Episode 264

Guard

In episode 257[watch[1], read[2]] we wrote an application that was built using test-driven development with RSpec's Request Specs. We can check that the application's tests pass at any time by running `rake spec`.

```
$ rake spec
(in /Users/eifion/rails/todo)
/Users/eifion/.rvm/rubies/ruby-1.9.2-p0/bin/ruby -S bundle exec
rspec ./spec/models/task_spec.rb ./spec/requests/tasks_spec.rb
...

Finished in 0.92211 seconds
3 examples, 0 failures
```

It can get a little tedious to have to run that `rake` command every time we make a code change. If we have a large test suite then we might have to wait a long time for all of the tests to run, even if we've only changed one or two files. It would be useful if there was a way to automate this.

There are a number of tools to solve this problem but in this episode we're going to take a look at Guard[3]. This provides a way to listen to file modifications and then perform an action when a file is modified. There are a large number of extensions[4] available that enable Guard for different environments including one for RSpec[5] that will do just what we want: listening for file changes and then running the specs when there's a change. If you're using Test::Unit instead of RSpec there's an extension[6] that will work with that, too. There are also extensions for Cucumber, minitest and many other environments.

Installing

Let's try out Guard with guard-rspec in our application and see how we get on. We'll start by adding some gems to the `test` group in the `Gemfile`. As we're

---

[1] http://railscasts.com/episodes/257-request-specs-and-capybara

[2] http://asciicasts.com/episodes/257-request-specs-and-capybara

[3] https://github.com/guard/guard

[4] https://github.com/guard/guard/wiki/List-of-available-Guards

[5] https://github.com/guard/guard-rspec

[6] https://github.com/guard/guard-test

developing under OS X we need to install the rb-fsevent[7] gem as a prerequisite for Guard so we'll add that with a check that the current platform is OS X. We'll also add a reference to the `guard-rspec` gem. There's no need to add `guard` itself as it will be installed as a dependency on `guard-rspec`.

/Gemfile

```
source 'http://rubygems.org'

gem 'rails', '3.0.5'
gem 'sqlite3'
gem 'nifty-generators'
gem 'jquery-rails'

group :development, :test do
  gem 'rspec-rails'
  gem 'capybara', :git => 'git://github.com/jnicklas/capybara.git'
  gem 'launchy'
  gem 'database_cleaner'
  gem 'rb-fsevent', :require => false if RUBY_PLATFORM =~ ↵
    /darwin/i
  gem 'guard-rspec'
end
```

We can now run `bundle` to install everything and, once that's done, run `guard init rspec` to set up Guard.

```
$ guard init rspec
Writing new Guardfile to /Users/eifion/rails/todo/Guardfile
rspec guard added to Guardfile, feel free to edit it
```

If you have problem running this command you can try prefixing it with `bundle exec`. This command will generate a `Guardfile` which we can leave alone for now, although we'll come back and take a look at it later.

Guard also has support for Growl[8] notifications and these can be enabled by installing the `growl` gem, though we won't do that there.

---

[7] http://rubygems.org/gems/rb-fsevent

[8] http://growl.info/

We can now run the `guard` command to start up the Guard server.

```
$ guard
Guard is now watching at '/Users/eifion/Desktop/Dropbox/rails/
apps_for_asciicasts/ep264/todo'
Guard::RSpec is running, with RSpec 2!
Running all specs
...

Finished in 1.02 seconds
3 examples, 0 failures
```

If we make a breaking change to our code, for example removing a validation from the `Task` model, Guard will run immediately and we'll see the broken test.

/app/models/task.rb

```
class Task < ActiveRecord::Base
  attr_accessible :name
  #validates_presence_of :name
end
```

```
Running: spec/models/task_spec.rb
F

Failures:

  1) Task validates name
     Failure/Error: Task.new.should have(1).error_on(:name)
       expected 1 error on :name, got 0
     # ./spec/models/task_spec.rb:5:in `block (2 levels) in <top
(required)>'

Finished in 0.04825 seconds
1 example, 1 failure
```

As soon as we reinstate the validator the specs run again and pass.

Customizing Behaviour

If we want to we can customize Guard's behaviour. By default it won't run when we make a change to a view file. We can add this behaviour by modifying the `Guardfile` that was generated by the `guard init` command.

The default Guardfile is shown below.

/Guardfile

```
# A sample Guardfile
# More info at https://github.com/guard/guard#readme

guard 'rspec', :version => 2 do
  watch(%r{^spec/.+_spec\.rb})
  watch(%r{^lib/(.+)\.rb})     { |m| "spec/lib/#{m[1]}_spec.rb" }
  watch('spec/spec_helper.rb') { "spec" }

  # Rails example
  watch('spec/spec_helper.rb')                      { "spec" }
  watch('config/routes.rb')                         { "spec/routing" }
  watch('app/controllers/application_controller.rb')
     { "spec/controllers" }
  watch(%r{^spec/.+_spec\.rb})
  watch(%r{^app/(.+)\.rb})
     { |m| "spec/#{m[1]}_spec.rb" }
  watch(%r{^lib/(.+)\.rb})
     { |m| "spec/lib/#{m[1]}_spec.rb" }
  watch(%r{^app/controllers/(.+)_(controller)\.rb})
     { |m| ["spec/routing/#{m[1]}_routing_spec.rb",
       "spec/#{m[2]}s/#{m[1]}_#{m[2]}_spec.rb",
      "spec/acceptance/#{m[1]}_spec.rb"]
     }
end
```

Each line in the file is a call to a `watch` method with a block which contains the specs that will be run when a matching file changes. For example if the routes file is changed then it will look for a routing spec and change that. Some cases are more complex and take a regular expression. For example this line matches all of the Ruby files under the `/lib` directory. The modified file's is captured by the part of

the regular expression in parentheses and this is passed to the block so that the matching spec is run.

```
watch(%r{^lib/(.+)\.rb})    { |m| "spec/lib/#{m[1]}_spec.rb" }
```

All of the files matched in the default `Guardfile` are Ruby files. We want to run specs when a view file changes so we'll add a new `watch` command to match these.

`/Guardfile`

```
watch(%r{^app/views/(.+)/})
  { |m| "spec/requests/#{m[1]}_spec.rb" }
```

In this new rule we match everything under the `/app/views` directory and capture everything after `app/views/` until the last slash, so for the file `/app/views/tasks/index.html.erb` this will capture `tasks`. We can then pass this into the block to run the appropriate request specs.

We can test this out by changing one of the view files so that it breaks a test. If we remove the submit button from the tasks `index` view file Guard will run the request specs for tasks and we'll see a failing spec.

```
Running: spec/requests/tasks_spec.rb
.F

Failures:

  1) Tasks creates a task with validation error
     Failure/Error: click_button "Add"
     Capybara::ElementNotFound:
       no button with value or id or text 'Add' found
     # ./spec/requests/tasks_spec.rb:13:in `block (2 levels) in
<top (required)>'

Finished in 0.82203 seconds
2 examples, 1 failure
```

When we add it back the specs all pass again.

Using Guard To Automatically Refresh The Browser

Guard is a great way to run tests automatically when certain files in an application change but there are some other nice things we can do with it. We can for example automatically compile CoffeeScript and SASS files when they change, automatically run the `bundle` command when the `Gemfile` changes or restart a Passenger or Pow server when a config or initializer file changes. The one we'll take a look at is guard-livereload which is a gem that will automatically reload a browser when certain files change.

To use it we add a reference to it in the `Gemfile` and run `bundle` again.

`/Gemfile`

```
source 'http://rubygems.org'

gem 'rails', '3.0.5'
gem 'sqlite3'
gem 'nifty-generators'
gem 'jquery-rails'

group :development, :test do
  gem 'rspec-rails'
  gem 'capybara', :git => 'git://github.com/jnicklas/capybara.git'
  gem 'launchy'
  gem 'database_cleaner'
  gem 'rb-fsevent', :require => false if RUBY_PLATFORM =~ /darwin/i
  gem 'guard-rspec'
  gem 'guard-livereload'
end
```

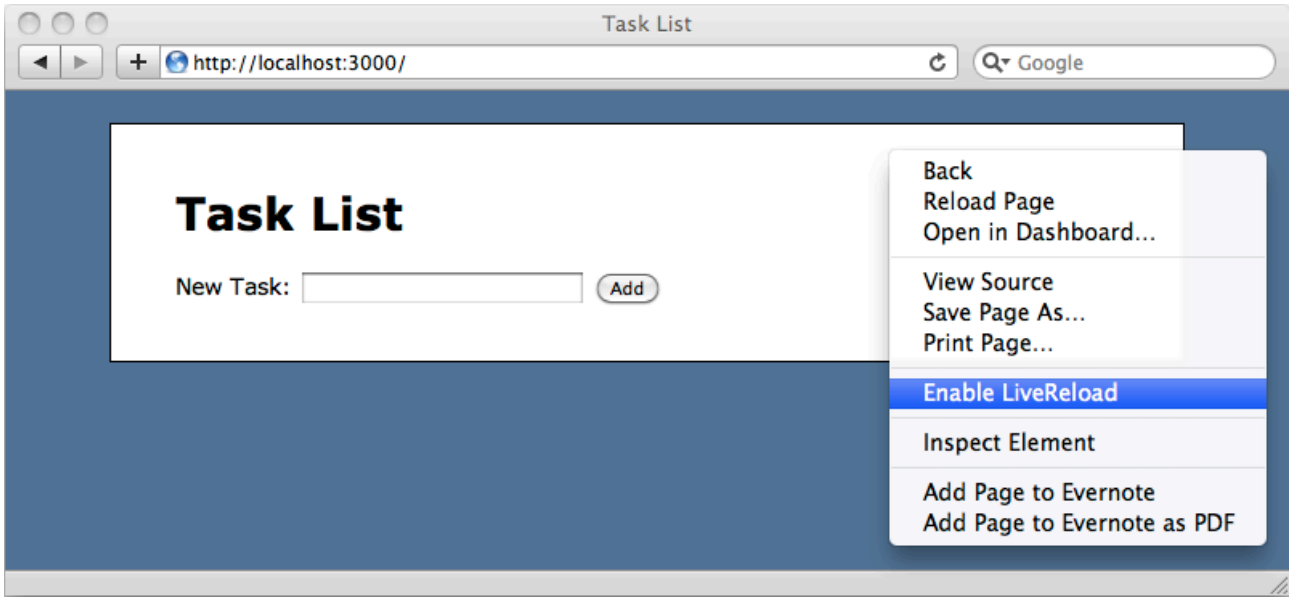
When the gem has installed we can need to run `guard init livereload` to modify the `Guardfile` and then start `guard` again.

Next we'll need to install a LiveReload extension for Chrome, Safari or Firefox. The LiveReload page[9] has links for installing this and as we're using Safari we'll install that one.

Once we've installed the extension we can enable it by going to our application's home page, right-clicking and selecting "Enable Live Reload".

---

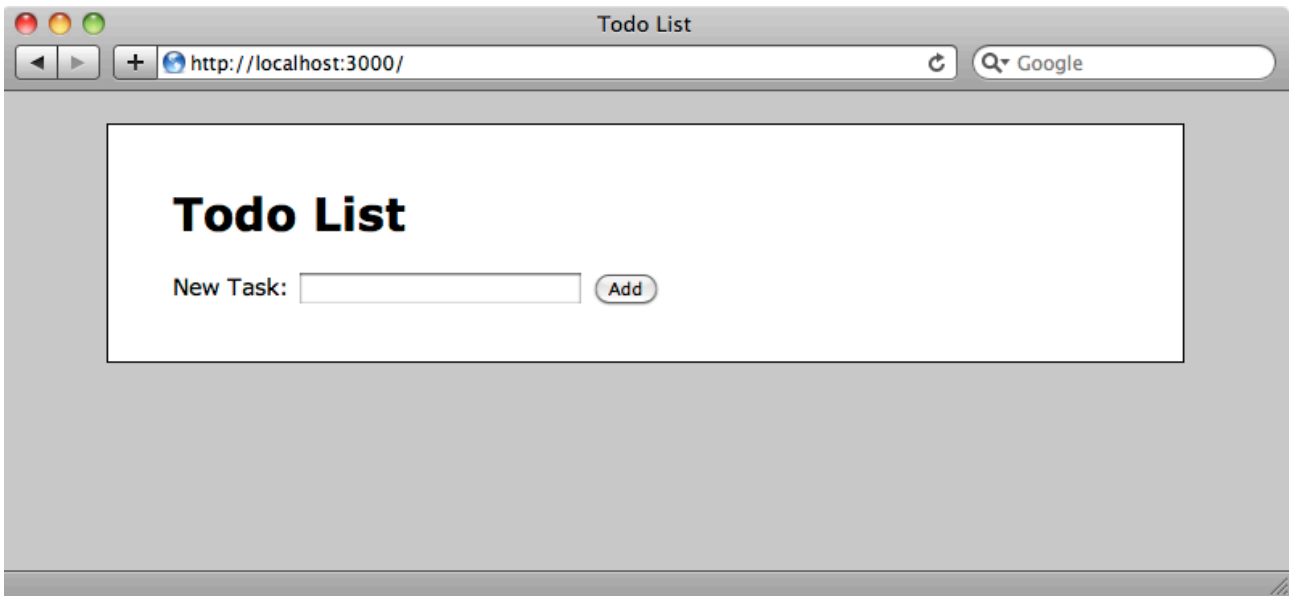
[9] https://github.com/mockko/livereload



If we make a change to one of our application's files, say the stylesheet, the page is reloaded in the browser as soon as we save the file. This works with other files too. If we change the title of the page in an erb file that change is reflected in the browser when we save the file, too.

/app/views/tasks/index.html.erb

```
<% title "Todo List" %>
```



There's more of a delay when we change the erb file as the specs are rerun before LiveReload loads the page. This is because of the order of the items in the

`Guardfile`. It would be more useful if the browser was updated first as we could then check the new page as the specs are running. To do that we just need to swap the two `guard` blocks in the `Guardfile` over so that the `livereload` section runs first.

/Guardfile

```
# A sample Guardfile
# More info at https://github.com/guard/guard#readme

guard 'livereload' do
  watch(%r{app/.+\.(erb|haml)})
  watch(%r{app/helpers/.+\.rb})
  watch(%r{public/.+\.(css|js|html)})
  watch(%r{config/locales/.+\.yml})
end

guard 'rspec', :version => 2 do
  watch(%r{^spec/.+_spec\.rb})
  watch(%r{^lib/(.+)\.rb})
    { |m| "spec/lib/#{m[1]}_spec.rb" }
  watch('spec/spec_helper.rb')
    { "spec" }

  # Rails example
  watch('spec/spec_helper.rb')  { "spec" }
  watch('config/routes.rb')     { "spec/routing" }
  watch('app/controllers/application_controller.rb')
    { "spec/controllers" }
  watch(%r{^spec/.+_spec\.rb})
  watch(%r{^app/(.+)\.rb})
    { |m| "spec/#{m[1]}_spec.rb" }
  watch(%r{^lib/(.+)\.rb})
    { |m| "spec/lib/#{m[1]}_spec.rb" }
  watch(%r{^app/controllers/(.+)_(controller)\.rb})
    { |m| ["spec/routing/#{m[1]}_routing_spec.rb",
      "spec/#{m[2]}s/#{m[1]}_#{m[2]}_spec.rb",
      "spec/acceptance/#{m[1]}_spec.rb"] }
  watch(%r{^app/views/(.+)/})
    { |m| "spec/requests/#{m[1]}_spec.rb" }
end
```

Now if we make a change in a file the change is seen straight away.