INTERACTORS

Erich Kaderka
Ruby Developer @ PrimeHammer
https://twitter.com/kaderka
https://github.com/erich

WHAT IS AN INTERACTOR?

- Implementation of command pattern
- Interactor is simple, single purpose object
- Interactor orchestrates components in the system to complete a specific business use case

WHEN AND WHERE WE USE THEM?

- Most common use cases lay in controllers

```ruby
class SignupController < ApplicationController

  def create
    @user = User.new(user_params)
    if @user.valid?
      @user.save
      NewUserJob.send_email(@user.id)
      AdminJob.send_new_user_notification(@user.email)
      redirect_to dashboard_path
    else
      render :new
    end
  end

  private

  def user_params
    params.require(:user).permit(:first_name, :last_name, :email, :password)
  end
end
```

EXAMPLE OF CONTROLLER SPEC

+ s/c/signup_controller_spec.rb

```
3  RSpec.describe SignupController, type: :controller do
4
5    describe 'POST create' do
6      it 'creates a valid user' do
7        post :create, {  user: { first_name: 'Joe', last_name: 'Doe',
8                                  email: 'joe.doe@example.com', password: 'test' } }
9        expect(User.find_by(email: 'joe.doe@example.com').first_name).to eq 'Joe'
10     end
11
12     it 'redirects to dashboard after a valid user is created' do
13       post :create, {  user: { first_name: 'Joe', last_name: 'Doe',
14                                  email: 'joe.doe@example.com', password: 'test' } }
15       expect(response).to redirect_to(dashboard_url)
16     end
17
18     it 'sends notification to admin' do
19       expect(AdminJob).to receive(:send_new_user_notification)
20       post :create, {  user: { first_name: 'joe', last_name: 'doe',
21                                  email: 'joe.doe@example.com', password: 'test' } }
22     end
23     ## more specs, e.g. test NewUserJob
24   end
25
26   it "doesn't create an invalid user" do
27       post :create, {  user: { first_name: 'joe', last_name: '',
28                                  email: 'joe.doe@example.com', password: 'test' } }
29       expect(User.find_by(email: 'joe.doe@example.com')).to be_nil
30   end
31
32 end
33
34
```

EXAMPLE OF AN INTERACTOR

+ a/i/signup_user.rb

```
1  class SignupUser
2    include Interactor
3
4    def call
5      if user.valid?
6        user.save
7        NewUserJob.send_email(user.id)
8        AdminJob.send_new_user_notification(user.email)
9      else
10       context.fail!(message: user.errors.full_messages)
11     end
12   end
13
14   private
15
16   def user
17     context.user
18   end
19 end
20
```

BENEFITS

- Thin models and controllers
- Single responsibility principle
- Easy to test in isolation

```ruby
it 'sends an email to the new user' do
  user = double(User, valid?: true, id: '1', save: true, email: 'jane.doe@example.com')
  expect(NewUserJob).to receive(:send_email)
  SignupUser.call(user: user)
end
```

```ruby
describe 'POST create' do

  it 'is successful' do
    allow(SignupUser).to receive(:call).and_return(double(success?: true))
    post :create, {  user: { first_name: 'Joe', last_name: 'Doe',
                             email: 'joe.doe@example.com',
                             password: 'test' } }
    expect(response).to redirect_to(dashboard_path)
  end

  it 'is unsuccessful' do
    allow(SignupUser).to receive(:call).and_return(double(success?: false))
    post :create, {  user: { first_name: 'Joe', last_name: '',
                             email: 'joe.doe@example.com', password: 'test' } }
    expect(response).to have_http_status(:ok)
  end
end
```

ORGANIZERS

- An organizer is an important variation on the basic interactor. Its single purpose is to run other interactors.

a/o/signup_user.rb

```ruby
class SignupUser
  include Interactor::Organizer

  organize SaveUser, SendEmailToUser, SendNotification
end
```

WHAT'S WRONG WITH INTERACTOR GEM?

- The call method can often become a mess
- Class names are verbs
- The hash that is passed between interactors in an organizer is just global variable

IS THERE A BETTER ALTERNATIVE?

- DDD Domain-driven design
- Trailblazer's Operation http://trailblazer.to/gems/operation/
- Wisper gem https://github.com/krisleech/wisper

RESOURCES

- https://github.com/collectiveidea/interactor
- https://www.sitepoint.com/ddd-for-rails-developers-part-1-layered-architecture/
- http://eng.joingrouper.com/blog/2014/03/03/rails-the-missing-parts-interactors
- http://collectiveidea.com/blog/archives/2012/06/28/wheres-your-business-logic/
- http://www.rubyplus.net/2014/06/why-using-interactor-gem-is-very-bad.html