

Efficient Rails Test-Driven Development — Week 4

Wolfram Arnold
www.rubyfocus.biz

In collaboration with:
Sarah Allen, BlazingCloud.net
marakana.com

Controllers



Controllers

Controllers are pass-through entities

Mostly boilerplate—biz logic belongs in the model

Controllers are “dumb” or “skinny”

They follow a run-of-the mill pattern:

the Controller Formula

Controller RESTful Actions

Display methods (“Read”)

GET: index, show, new, edit

Update method

PUT

Create method

POST

Delete method

DELETE

REST?

Representational State Transfer

All resource-based applications & API's need to
do similar things, namely:

create, read, update, delete

It's a convention:

no configuration, no ceremony

superior to CORBA, SOAP, etc.

RESTful resources in Rails

`map.resources :people` (in `config/routes.rb`)

`people_path, people_url` “named route methods”

GET `/people` → “index” action

POST `/people` → “create” action

`new_person_path, new_person_url`

GET `/people/new` → “new” action

`edit_person_path, edit_person_url`

GET `/people/:id/edit` → “edit” action with ID

`person_path, person_url`

GET `/people/:id` → “show” action with ID

PUT `/people/:id` → “update” action with ID

DELETE `/people/:id` → “destroy” action with ID

Reads Test Pattern

Make request (with id of record if a single record)

Check Rendering

- correct template

- redirect

- status code

- content type (HTML, JSON, XML,...)

Verify Variable Assignments

- required by view

Read Formula

Find data, based on parameters

Assign variables

Render

How much test is too much?

Test anything where the code deviates from defaults, e.g. redirect vs. straight up render

These tests are not strictly necessary:

`response.should be_success`

`response.should render_template('new')`

Test anything required for the application to proceed without error

Specifically variable assignments

Do test error handling code!

form_for's automagic powers

```
form_for @person do |f| ... end
```

When @person is new

- <form action="people" method="post">
- PeopleController#create
- uses people_path method to generate URL

When @person exists already

- <form action="people" method="put">
- PeopleController#update
- uses person_path method to generate URL

Create/Update Test Pattern

Make request with form fields to be created/upd'd

Verify Variable Assignments

Verify Check Success

Rendering

Verify Failure/Error Case

Rendering

Variables

Verify HTTP Verb protection

Create/Update Formula

Update: Find record from parameters

Create: Instantiate new model object

Assign form fields parameters to model object

This should be a single line

It is a pattern, the “Controller Formula”

Save

Handle success—typically a redirect

Handle failure—typically a render

Destroy Test Pattern

Make request with id of record to be destroyed

Rendering

Typically no need to check for success or failure

Invalid item referenced → Same behavior as read

Database deletion failed → System not user error

Destroy Formula

Find record from parameters

Destroy

Redirect

How much is enough?

Notice: No view testing so far.

Emphasize behavior over display.

Check that the application handles *errors* correctly

Test views only for things that could go wrong badly

- incorrect form URL

- incorrect names on complicated forms, because they impact parameter representation

View Testing

RSpec controllers do *not* render views (by default)

Test form urls, any logic and input names

Understand CSS selector syntax

View test requires set up of variables

another reason why there should only be very few variables between controller and view

A note on RJS

RJS lets you render a JS response using “RJS”

Built on Prototype JS framework

Hard to test

Largely superseded by

- RSpec tested controllers responding in JSON

- JS tests mocking the server response

- jQuery's Ajax library very helpful

RSpec Scaffold & Mocks

RSpec Scaffold mocks models

Their philosophy is outside-in development

I'm not a fan of this because:

- It causes gaps in coverage

- Mocks need to be maintained

- Outside-in remains unproven beyond textbook examples

- When taken to the extreme, it can become un-Agile by over-specifying the application

Nested Attributes



One Form—Multiple Models

DB schema should not limit view/form layout

View/form layout should not impose DB schema

A single form to manipulate associated rows of multiple tables.

→ Nested Attributes

View & Model Collaboration

Model:

`accepts_nested_attributes_for`

View:

`form_for`

`fields_for`

Controller is unaware! Strictly pass-through

`Person.new(params[:person])`

accepts_nested_attributes_for

```
class Person < ActiveRecord::Base
  has_many :addresses
  accepts_nested_attributes_for :addresses
end
```

```
Person.create(:first_name => "Joe",
              :last_name  => "Smith",
              :addresses_attributes => [ {
                :street => "123 Main,
                :city   => "San Francisco",
                :zip     => "94103",
                :state   => "CA" } ]
```

Nested Attributes New & Old

A new record is *recognized by absence* of an id:

```
:address_attributes => [ { :city => "SF", :zip =>
  "12345", :street => "123 Main", :state => "CA" } ]
```

An existing record is *recognized by presence* of an id:

```
:address_attributes => [ { :id => "5",
  :city => "SF", :zip => "12345",
  :street => "123 Main", :state => "CA" } ]
```


Nested Attr's Singular & Plural

has_many :addresses

:addresses_attributes => [{ *record1* },
{ *record2* }]

has_one :address_of_record

:address_attributes => { *record* }

Nested Attributes Parameters

`:allow_destroy => true`

removes an existing record with `_destroy => '1'`

default is false

`:reject_if => :method or Proc.new { |attrs| ... }`

suppresses creation or updating when false

can pass `:all_blank` to skip records with all blank attr's

`:limit => 5`

limits numbers of records creatable from nested attr's

`:update_only`

prevents multiple records for 1-on-1 associations

Textbook Outside-In BDD

Yes...

Helps to lay out model requirements via mocks

But...

Mocks get out of sync

A lot more work for not very much improved coverage

Best place is exploratory testing

Does nothing to encourage fat model/skinny
controller paradigm

Knowledge of advanced model features still key

Value-Driven Outside-In BDD

Write controller tests first

Make them pending while fleshing out model

Implement model logic with

- nested_scopes

- associations, with rich options & proxy methods

- accepts_nested_attributes_for

Resume controller test

Add views

Exploratory Cucumber/Selenium test

- to avoid test gap between view & controller

Homework



Homework

As a person, I can update my name along with an addresses on the same form.

Extra credit:

Using JavaScript, create an “add one” link on the new form that replicates the address sub-form, to permit entering more than one address.
Same for a delete link.

Reuse the above mechanism for the edit form.

Recommended Reading

<http://railscasts.com/episodes/196-nested-model-form-part-1>

<http://asciicasts.com/episodes/196-nested-model-form-part-1>

<http://railscasts.com/episodes/197-nested-model-form-part-2>

<http://asciicasts.com/episodes/197-nested-model-form-part-2>

http://docs.jquery.com/Main_Page

<http://activescaffold.com/>

Nested Attributes API Docs: <http://bit.ly/7cnt0>

Form for (with nest'd attr's) API Docs:

<http://bit.ly/9DAscq>

My presentation on nested attributes: <http://blip.tv/file/3957941>

RSpec book chapters 1.5, 10, 11 (on BDD)

RSpec book chapter 20 (outside-in development)

Flickr Photo Attribute Credit

Matroshka nested dolls

<http://www.flickr.com/photos/shereen84/2511071028/sizes//>

Notebook with pencil

<http://www.flickr.com/photos/tomsaint/2987926396/sizes//>

Control Panel

<http://www.flickr.com/photos/900hp/3961052527/sizes//>