framework, keep reading to learn more about them!

skillcrush

TESTING WITH RSPEC

cheatsheet

RSPEC EXPECTATIONS

In RSpec an "expectation" is a line of code you write that describes what you expect will happen. You have two options, you can either say that you expect something something to match:

```
expect(actual).to matcher(expected)
```

OR you can say that you expect something NOT to match:

```
expect(actual).not_to matcher(expected)
```

Your expectation must have the following:

- ❏ ALWAYS begin with the keyword **`expect`**
- ❏ followed by what you expect to happen (where 'actual' sits in the ex. above)
- ❏ then that you expect it **`to`** match or **`not_to`** match the expected 'actual'

For example, this line checks that 4 + 5 is equal to 9:

```
expect(4+5).to eq(9);
```

TESTING WITH RSPEC

cheatsheet

On the other hand, this line checks that 4 + 4 does NOT equal 9:

```
expect(4+5).to eq(9);
```

MATCHER METHODS

Matchers are methods that check that the 'actual' matches the 'expected'.

Many of the matching methods you can use are the same ones you can use in any other Ruby situation.

eq

Tests whether the actual is equal to the expected.

```
expect(order.total).to eq(Money.new(5.55, :USD)
```

be

Tests whether the actual is the specified value

```
expect("this string").to be_a(String)
```

include

Tests whether the actual includes the specified value

```
expect("this string").to include("is str")
```

skillcrush

TESTING WITH RSPEC

cheatsheet

start_with

Tests whether the actual starts with the specified value

```
expect("this string").to start_with("this")
```

end_with

Tests whether the actual ends with the specified value

```
expect("this string").to end_with("ring")
```

contain_exactly

Tests whether the actual contains exactly the specified value

```
expect([1, 2, 3]).to contain_exactly(2, 3, 1)
```

match_array

Tests whether the actual matches a specified array

```
expect([1, 2, 3]).to match_array([3, 2, 1])
```

raise_error

tests whether the actual raises a specific error

```
expect { error }.to raise_error("this is an error message!")
```

For a more complete list of matcher methods, checkout the RSpec documentation.

skillcrush

TESTING WITH RSPEC

cheatsheet

RAILS SPECIFIC MATCHER METHODS

Some matcher methods are specific to certain Rails scenarios, for example, here are matcher methods you can use for tests on your controllers:

render_template

Checks to see what template is rendered during a specific controller action

```
expect(view).to render_template(:partial => "_form")
```

redirect_to

Checks where a controller action is redirected to

```
expect(response).to redirect_to("/people")
```

route_to

Checks what route a controller action routes to

```
expect(:get => "/people").to route_to(:controller => "people")
```

have_http_status

Checks the HTTP status of a controller action

```
expect(response).to have_http_status(201
```

For a more complete list, checkout the RSpec documentation on Rails specific matchers.

skillcrush

TESTING WITH RSPEC

cheatsheet

MORE EXPECTATION EXAMPLES

This expectation says "I expect the array [1, 2, 3] to include the number 1:

```
expect([1, 2, 3]).to include(1)
```

This expectation says "I expect the game score to equal 0":

```
expect(game.score).to eq(0)
```

This expectation says "I expect the controller action to provide a response that renders the show template":

```
expect(response).to render_template("show")
```

This expectation says "I expect the controller action to provide a response that has the HTTP status of 201":

```
expect(response).to have_http_status(201)
```