How to start testing your Rails app with RSpec

5 Tips and Tricks

4 Gems that go well with RSpec

Unit testing

```
class User < ApplicationRecord
  def admin?(account)
    role.include? :admin && has_account_access? account
  end
end
```

Unit testing

```ruby
# Example!
RSpec.describe User, type: :model do
  describe "#admin?" do
    subject { user.admin?(account) }
    let(:user) { create(:user, role: :admin) }
    let(:account) { create(:account) }

    before do
      allow(user).to \
        receive(:has_account_access?).with(account).and_return(account_access)
    end

    context "when the user doesn't have access to the account" do
      let(:account_access) { false }

      it { is_expected.to eq false }
    end

    context "when the user has access to the account" do
      let(:account_access) { true }

      it "returns true" do
        expect(subject).to eq(true)
      end
    end
  end
end
```

describe

- `#` instance methods,
  `.` class methods or
  nothing

context

- `with, without or when`
- `before, after and during`
- `if, unless and for`

`before` block

```
before :each do
end
```

let

- helper method
- memoized return value
- lazy-evaluated

let

```
RSpec.describe User, type: :model do
  describe "#admin?" do
    subject { user.admin?(account) }

    before do
      allow(user).to \
        receive(:has_account_access?).with(account).and_return(account_access)
    end

    context "when the user doesn't have access to the account" do
      it "returns false" do
        user = create(:user)
        account = create(:account, role: :admin)
        account_access = false

        expect(subject).to eq(false)
      end
    end

    context "when the user has access to the account" do
      it "returns true" do
        user = create(:user)
        account = create(:account, role: :admin)
        account_access = true

        expect(subject).to eq(true)
      end
    end
  end
end
```

let

```
RSpec.describe User, type: :model do
  describe "#admin?" do
    subject { user.admin?(account) }

    before do
      @user = create(:user)
      @account = create(:account)

      allow(user).to \
        receive(:has_account_access?).with(account).and_return(account_access)
    end

    context "when the user doesn't have access to the account" do
      let(:account_access) { false }

      it "returns false" do
        account_access = false

        expect(subject).to eq(false)
      end
    end

    context "when the user has access to the account" do
      let(:account_access) { true }

      it "returns true" do
        account_access = true

        expect(subject).to eq(true)
      end
    end
  end
end
```

let

```
# False positive example.

...
before do
  @complicated_query = BusinessLogic.complicated_dataset # And it actually returns some dataset value and not `nil`.
end

it "returns `nil` when something" do
  expect(@complicated_qery).to eq(nil)
end
```

Unit testing

```ruby
# Example!
RSpec.describe User, type: :model do
  describe "#admin?" do
    subject { user.admin?(account) }
    let(:user) { create(:user, role: :admin) }
    let(:account) { create(:account) }

    before do
      allow(user).to \
        receive(:has_account_access?).with(account).and_return(account_access)
    end

    context "when the user doesn't have access to the account" do
      let(:account_access) { false }

      it { is_expected.to eq false }
    end

    context "when the user has access to the account" do
      let(:account_access) { true }

      it "returns true" do
        expect(subject).to eq(true)
      end
    end
  end
end
```

subject

Explicit not named subject.

```
subject { user.admin?(account) }
```

Explicit named subject

```
subject(:is_user_admin) { user.admin?(account) }
```

One-liner syntax - GOOD

```
...

it { is_expected.to eq false }
```

One-liner syntax - BAD

```
...

it "returns true" do
  expect(subject).to eq(true)
end
```

Testing controllers

- type: :controller
- type :request

Integration tests

```ruby
RSpec.describe "login", type: :request do
  describe "POST /login" do
    it "signs in the user" do
      post login_path, params: {username: "mirko", password: "Go0DPa55w0rD"}

      expect(response).to be_successful
      expect(response.body).to include("Welcome mirko to the app")
    end
  end
end
```

Tips and tricks 1

```
def login
  result = LoginUser.call(session_params)

  if result.success?
    session[:user_token] = result.token
    redirect_to result.user
  else
    flash.now[:message] = t(result.message)
    render :new
  end
end
```

Stub everything you can

```ruby
RSpec.describe "login", type: :request do
  describe "POST /login" do
    let(:user) { create(:user, username: "mirko", password: "Go0DPa55w0rD") }
    let(:user_interactor) do
      OpenStruct.new(result: result_value, token: "1234")
    end
    let(:session_params) do
      { username: user.username, password: user.password }
    end

    before do
      allow(LoginUser).to receive(:call).with(session_params).and_return(user_interactor)
    end

    context "when success" do
      let(:result_value) { true }

      it "signs in the user" do
        post login_path, params: session_params

        expect(LoginUser).to have_received(:call).with(session_params)
        expect(response).to be_successful
        expect(response.body).to include("Welcome mirko to the app")
      end
    end
  end
end
```

Tips and tricks 2

Do not save to the database if you do not have to

BAD

```
let(:user) { create(:user, username: "mirko", password: "Go0DPa55w0rD") }
```

Good

```
let(:user) { instance_double(User, username: "mirko", password: "Go0DPa55w0rD") }
```

Tips and tricks 3

`anything` general matcher

```
before do
  allow(ExampleClass).to receive(:call).with(id: 12, name: anything)
end
```

Tips and tricks 4

`aggregate_failures`

```
it "signs in the user" do
  post login_path, params: session_params

  expect(response).to be_successful
  expect(response.body).to include("Welcome mirko to the app")
  expect(LoginUser).to have_received(:call).with(session_params)
end
```

Tips and tricks 4

```
it "signs in the user", aggregate_failures: true do
  post login_path, params: session_params

  expect(response).to be_successful
  expect(response.body).to include("Welcome mirko to the app")
  expect(LoginUser).to have_received(:call).with(session_params)
end
```

Tips and tricks 4

```
it "signs in the user" do
  post login_path, params: session_params

  expect(LoginUser).to have_received(:call).with(session_params)

  aggregate_failures "request response" do
    expect(response).to be_successful
    expect(response.body).to include("Welcome mirko to the app")
  end
end
```

Tips and tricks 5

Running specific spec, context or description.

1. With the line number

```
bundle exec rspec spec/example_spec.rb:18
```

2. With regex

```
bundle exec rspec spec/example_spec.rb -e "a part or a full text from the spec description"
```

Tips and tricks 5

3. With tags

```
describe "tagged specs" do
  it "focused example that I'm just developing", :focus => true do; end
  it "special example", :focus => 'special' do; end
  it "untagged example" do; end
end

rspec spec --tag focus                # Runs specs that have :focus => true
rspec spec --tag focus:special        # Run specs that have :focus => special
rspec spec --tag focus ~skip          # Run tests except those with :focus => true
```

Gems that go well with RSpec 1

FactoryBot

Factory

```
FactoryBot.define do
  factory :user do
    username { "Test name" }
    password { "Go0DPa55w0rD" }

    trait :admin do
      role { :admin }
    end
  end
end
```

Gems that go well with RSpec 1

FactoryBot

Usage

```
create(:user)

# or

create(:user, username: "mirko")

# or

create(:user, :admin)
```

Gems that go well with RSpec 2

Faker

```
FactoryBot.define do
  factory :user do
    username { Faker::Internet.username }
    password { Faker::Internet.password }
  end
end
```

Gems that go well with RSpec 3

Timecop

```
describe "some set of tests to mock" do
  before do
    Timecop.freeze(Time.local(1990))
  end

  after do
    Timecop.return
  end

  it "should do blah blah blah" do
  end
end
```

It is similar for `travel` method.

Gems that go well with RSpec 4

VCR

```
...

it do
  VCR.use_cassette("file_to_save_the_request_to") do
    get "http..."
  end
end
```