Capybara

Feature Specs Overview

When we wrote our model spec tests, our tests covered just the functionality of our model. Similarly, if we had written controller or route specs, we would only be testing the functionality of our controllers or routes. This is a code-base-centric way of testing our application. These spec tests test the functionality of our application as they are organized in our code base.

Feature specs are a little different; they represent a user-centric way of testing our application. Instead of testing a specific section of our code base, with feature specs we test the flow of inputs and outputs as seen by a user.

As far as an end user of your web application is concerned, this is what a *feature* is. When he types some text and clicks on a button(the input), the web page changes and presents him a different view(the output).

Feature specs often span different sections of your code. A feature spec will often traverse through your view, your controller, your route, and your model all in one test.

Capybara

To help us write our feature specs we will be using a tool called *Capybara*. Capybara helps us test our web application by simulating how a human would interact with our app (like visiting a URL, clicking a link, typing text into a form and submitting it).

Capybara Setup

In this lesson, we are going to continue to build off of our Example 1 from the ModelSpec lesson.

Add Capybara as a dependency in your gemfile:

Gemfile

```
group :development, :test do
  gem 'rspec-rails'
  gem 'capybara'
end
```

and run `$bundle install`

That's it, you are now all set to use Capybara!

Create the Users Controller and Route

Lets create a users controller and route for us to use with the capybara spec we will be writing.

Terminal

```
$ rails g controller Users index
```

routes.rb

```
resources :users
```

Running our First Capybara Spec Test

RSpec automatically adds the *"/controllers"* folder and the *"/controllers_users_controller_spec.rb"* file within our *"spec"* folder. Replace the code currently in *"users_controller_spec.rb"* with the code below.

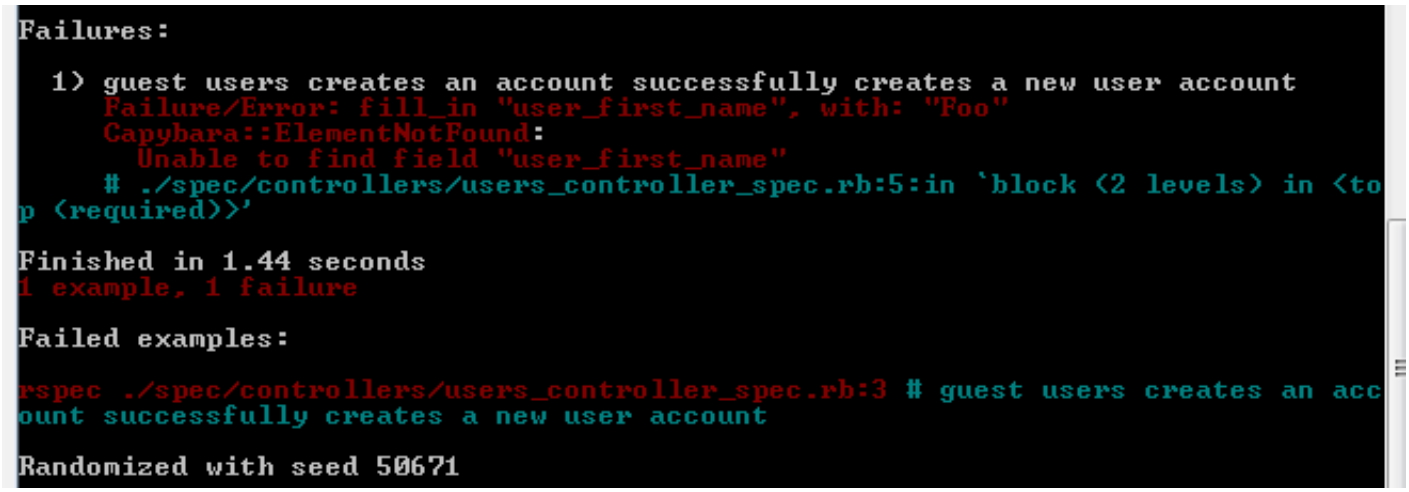
spec/controllers/users_controller_spec.rb

```
require 'rails_helper'
feature "guest user creates an account" do
  scenario "successfully creates a new user account" do
    visit "users#index"
    fill_in "user_first_name", with: "Foo"
    fill_in "user_last_name", with: "Bar"
    fill_in "user_email", with: "foo@bar.com"
    click_button "Create User"
    expect(page).to have_content "User successfully created"
  end
end
```

Lets run our capybara test

Terminal
`$ rspec spec/controllers/users_controller_spec.rb`

We should get the following result:



That is it, you have run your first capybara spec test!

Understanding our First Capybara Spec Test

We have introduced some new syntax in this capybara spec. All of the following are part of Capybara's DSL(Domain Specific Language).

- `feature` and `scenario` are Capybara's version of `describe` and `it` respectively. They serve the same purpose as describe and it. Using `feature` instead of `describe` lets Rails know that you are writing a capybara spec.
- `visit` navigates to a particular path. You can pass a string or use one of the Rails path helpers.
    - `visit '/blog'`
    - `visit blogs_path`
- `click_button` will press a button or input[type="submit"]
- `have_content` asserts that certain text content is present on the page.
- `fill_in` will fill in fields for you. You can select what field to fill in using the label text or the name of the input.
    - fill_in "Title", with: "I love rails!"
    - fill_in 'post[title]', with: "I love rails!"

Here's a cheat sheet for Capybara - read more

Now lets look at the error we got. We got the error `Unable to find field "user_first_name"` because we haven't created that field in our users view yet.

Adding fields to our view file and @user to users_controller

Lets add some fields to our users view. And while we are at it, lets also add an @user to the index method of our UsersController for the view file to reference.

/app/views/users/index.html.erb
NOTE: There is nothing special about this view's codes. This is just the basic form to create a new user in Rails)

```
<h1>Create new User</h1>
<p id="notice"><%= notice %></p>
<%= form_for(@user) do |f| %>
  <div class="field">
    <%= f.label :first_name %><br>
    <%= f.text_field :first_name %>
  </div>
  <div class="field">
    <%= f.label :last_name %><br>
    <%= f.text_field :last_name %>
  </div>
  <div class="field">
    <%= f.label :email %><br>
    <%= f.text_field :email %>
  </div>
  <div class="actions">
    <%= f.submit %>
  </div>
<% end %>
<%= link_to 'Back', users_path %>
```

/app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def index
    @user = User.new
  end
end
```

Run the capybara test again.

This will result in another failure; this time with the message
`The action 'create' could not be found for UsersController.`

What this means is that the test is failing because there is no *create* action. Capybara clicked the button "Create User" which submitted the form which routes us to the create action of our users*controller. However this method doesn't exist yet on our users*controller.

Adding a create method

Lets make this test succeed by adding a *create* method in our UsersController.

/app/controllers/users_controller.rb

```
def create
  @user = User.new(params.require(:user).permit(:first_name, :last_name, :email))
  if @user.save
    flash[:notice] = 'User successfully created'
    redirect_to action: 'index'
  else
    #errors we need to code later
  end
end
```

Lets run the capybara test again.

You should get a success message. Our capybara test has passed!