Capybara

Feature Specs Overview

When we wrote our model spec tests, our tests covered just the functionality of our model. Sim
route specs, we would only be testing the functionality of our controllers or routes. This is a coc
application. These spec tests test the functionality of our application as they are organized in ou

Feature specs are a little different; they represent a user-centric way of testing our application.
section of our code base, 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 ty
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 thro
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 te
how a human would interact with our app (like visiting a URL, clicking a link, typing text into a f

Capybara Setup

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

Add Capybara as a dependency in your gemfile:

Gemfile

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

and run `dojo$ 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 add routes for us to use with the capybara spec we will be wr

```
dojo$ rails g controller Users new
```

config/routes.rb

```
  resources :users
```

Running our First Capybara Spec Test

Now let's create a folder where we can put our feature specs. From the root of your rails projec

```
dojo$ mkdir spec/features
dojo$ touch spec/features/register_user_spec.rb
```

From now on whenever we create a new feature spec we will be adding it to our features folder
create subfolders within this folder. For example, we can have a folder for admin features and u

spec/features/register_user_spec.rb

```
require 'rails_helper'
feature "guest user creates an account" do
  scenario "successfully creates a new user account" do
    visit new_user_path
    fill_in "user_first_name", with: "shane"
    fill_in "user_last_name", with: "chang"
    fill_in "user_email", with: "schang@codingdojo.com"
    click_button "Create User"
    expect(page).to have_content "User successfully created"
  end
end
```

Lets run our capybara test

Terminal

```
rspec spec/features/register_user_spec.rb
```

Your output should look like the following:

Terminal Output

```
F

Failures:

  1) guest user creates an account successfully creates a new user account
     Failure/Error: fill_in "user_first_name", with: "shane"

     Capybara::ElementNotFound:
       Unable to find field "user_first_name"
     # ./spec/features/register_user_spec.rb:5:in `block (2 levels) in <top (required)>'

Finished in 1.01 seconds (files took 2.98 seconds to load)
1 example, 1 failure

Failed examples:

rspec ./spec/features/register_user_spec.rb:3 # guest user creates an account successfully creates
```

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 Cap Language).

- `feature` and `scenario` are Capybara's version of `describe` and `it` respectively. They serve 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 he
  - `visit '/blog'`
  - `visit blogs_path`
- `click_button` will press a button or input[type="submit"]. You can select what button to p
- `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, the
  - fill_in "Title", with: "I love rails!"
  - fill_in 'post[title]', with: "I love rails!"

Here's a cheat sheet for Capybara - read more (https://upcase.com/test-driven-rails-resources,

Now lets look at the error we got. We got the error `Unable to find field "user_first_name"` beca our users view yet.

Adding fields to our view file and @user to users_controller

Following RESTFUL let's add a form to our *new.html.erb* file with the fields we specified in our s or the name of the inputs match with the names we provided for our spec test.

/app/views/users/new.html.erb

```
<h1>Create new User</h1>
<% if flash[:notice] %>
 <% flash[:notice].each do |note| %>
    <p id="notice"><%= note %></p>
 <% end %>
<% end %>
<form action = "/users" method = "post">
  <input type = 'hidden' name = "authenticity_token" value = "<%=form_authenticity_token%>">
 <label>First Name</label>
 <input type = "text" name = "user[first_name]" id = "user_first_name">
  <label>Last Name</label>
  <input type="text" name="user[last_name]" id = "user_last_name">
  <label>Email Address</label>
  <input type="text" name="user[email]" id = "user_email">
  <input type = "submit" value = "Create User">
</form>
```

/app/controllers/users_controller.rb

```
class UsersController < ApplicationController
  def 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 fo`

What this means is that the test is failing because we haven't specified a *create* method within c clicked the button "Create User" which submitted the form which routes us to the create metho

Adding a create method

Lets make this test succeed by adding a *create* method to 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 new_user_path
  else
    #errors we need to code later
  end
end
```

Lets run the capybara test again.

You should now see a success message in your terminal. Our capybara test has passed!

Terminal Output

```
.

Finished in 0.30183 seconds (files took 2.73 seconds to load)
1 example, 0 failures
```


PREVIOUS (/M/9/4310/31486)