Rails Testing Guide

General

TimeCop Usages

```
around(:example) do |example|
 Timecop.freeze(today) do
   example.run
 end
end

Timecop.freeze do
 expect(subject.remote_created_at.to_i).to eq(Time.now.to_i)
 expect(subject.remote_updated_at.to_i).to eq(Time.now.to_i)
end
```

Scheduling

```
RSpec.describe Model do
  describe "#perform_later" do
    it "uploads a backup" do
      expect {
        model.schedule_job
      }.to
have_enqueued_job.with('backup').on_queue("low").at(Date.tomorrow.noon)
    end
  end
end
```

Model Testing

- Ensure to test all custom methods
- Ensure to test any custom validators
- Ensure to test any complex unique constraints
- Ensure to test with_priviledge and use this as a good example
- Ensure to test all scopes fully

```
describe User, '.active' do
  it 'returns only active users' do
    # setup
    active_user = create(:user, active: true)
    non_active_user = create(:user, active: false)
```

```
    # exercise
    result = User.active

    # verify
    expect(result).to eq [active_user]

    # teardown is handled for you by RSpec
  end
end
```

Matchers

```
expect(thing).to be_valid
expect(thing).to be_invalid
expect(user.errors.full_messages.first).to eql('Email has already
been taken')
```

Validations

- Ensure to test the boundaries and not the rails in-built functions directly e.g. no need for shoulda matchers
- Ensure to test the foreign key dependencies
- Ensure to test on delete
    - t.*belongs_to* **:model foreign_key**: { **on_delete**: **:cascade** }
    - t.*belongs_to* **:model**, **foreign_key**: { **on_delete**: **:nullify** }
    - t.*belongs_to* **:model**, **foreign_key**: { **on_delete**: **:restrict** }

Relationships

- Ensure to filter out deleted objects from the has_many by default
- When dealing with potentially many records on a relation

```
scope :active, -> { where(status: ACTIVE) }
has_many :user_organizations, -> { active }, autosave: true, inverse_of: :user
```

Testing standard rails functions

```
FactoryGirl.create :user, email: 'admin@example.com'
user = FactoryGirl.build :user, email: 'admin@example.com'
expect(user.errors.full_messages.first).to eql('Email has already been taken')
```

Schema Testing

- Do not use dependant: :destroy if you have a foreign_key: {**on_delete**: **:cascade**} on the schema

On delete

Cascade

```
# t.belongs_to :time_off_policy, foreign_key: { on_delete: :cascade }
describe 'validations' do
 context 'schema' do
   before { time_off_policy }

   context 'time_off_policy relationship' do
     it 'should remove time off when policy deleted' do
       expect { time_off_policy.destroy }.to change(TimeOffPolicyUser, :count).by(-1)
     end
   end
 end
end
```

Nullify

```
# t.integer :created_by_id, foreign_key: { references: :users, on_delete: :nullify }
describe 'validations' do
 context 'schema' do
   before { time_off_policy }

   context 'user relationship' do
     it 'should nullify created_by_id when associated user is deleted' do
       expect { user.destroy }.to change { time_off_policy.reload.created_by_id
}.from(user.id).to(nil)
     end
   end
 end
end
```

Restrict

```
# t.belongs_to :time_off_policy, index: true, foreign_key: { on_update: :restrict,
on_delete: :restrict }
describe 'validations' do
 context 'schema' do
   context 'time_off_requests relationship' do
     it 'should not remove time off when policy deleted' do
       FactoryGirl.create :time_off_request, time_off_policy: time_off_policy

       expect { time_off_policy.destroy }.to raise_error ActiveRecord::InvalidForeignKey
     end
   end
 end
end
```

Controller Testing

- Ensure to include security checks (fully understand how to use declarative authorization) summarized here

```
describe 'security checks' do
 context 'as an org owner' do
   let(:org_role) {:owner}
 end

 context 'as an org manager' do
   let(:org_role) {:manager}
 end

 context 'as an org user' do
   let(:org_role) {:user}

   context 'and not a project member' do
   end

   context 'and a project user' do
     let(:proj_role) {:user}
   end

   context 'and a project manager' do
     let(:proj_role) {:manager}
   end

   context 'and a project viewer' do
     let(:proj_role) {:viewer}
   end
 end
end
```

Input permutations

```
describe 'GET #index' do
  context 'when params[:filter_by] == first_name' do
    it 'filters results by first_name'
  end
```

```
  context 'when params[:filter_by] == last_name' do
    it 'filters results by last_name'
  end
end
```

Valid vs invalid params

```
describe 'POST #create' do
  context 'with valid params' do
    it 'redirects to show page'
  end
  context 'with invalid params' do
    it 're-renders #new form'
  end
end
```

Authenticated access

```
describe 'GET #index ' do
  context 'when user is logged in' do
    it 'renders the listing page'
  end
  context 'when user is logged out' do
    it 'redirects to login page'
  end
end
```

Authorized access

```
describe 'GET #show' do
  context 'as content owner' do
    it 'renders the permalink template'
  end
  context 'as an admin' do
    it 'renders the permalink template'
  end
  context 'as a guest user' do
    it 'displays access forbidden message'
  end
end
```

Matchers

```
expect(response).to redirect_to(location)
expect(response).to have_http_status(:created)
expect(assigns(:widget)).to be_a_new(Widget)
expect(assigns(:product)).to eq(bestseller)
expect(assigns(:widgets)).to eq([widget1, widget2, widget3])
expect(flash[:notice]).to eq "Congratulations on buying our stuff!"

# change in active record
expect { post_with user, :activity }.to change(Activity, :count).by(1)

# change static method
Date.beginning_of_week = :sunday
expect { controller.send(:reset_beginning_of_week) }.to change(Date,
:beginning_of_week).to(:monday)

# change in instance method
expect {
 delete_with org_owner, :destroy, params
 client.reload
}.to change(client, :status).to(Client::STATUS_ARCHIVED)
```

View Testing

- You want to test all logical paths
    - Authorization
    - Contents of elements

```
# subject should always render the view and return rendered which returns the resultant
view as text
subject { render partial: 'dashboard/new_dashboard_alerts.html.slim'; rendered }

before do
 assign(:delete_screens_allowed, [11]) # artificially assign any required params
 subject
end

# rendered view includes slicer
expect(subject).to match /slicer/

expect(subject).to have_css('.state_shot.no_screenshot')

# you can use this for example when you are not a manager / owner logic
expect(rendered).to_not have_css('.state_shot.browser_time')

# RSpec matcher for whether the element(s) matching a given selector exist
```

```
expect(rendered).to have_selector("li a", text: 'Overview')

# RSpec matcher for text on the page
expect(subject).to have_text(',"Sun, Dec 3, 2017 12:00 pm"]')

# RSpec matcher for links
expect(rendered).to have_link('Cancel', href: invoices_url)

assert_select 'td', text: "#{project.name} Archived"
```

Mailer Testing

```
RSpec.describe SomeMailer do
  let(:mail)      { ProjectMailer.send_something(arg1, arg2) }
  let(:text_part) { mail.text_part.body.raw_source }
  let(:html_part) { mail.html_part.body.raw_source }

  it 'renders the headers' do
      expect(subject.subject).to eq('A new payment from Org A was
completed')
      expect(subject.from).to eq(['support@hubstaff.com'])
      expect(subject.reply_to).to eq(['support@hubstaff.com'])
      expect(subject.bcc).to eq([owner.email])
  end

it 'should have a link to view budget' do
        expect(html_part).to match(/View budget/)
        expect(html_part).to match(/#{project_url(project)}/)
      end

  it 'should have a link to project page' do
    expect(text_part).to match(/#{project_url(project)}/)
  end
end
```

Job Testing

- Ensure to test the scheduling of a job

```
#### ActiveJob
RSpec.describe Model do
  describe "#perform_later" do
    it "uploads a backup" do
```

```
      expect {
        model.schedule_job
      }.to
have_enqueued_job.with('backup').on_queue("low").at(Date.tomorrow.noon)
    end
  end
end

#### Sidekiq
it 'creates a background job' do
  expect{ subject }.to change(Sidekiq::Extensions::DelayedMailer.jobs,
:size).by(1)
end
```

Functional / Integration Testing

General

Integration is known also as black box testing which essentially means you test without the need for prior knowledge of how it works. We are using Capybara and Rspec to fulfill the role of testing the UI interactions and bridges the gap between ruby & JS testing.

I am going to step you through how to create integration specs.

1. We generally create one spec per page e.g. /login interactions go into its own spec
    a. e.g. login_page_spec.rb
2. When creating capybara specs it is important to have a good understanding of where and when to use asynchronous calls vs synchronous calls in order to ensure the specs run in a timely fashion.
3. Ensure to extract common functionality into methods or if you are using them across multiple specs think of adding them spec/support/features.rb
4. Follow this as an excellent example for creating Capybara specs

Capybara Toolbox

Here's a quick overview of methods that do wait:

- find, find_field, find_link and other finders
- within
- has_selector?, has_no_selector? and similar
- click_link, fill_in, check, select, choose and other actions

methods that don’t wait:

- visit(path)
- current_path
- all(selector)
- first(selector)
- execute_script, evaluate_script
- simple accessors: text, value, title, etc

Directly interacting with JavaScript

```
find(".active") # Ensures the element is present before continuing
execute_script("$('.active').focus()") # only use execute_script
as last resort when capybara does not support what you are trying
to achieve
```

Checking a fields value

```
expect(page).to have_field("Username", with: "Joe") # wait until
matching field has value or fail if exceeds wait time
```

Checking an element’s attribute

```
expect(page).to have_css(".user[data-name='Joe']") # capybara will
wait for the element to appear and verifies whether or not it has
the expected value
```

Looking for matching CSS

```
it "doesn't have an active class name" do
  expect(page).not_to have_active_class # passes immediately if
not on the page or waits up to the time wait for it to disappear
end

def have_active_class
  have_css(".active")
end
```

Navigating

```
visit root_path
```

Clicking links and buttons

```
click_link('id-of-link')
click_link('Link Text')
click_button('Save')
click_on('Link Text') # clicks on either links or buttons
click_on('Button Value')
```

Interacting with forms

```
fill_in('First Name', with: 'John')
fill_in('Password', with: 'Seekrit')
fill_in('Description', with: 'Really Long Text...')
choose('A Radio Button')
check('A Checkbox')
uncheck('A Checkbox')
attach_file('Image', '/path/to/image.jpg')
select('Option', from: 'Select Box')
```

Querying

```
page.has_selector?('table tr')
page.has_selector?(:xpath, './/table/tr')

page.has_xpath?('.//table/tr')
page.has_css?('table tr.foo')
page.has_content?('foo')

# Examples
expect(page).to have_selector('table tr')
expect(page).to have_selector(:xpath, './/table/tr')

expect(page).to have_xpath('.//table/tr')
expect(page).to have_css('table tr.foo')
```

```
expect(page).to have_content('foo')
```

Finding

```
find_field('First Name').value
find_field(id: 'my_field').value
find_link('Hello', :visible => :all).visible?
find_link(class: ['some_class', 'some_other_class'], :visible =>
:all).visible?

find_button('Send').click
find_button(value: '1234').click

find(:xpath, ".//table/tr").click
find("#overlay").find("h1").click
all('a').each { |a| a[:href] }

# Examples
find_field('First Name'){ |el| el['data-xyz'] == '123' }
find("#img_loading"){ |img| img['complete'] == true }

find('#navigation').click_link('Home')
expect(find('#navigation')).to have_button('Sign out')
```

Scoping

```
within("li#employee") do
  fill_in 'Name', with: 'Jimmy'
end

within(:xpath, ".//li[@id='employee']") do
  fill_in 'Name', with: 'Jimmy'
end

# Examples
within_fieldset('Employee') do
  fill_in 'Name', with: 'Jimmy'
end

within_table('Employee') do
  fill_in 'Name', with: 'Jimmy'
end
```

Working with windows

Capybara provides some methods to ease finding and switching windows:

```
facebook_window = window_opened_by do
  click_button 'Like'
end
within_window facebook_window do
  find('#login_email').set('a@example.com')
  find('#login_password').set('qwerty')
  click_button 'Submit'
end
```

Scripting

```
page.execute_script("$('body').empty()")
result = page.evaluate_script('4 + 4');
```

Modals

```
accept_alert do
  click_link('Show Alert')
end

dismiss_confirm do
  click_link('Show Confirm')
end

accept_prompt(with: 'Linus Torvalds') do
  click_link('Show Prompt About Linux')
end

message = accept_prompt(with: 'Linus Torvalds') do
  click_link('Show Prompt About Linux')
end
expect(message).to eq('Who is the chief architect of Linux?')
```

Debugging

```
save_and_open_page # take screenshot when page opens
print page.html # print snapshot of DOM
page.save_screenshot('screenshot.png')
Save_and_open_screenshot # save screenshot & open file
```

Extra Resources

- https://robots.thoughtbot.com/write-reliable-asynchronous-integration-tests-with-capybara
- http://www.rubydoc.info/github/jnicklas/capybara
- https://www.varvet.com/blog/simple-tricks-to-clean-up-your-capybara-tests/
- https://www.simplybusiness.co.uk/about-us/tech/2015/02/flaky-tests-and-capybara-best-practices/
- https://www.sitepoint.com/basics-capybara-improving-tests/
- https://semaphoreci.com/community/tutorials/5-tips-for-more-effective-capybara-tests