VCR

A Gem used for caching HTTP requests during tests

Who am I?

- Mike Dalton
- Developer @ GrubHub
- Using Ruby for 7 years
- Frequent attendee of meetups

The problem

- Tests should be deterministic
- Result of an HTTP request might not be known
  - Change in data beyond your control
  - Network connectivity issues
- How do we have deterministic tests that involve 3rd party web services?

The solution

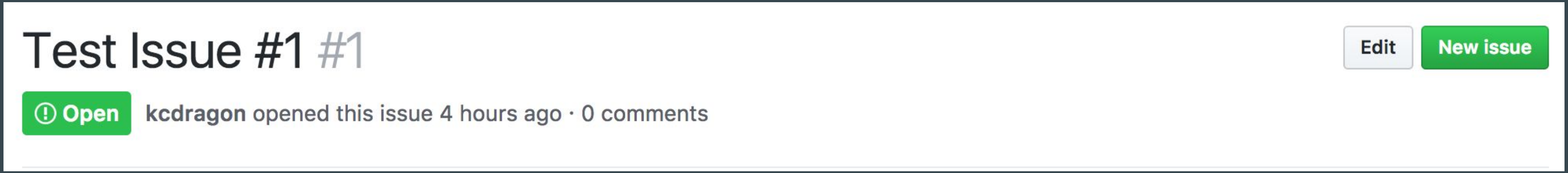
- VCR Gem
- https://github.com/vcr/vcr
- Created by Myron Marston (maintainer of RSpec)
- Around since 2010
- *Record your test suite's HTTP interactions and replay them during future test runs for fast, deterministic, accurate tests.*

Examples

First example

Query all issues from a GitHub repository

Create first GitHub Issue


Test Issue #1 #1

kcdragon opened this issue 4 hours ago · 0 comments

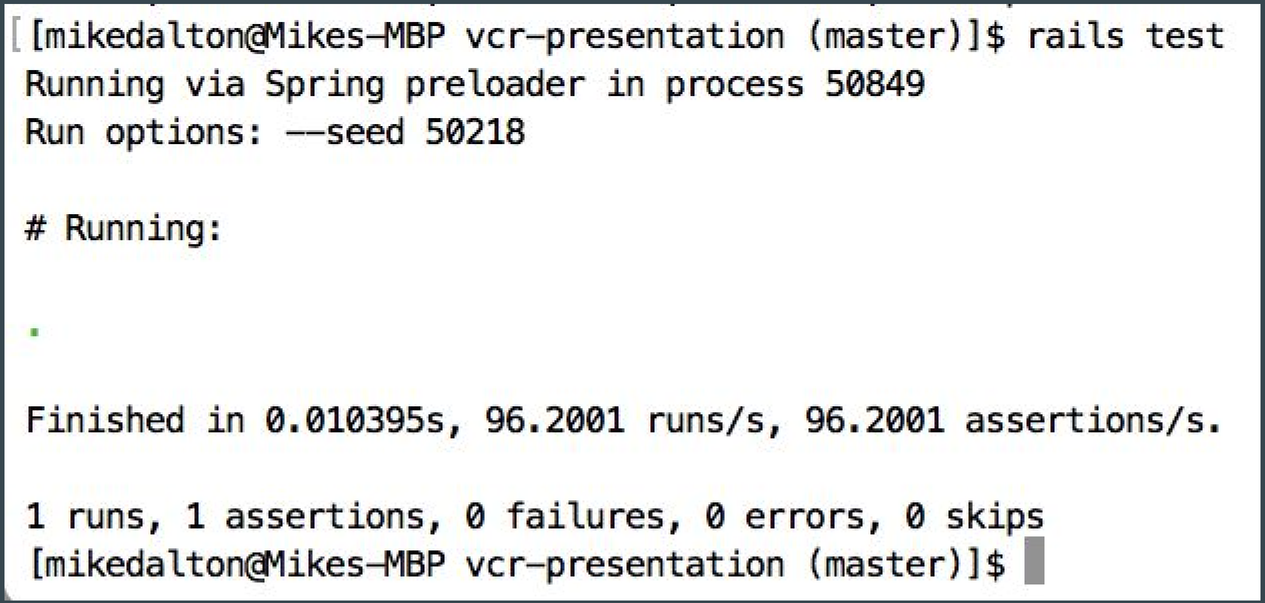
Test for Issue.all

```ruby
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_all_issues
    issues = Issue.all

    assert_equal 1, issues.count
  end
end
```

Implementation of Issue.all

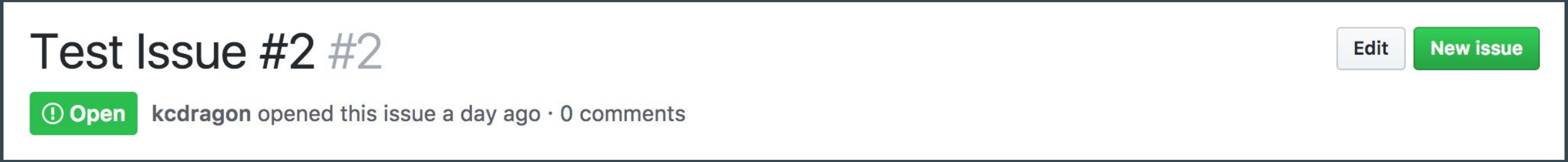
```ruby
class Issue
  include ActiveModel::Model

  REPOSITORY = 'https://api.github.com/repos/kcdragon/vcr-presentation'

  attr_accessor :title

  def self.all
    uri = URI.parse("#{REPOSITORY}/issues")
    response = Net::HTTP.get_response(uri)

    JSON.parse(response.body).map do |issue_data|
      Issue.new(
        title: issue_data['title']
      )
    end
  end
end
```

Result of running test



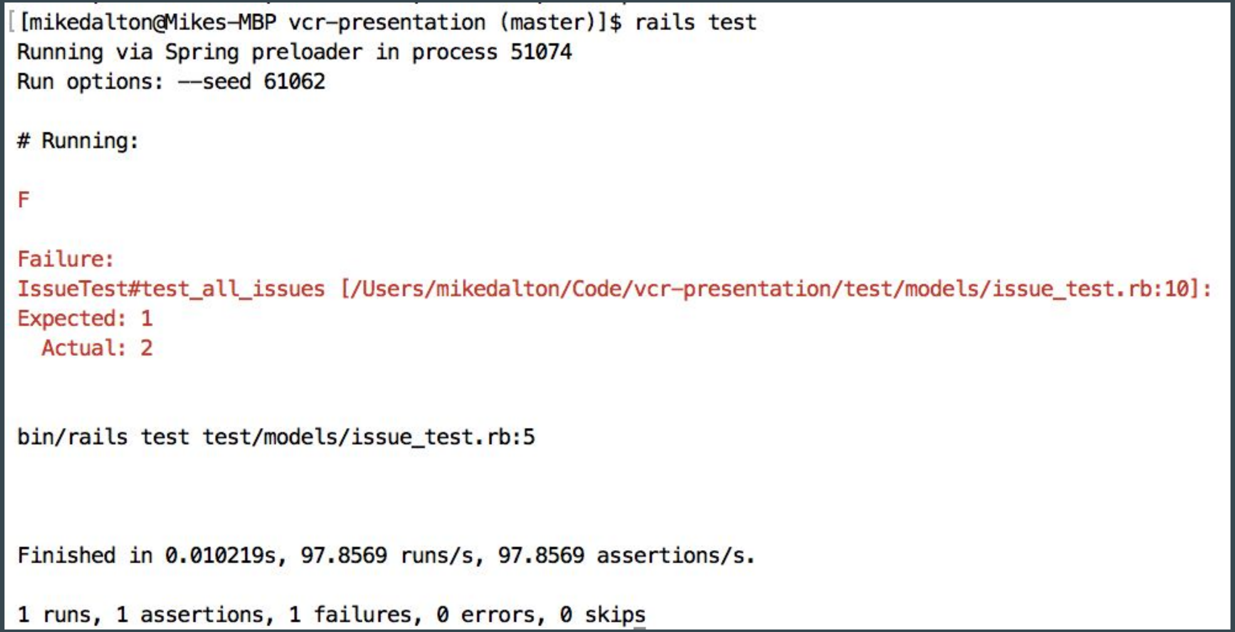
```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50849
Run options: --seed 50218

# Running:

.

Finished in 0.010395s, 96.2001 runs/s, 96.2001 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
[mikedalton@Mikes-MBP vcr-presentation (master)]$
```

Create second GitHub Issue


Test Issue #2 #2

kcdragon opened this issue a day ago · 0 comments

Result of running test



```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 51074
Run options: --seed 61062

# Running:

F

Failure:
IssueTest#test_all_issues [/Users/mikedalton/Code/vcr-presentation/test/models/issue_test.rb:10]:
Expected: 1
  Actual: 2


bin/rails test test/models/issue_test.rb:5



Finished in 0.010219s, 97.8569 runs/s, 97.8569 assertions/s.

1 runs, 1 assertions, 1 failures, 0 errors, 0 skips
```

VCR to the rescue!

```ruby
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_all_issues
    issues = VCR.use_cassette('issue/all') do
      Issue.all
    end

    assert_equal 2, issues.count
  end
end
```

```ruby
# test/test_helper.rb

VCR.configure do |config|
  config.cassette_library_dir = 'test/cassettes'
  config.hook_into :webmock
end
```

```ruby
# Gemfile

gem 'vcr', '3.0.3'
gem 'webmock', '3.0.1'
```

How does this work?

- First time test is run:
  - HTTP request is performed
  - VCR creates a YAML file (called a “cassette”) to store request and response
- Second time test is run:
  - VCR recognizes the same request is being made
  - VCR uses YAML file to return the response

Cassette file

- YAML format
- Contains both the HTTP request and response
- Single YAML file can contain multiple requests
- Each request must have a response
- Single YAML file can be used in multiple tests

Cassette for Issue.all request/response

```
---
http_interactions:
- request:
    method: get
    uri: https://api.github.com/repos/kcdragon/vcr-presentation/issues
    ...
  response:
    status:
      code: 200
      message: OK
    headers:
      ...
    body:
      encoding: ASCII-8BIT
      string: '[{...}]'
    http_version:
  recorded_at: Mon, 17 Apr 2017 19:08:29 GMT
recorded_with: VCR 3.0.3
```

Second example

- Create an issue via the GitHub API
- Check that issue has been created

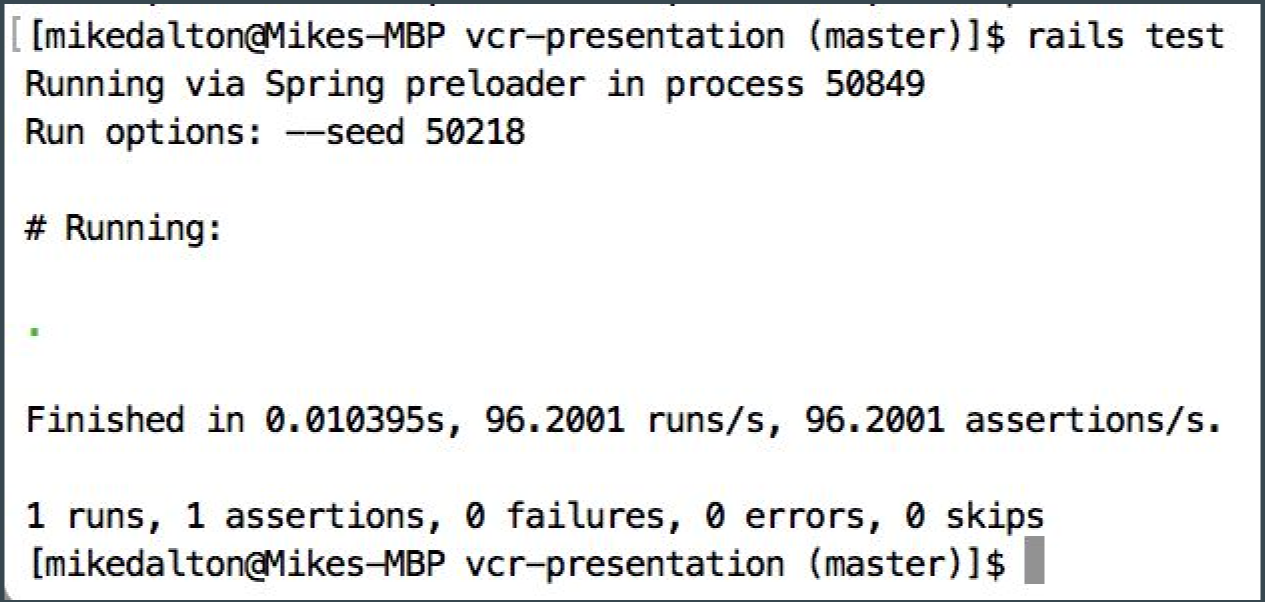
Test for Issue.create

```ruby
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_create_issue
    title = 'Issue created from API #1'
    issue = Issue.new(title: title)
    VCR.use_cassette('issue/create') do
      Issue.create(issue) # first HTTP request

      issues = Issue.all # second HTTP request
      issue = issues.first
      assert_equal title, issue.title
    end
  end
end
```

Implementation for Issue.create

```ruby
class Issue
  include ActiveModel::Model

  REPOSITORY = 'https://api.github.com/repos/kcdragon/vcr-presentation'

  attr_accessor :title

  def self.create(issue)
    uri = URI.parse("#{REPOSITORY}/issues")
    request = Net::HTTP::Post.new(uri)
    request.body = JSON.generate(title: issue.title)
    request.basic_auth("user", "token")

    Net::HTTP.start(uri.hostname, uri.port, use_ssl: true) do |http|
      http.request(request)
    end
  end
end
```

Result of running test

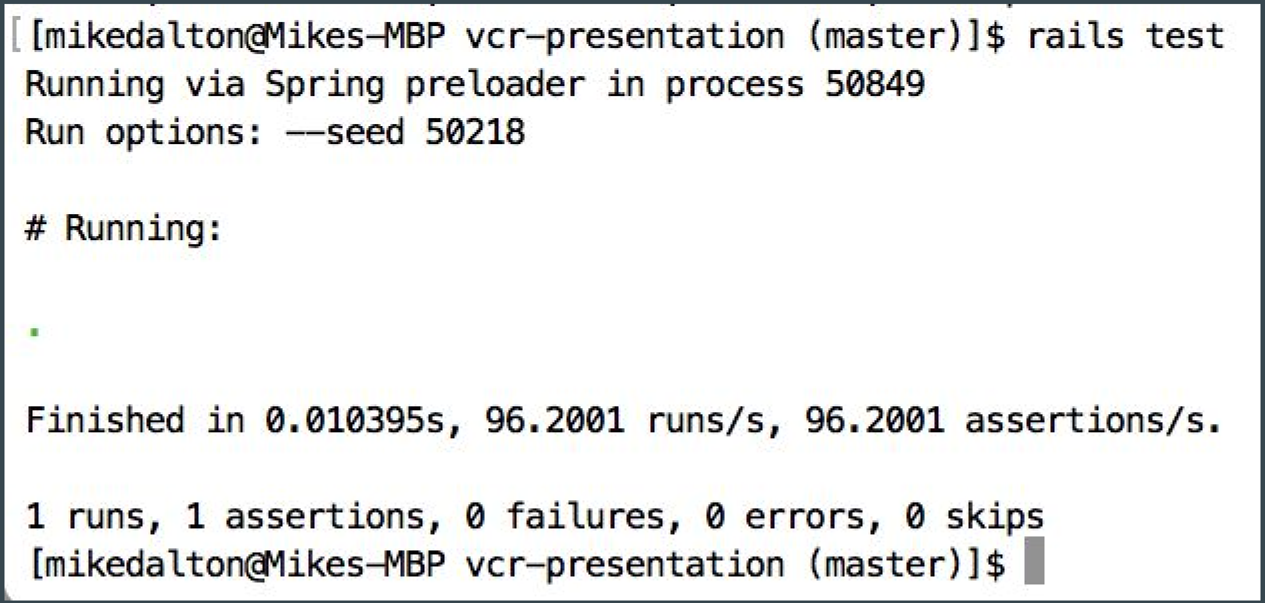


```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50849
Run options: --seed 50218

# Running:

.

Finished in 0.010395s, 96.2001 runs/s, 96.2001 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
[mikedalton@Mikes-MBP vcr-presentation (master)]$
```

“Accidentally” introduce a bug

```
class Issue
  # ...

  def self.create(issue)
    # ...
    request.body = JSON.generate(title: nil) # ⇐ Change `issue.title` to `nil`
    # ...
  end
end
```

Result of running test



```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50849
Run options: --seed 50218

# Running:

.

Finished in 0.010395s, 96.2001 runs/s, 96.2001 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
[mikedalton@Mikes-MBP vcr-presentation (master)]$
```

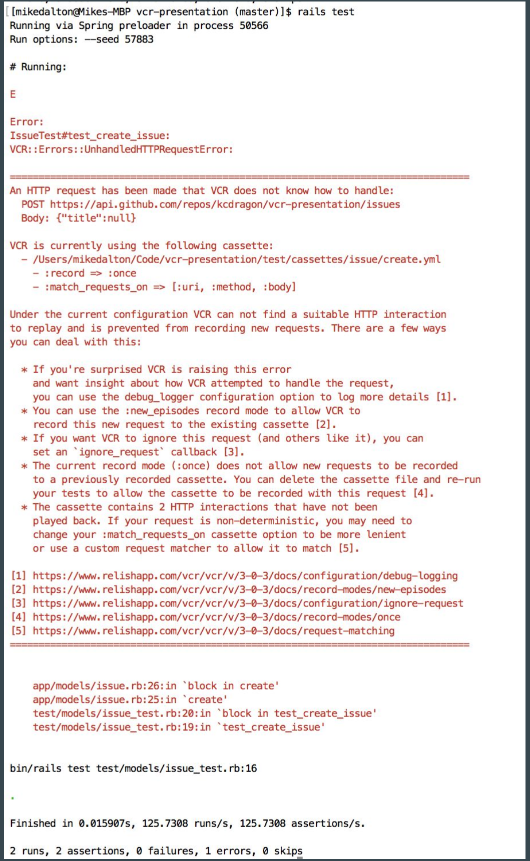
We changed the application code but the tests still pass?

- VCR default matching
    - URI
    - HTTP Method (GET, POST, etc)
- Need to tell VCR how to match

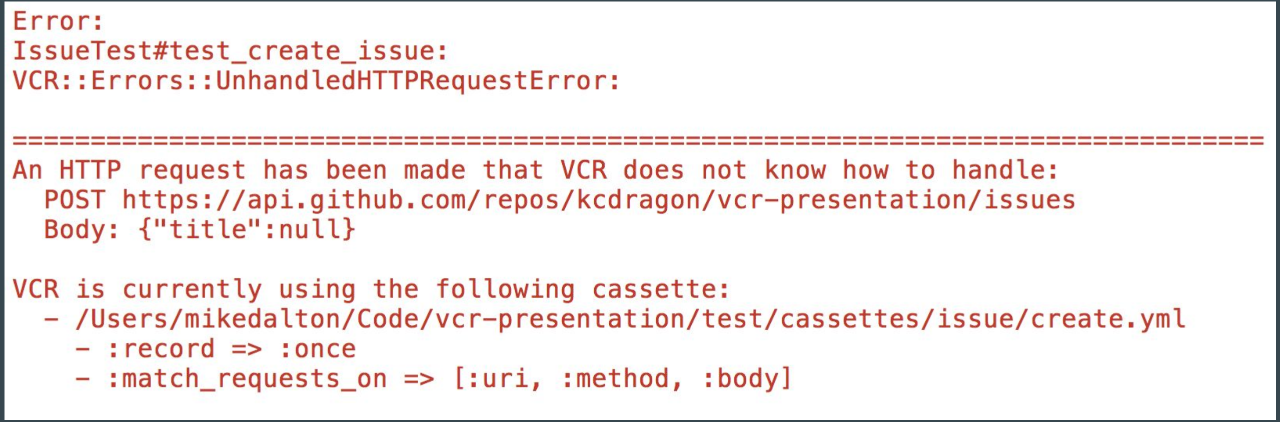
Test for Issue.create

```ruby
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_create_issue
    # ...
    VCR.use_cassette('issue/create', match_requests_on: %i(uri method body)) do
      # ...
    end
  end
end
```

Result of running test



```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50566
Run options: --seed 57883

# Running:

E

Error:
IssueTest#test_create_issue:
VCR::Errors::UnhandledHTTPRequestError:

================================================================================
An HTTP request has been made that VCR does not know how to handle:
  POST https://api.github.com/repos/kcdragon/vcr-presentation/issues
  Body: {"title":null}

VCR is currently using the following cassette:
  - /Users/mikedalton/Code/vcr-presentation/test/cassettes/issue/create.yml
    - :record => :once
    - :match_requests_on => [:uri, :method, :body]

Under the current configuration VCR can not find a suitable HTTP interaction
to replay and is prevented from recording new requests. There are a few ways
you can deal with this:

  * If you're surprised VCR is raising this error
    and want insight about how VCR attempted to handle the request,
    you can use the debug_logger configuration option to log more details [1].
  * You can use the :new_episodes record mode to allow VCR to
    record this new request to the existing cassette [2].
  * If you want VCR to ignore this request (and others like it), you can
    set an `ignore_request` callback [3].
  * The current record mode (:once) does not allow new requests to be recorded
    to a previously recorded cassette. You can delete the cassette file and re-run
    your tests to allow the cassette to be recorded with this request [4].
  * The cassette contains 2 HTTP interactions that have not been
    played back. If your request is non-deterministic, you may need to
    change your :match_requests_on cassette option to be more lenient
    or use a custom request matcher to allow it to match [5].

[1] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/configuration/debug-logging
[2] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/record-modes/new-episodes
[3] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/configuration/ignore-request
[4] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/record-modes/once
[5] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/request-matching
================================================================================


    app/models/issue.rb:26:in `block in create'
    app/models/issue.rb:25:in `create'
    test/models/issue_test.rb:20:in `block in test_create_issue'
    test/models/issue_test.rb:19:in `test_create_issue'


bin/rails test test/models/issue_test.rb:16

.

Finished in 0.015907s, 125.7308 runs/s, 125.7308 assertions/s.

2 runs, 2 assertions, 0 failures, 1 errors, 0 skips
```

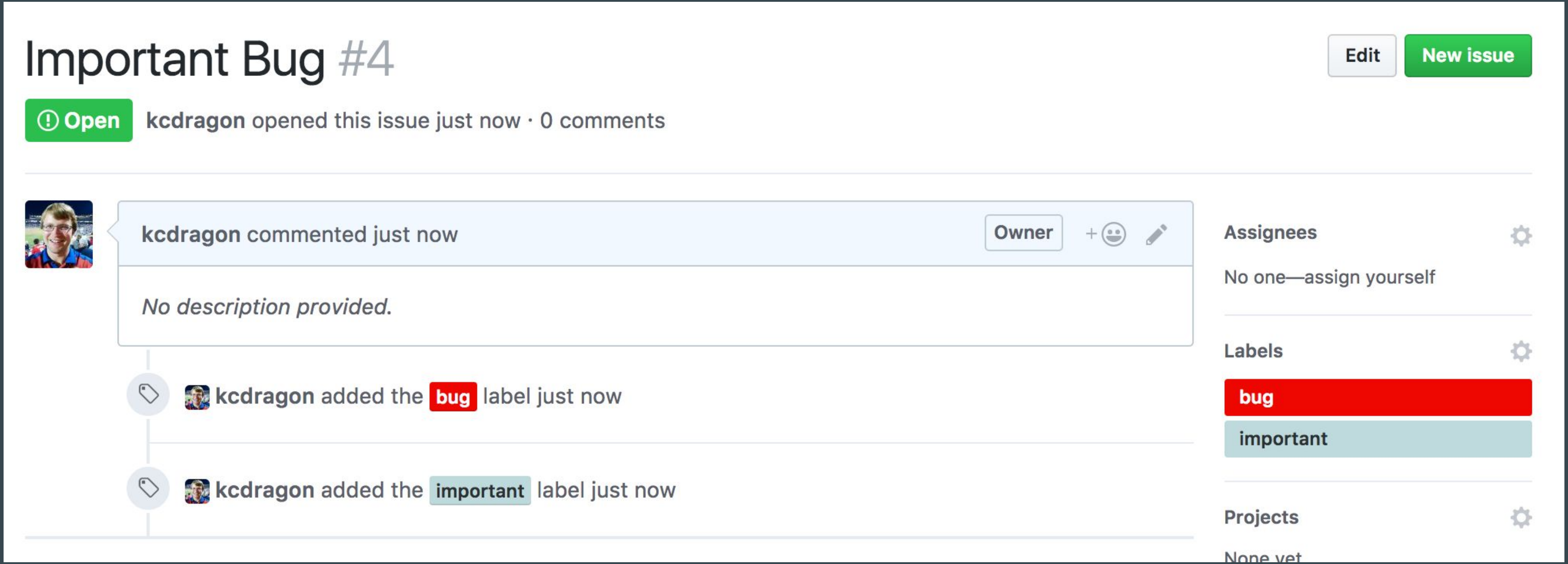


```
Error:
IssueTest#test_create_issue:
VCR::Errors::UnhandledHTTPRequestError:

================================================================================
An HTTP request has been made that VCR does not know how to handle:
  POST https://api.github.com/repos/kcdragon/vcr-presentation/issues
  Body: {"title":null}

VCR is currently using the following cassette:
  - /Users/mikedalton/Code/vcr-presentation/test/cassettes/issue/create.yml
    - :record => :once
    - :match_requests_on => [:uri, :method, :body]
```

Third example

Query GitHub for important bugs

Create an important bug issue in GitHub



Important Bug #4

Edit New issue

Open kcdragon opened this issue just now · 0 comments

kcdragon commented just now Owner

No description provided.

kcdragon added the bug label just now

kcdragon added the important label just now

Assignees

No one—assign yourself

Labels

bug

important

Projects

None yet

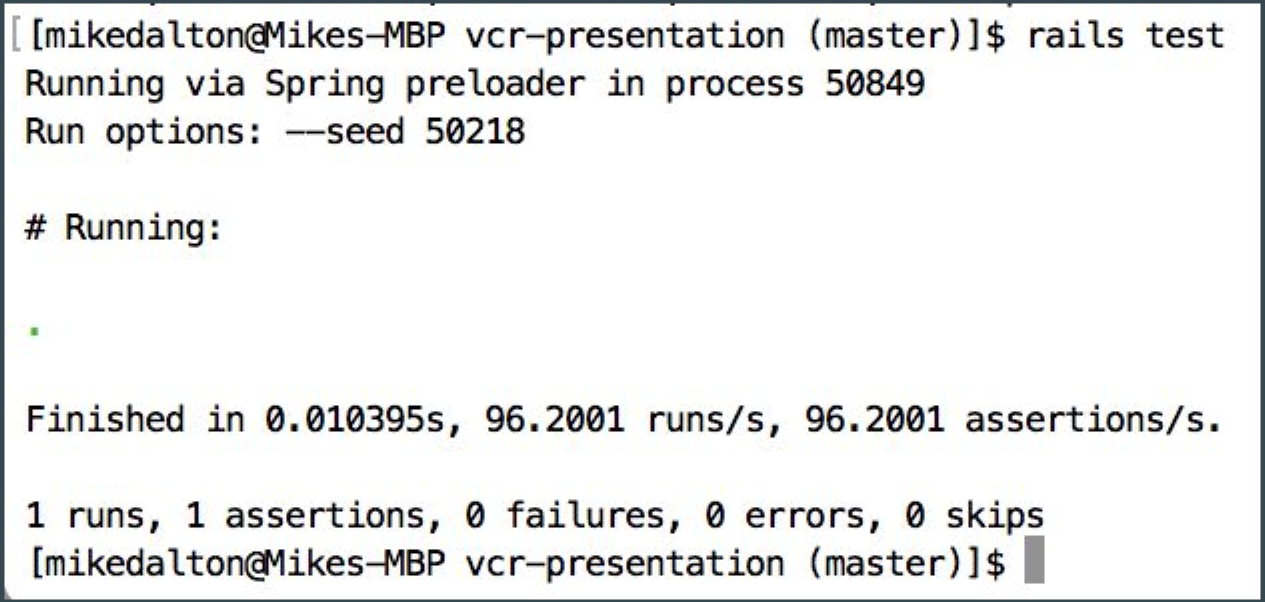
Test for Issue.important_bugs

```ruby
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_important_bug_issues
    issues = VCR.use_cassette('issue/important_bugs') do
      Issue.important_bugs
    end

    assert_equal 1, issues.count
  end
end
```

Implementation for Issue.important_bugs

```ruby
class Issue
  # ...

  def self.important_bugs
    uri = URI.parse("#{REPOSITORY}/issues?labels=bug,important")
    response = Net::HTTP.get_response(uri)

    JSON.parse(response.body).map do |issue_data|
      Issue.new(
        title: issue_data['title']
      )
    end
  end
end
```

Result of running test

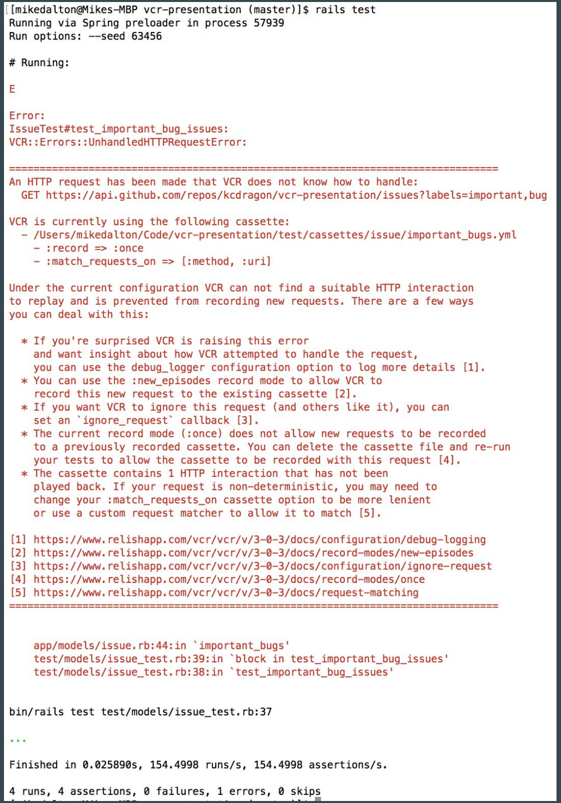


```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50849
Run options: --seed 50218

# Running:


.


Finished in 0.010395s, 96.2001 runs/s, 96.2001 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
[mikedalton@Mikes-MBP vcr-presentation (master)]$
```

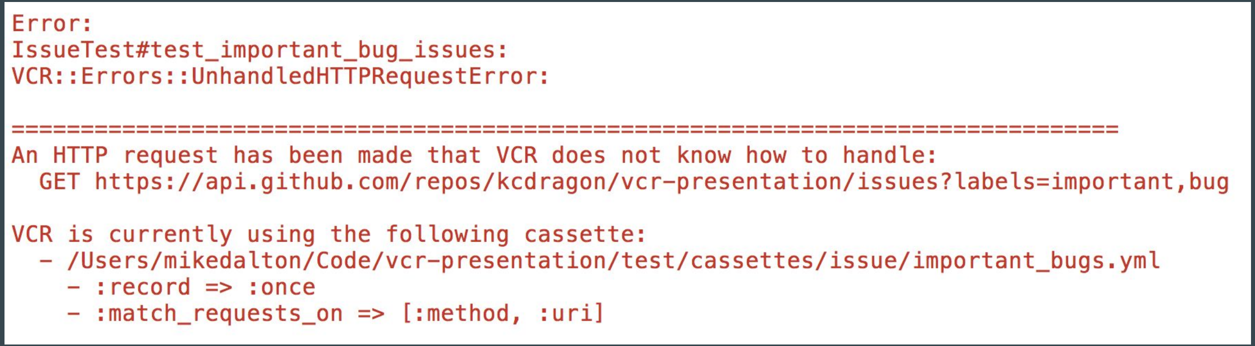
“Refactor” some code

```
class Issue
  # ...

  def self.important_bugs
    uri = URI.parse("#{REPOSITORY}/issues?labels=important,bug") # ⇐ Change “bug,important” to “important,bug”
    # ...
  end
end
```

Uh-oh...



```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 57939
Run options: --seed 63456

# Running:

E

Error:
IssueTest#test_important_bug_issues:
VCR::Errors::UnhandledHTTPRequestError:

================================================================================
An HTTP request has been made that VCR does not know how to handle:
  GET https://api.github.com/repos/kcdragon/vcr-presentation/issues?labels=important,bug

VCR is currently using the following cassette:
  - /Users/mikedalton/Code/vcr-presentation/test/cassettes/issue/important_bugs.yml
    - :record => :once
    - :match_requests_on => [:method, :uri]

Under the current configuration VCR can not find a suitable HTTP interaction
to replay and is prevented from recording new requests. There are a few ways
you can deal with this:

  * If you're surprised VCR is raising this error
    and want insight about how VCR attempted to handle the request,
    you can use the debug_logger configuration option to log more details [1].
  * You can use the :new_episodes record mode to allow VCR to
    record this new request to the existing cassette [2].
  * If you want VCR to ignore this request (and others like it), you can
    set an `ignore_request` callback [3].
  * The current record mode (:once) does not allow new requests to be recorded
    to a previously recorded cassette. You can delete the cassette file and re-run
    your tests to allow the cassette to be recorded with this request [4].
  * The cassette contains 1 HTTP interaction that has not been
    played back. If your request is non-deterministic, you may need to
    change your :match_requests_on cassette option to be more lenient
    or use a custom request matcher to allow it to match [5].

[1] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/configuration/debug-logging
[2] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/record-modes/new-episodes
[3] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/configuration/ignore-request
[4] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/record-modes/once
[5] https://www.relishapp.com/vcr/vcr/v/3-0-3/docs/request-matching
================================================================================


    app/models/issue.rb:44:in `important_bugs'
    test/models/issue_test.rb:39:in `block in test_important_bug_issues'
    test/models/issue_test.rb:38:in `test_important_bug_issues'


bin/rails test test/models/issue_test.rb:37

...

Finished in 0.025890s, 154.4998 runs/s, 154.4998 assertions/s.

4 runs, 4 assertions, 0 failures, 1 errors, 0 skips
```



```
Error:
IssueTest#test_important_bug_issues:
VCR::Errors::UnhandledHTTPRequestError:

================================================================================
An HTTP request has been made that VCR does not know how to handle:
  GET https://api.github.com/repos/kcdragon/vcr-presentation/issues?labels=important,bug

VCR is currently using the following cassette:
  - /Users/mikedalton/Code/vcr-presentation/test/cassettes/issue/important_bugs.yml
    - :record => :once
    - :match_requests_on => [:method, :uri]
```

Two solutions

- Delete the existing cassette and generate a new cassette
  - May require changing the test
- Use a “custom matcher” to accept any ordering of labels
  - There is no built-in matcher for our specific need

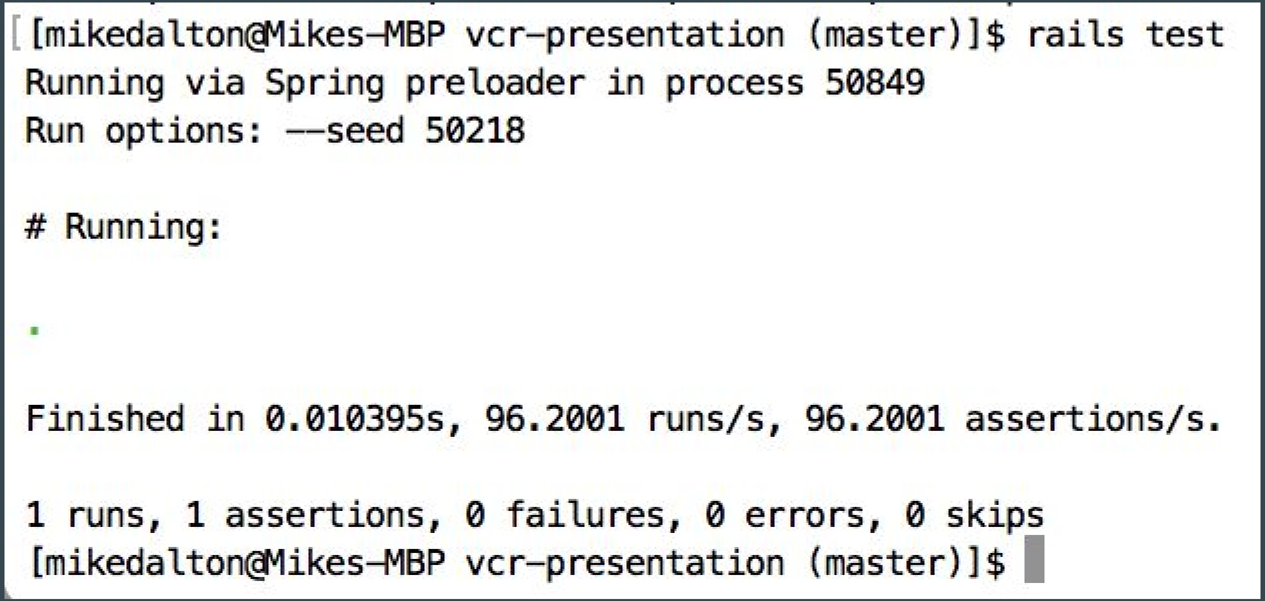
Custom matcher for “labels=bug,important” in query string

```
VCR.configure do |config|
 # ...

 config.register_request_matcher :label_in_query_string do |request_1, request_2|
  # extract labels=bug,important from query string
  labels_in_query_string = ->(request) do
   query_string = URI.parse(request.uri).query
   query_string.split('&').reduce({}) do |memo, pair|
    key, value = pair.split('=')
    memo.merge(key => value)
   end['labels']
  end

  labels_1 = labels_in_query_string.(request_1)
  labels_2 = labels_in_query_string.(request_2)

  labels_1.split(',').sort == labels_2.split(',').sort
 end
end
```

Test for Issue.important_bugs

```
require 'test_helper'

class IssueTest < ActiveSupport::TestCase

  def test_important_bug_issues
    issues = VCR.use_cassette('issue/important_bugs', match_requests_on: %i(path label_in_query_string)) do
      # ...
    end
    # ...
  end
end
```

Result of running test



```
[mikedalton@Mikes-MBP vcr-presentation (master)]$ rails test
Running via Spring preloader in process 50849
Run options: --seed 50218

# Running:


.


Finished in 0.010395s, 96.2001 runs/s, 96.2001 assertions/s.

1 runs, 1 assertions, 0 failures, 0 errors, 0 skips
[mikedalton@Mikes-MBP vcr-presentation (master)]$
```

Summary

- First example
  - GET requests
- Second example
  - POST requests
  - `match_requests_on`
    - Defaults: URI, method
- Third example
  - Delete cassette file to regenerate
  - Custom matchers

Thanks!