Guía 3: Trabajando con Rspec y Ruby

Introducción

Rspec es una herramienta de pruebas escrita en Ruby, mayormente utilizada para el desarrollo en base a pruebas o TDD y desarrollo en base comportamiento o DBB. En esta guía el estudiante pondrá en práctica los conocimientos obtenidos en el desarrollo de las guías anteriores en base a los principios básicos del lenguaje Ruby, de igual forma podrá introducirse en lo que es el desarrollo en base a pruebas utilizando la gema Rspec.

Objetivos

- Aprender a usar la herramienta Rspec.
- Crear pruebas en Ruby utilizando Rspec.
- Escribir código Ruby y hacer funcionar los test.

Tiempo

- Una sesión de clase.

Requerimientos

Software	Hardware
Sistema Debian 9 virtualizado en Virtual box con: • Ruby versión 2.4.1 • Sublimetext • Rspec	Computadora con características: • Memoria ram mínimo 2GB • Procesador mínimo 2.1 GHz

Referencia

- David Chelimsky. The Rspec book. https://goo.gl/GKNzwK
- Ed Wassemann. Pruebas RSpec para principiantes, Parte 1. https://code.tutsplus.com/es/articles/rspec-testing-for-beginners-part-1--cms-26716
- saasbook. Ruby Intro. https://github.com/saasbook/hw-ruby-intro
- Ruby community. Ruby a programmer´s Best friend. https://www.ruby-lang.org/en/documentation/quickstart/

Desarrollo

1. Crear un directorio donde almacenar el proyecto.

   1.1. Crear un archivo llamado Gemfile dentro del directorio creado anteriormente, este archivo es donde se agregará la gema que necesita el proyecto, en este caso solo se utilizará la gema rspec para realizar los tests al código.

   1.2. Abrir el archivo Gemfile y agregar las siguientes líneas, para poder integrar rspec al proyecto.

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

2. En el terminal, ubicarse dentro del directorio de trabajo y escribir los siguientes comandos.

   2.1. Instalar la gema rspec, que es la que se utiliza para trabajar con TDD y que servirá para elaborar los tests.

```
$ bundle install
```

   2.2. Ejecutar rspec.

Se crearán automáticamente dos archivos: .rspec y spec/spec_helper.rb que son los archivos que utiliza rspec para correr los tests.

```
$ bundle exec rspec --init
```

3.

   En el directorio raíz del proyecto, crear una carpeta nueva llamada lib

   3.1. Dentro de la carpeta lib, crear un archivo ruby.rb y agregar lo siguiente.

```
def intro array
#escribir el código
end

def nombre nom
#escribir el código
end

def datos? array
#escribir el código
end
```

Uno de los principales motivos de crear tests, es que dan garantía de un código de mejor calidad que ha sido sometido a diferentes eventualidades y todos los posibles bugs fueron corregidos. Como se observa, en el código anterior existen 3 métodos que serán donde se va escribir el código Ruby en base a los tests que se aplicarán en el proyecto, mostrado a continuación.

4. Dentro de la carpeta **spec,** crear un archivo **rspec_1.rb** y copiar el siguiente código, este código es cada uno de los tests aplicados, para escribir el código Ruby dentro de los métodos que están en el archivo **ruby.rb**.

```
require "ruby.rb"

describe "#intro" do

        it "Debe estar definada" do
                expect{intro([1,2,3,4])}.not_to raise_error
        end

        it "retorna la suma del array" do
                expect(intro ([2,6,10,5,2])).to eq(25)
                expect(intro ([15,6,10,19])).to eq(50)
                expect(intro ([15,6,4,5])).to eq(30)
         end

        it "retorna 0 si el array está vacío" do
                expect(intro([])).to eq(0)
         end

         it "retorna el valor del único elemento del array" do
                expect(intro([5])).to eq(5)
         end
end

describe "#nombre" do

        it "Debe iniciar con Mayúscula y concatena la cadena con la palabra hello" do
                expect(nombre("Pedro")).to eq("hello Pedro")
                expect(nombre("Juan")).to eq("hello Juan")
                expect(nombre("María")).to eq("hello María")
        end

        it "No trabaja con un string vacío" do
                expect(nombre("")).to eq("vacío")
        end
end

describe "#datos?" do

        it "retorna true si el tamaño del array es > 0" do
                expect(datos?([2,1,5,6])).to be true
                end
        it "retorna false si el tamaño del array es <= 0" do
                expect(datos?([])).to be false
        end
end
```

5. En el terminal, ejecutar el siguiente comando para verificar el funcionamiento de los test.

```
$ rspec spec/rspec_1.rb
```

Al ejecutar el comando anterior mostrará un conjunto de mensajes en color rojo, los cuales corresponden a cada uno de los tests que se están aplicando en el ejercicio, cuando los tests se muestran en color rojo es debido a que los tests ejecutados están fallando, en relación a la funcionalidad del código al cual se le están aplicando cada uno de ellos.

```
debian@debian:~/Proyectos_RoR/Rspec_test$ rspec spec/rspec_1.rb
.FFFFFFF

Failures:

  1) #intro retorna la suma del array
     Failure/Error: expect(intro ([2,6,10,5,2])).to eq(25)

       expected: 25
            got: nil

       (compared using ==)
     # ./spec/rspec_1.rb:10:in `block (2 levels) in <top (required)>'

  2) #intro retorna 0 si el array es vacio
     Failure/Error: expect(intro([])).to eq(0)

       expected: 0
            got: nil

       (compared using ==)
     # ./spec/rspec_1.rb:16:in `block (2 levels) in <top (required)>'

  3) #intro retorna el valor del unico elemento del array
     Failure/Error: expect(intro([5])).to eq(5)

       expected: 5
            got: nil

       (compared using ==)
     # ./spec/rspec_1.rb:20:in `block (2 levels) in <top (required)>'
```

Al final de todos los mensajes, se muestra exactamente los tests que están siendo ejecutados.

```
Failed examples:

rspec ./spec/rspec_1.rb:9 # #intro retorna la suma del array
rspec ./spec/rspec_1.rb:15 # #intro retorna 0 si el array es vacio
rspec ./spec/rspec_1.rb:19 # #intro retorna el valor del unico elemento del array
rspec ./spec/rspec_1.rb:27 # #nombre Debe iniciar con Mayuscula y concatena la cadena con hello
rspec ./spec/rspec_1.rb:33 # #nombre No trabaja con un string vacio
rspec ./spec/rspec_1.rb:40 # #datos? retorna true si el tamaño del array es > 0
rspec ./spec/rspec_1.rb:43 # #datos? retorna false si el tamaño del array es <= 0
```

El desarrollo en base a pruebas o TDD no es más que, primero escribir los tests y correrlos esperando que fallen; que es cuando se muestran los mensajes en rojo, escribir código y volver a correr los tests, esperando que estos pasen y que se muestren los mensajes en verde que significa que el código no contiene errores o que los tests funcionan de manera correcta.

Ejercicios propuestos para ser entregados al docente

1. Realizar cada uno de los enunciados de la guía.

2. Escribir el código Ruby necesario dentro cada una de los métodos creados en el archivo **lib/ruby.rb** para aprobar todos los tests, y que muestre los mensajes en color verde como se observa.

```
debian@debian:~/Proyectos_RoR/Rspec_test$ rspec spec/rspec_1.rb
........

Finished in 0.00657 seconds (files took 0.14529 seconds to load)
8 examples, 0 failures
```

Extra: escribir un conjunto de tests en relación a lo aprendido en la guía dentro del archivo **rspec_1.rb,** para luego crear un método y escribir código ruby de tal manera que pueda hacer funcionar los test a como se trabajó en la guía.

3 Supongamos que tienes la siguiente clase Calculator en un archivo llamado calculator.rb:

```
class Calculator
  def add(a, b)
    a + b
  end

  def subtract(a, b)
    a - b
  end

  def multiply(a, b)
    a * b
  end

  def divide(a, b)
    raise ZeroDivisionError if b == 0
    a / b
  end
end
```

Ahora, vamos a crear pruebas para esta clase utilizando RSpec. Crea un archivo llamado calculator_spec.rb para escribir tus pruebas:

```
# calculator_spec.rb

require_relative 'calculator'  # Asegúrate de ajustar la ruta según la ubicación de tu archivo calculator.rb

RSpec.describe Calculator do
```

```
  let(:calculator) { Calculator.new }

  describe "#add" do
    it "sums two numbers" do
      result = calculator.add(2, 3)
      expect(result).to eq(5)
    end
  end

  describe "#subtract" do
    it "subtracts two numbers" do
      result = calculator.subtract(5, 3)
      expect(result).to eq(2)
    end
  end

  describe "#multiply" do
    it "multiplies two numbers" do
      result = calculator.multiply(4, 3)
      expect(result).to eq(12)
    end
  end

  describe "#divide" do
    it "divides two numbers" do
      result = calculator.divide(10, 2)
      expect(result).to eq(5)
    end

    it "raises an error when dividing by zero" do
      expect { calculator.divide(10, 0) }.to raise_error(ZeroDivisionError)
    end
  end
end
```

Deberías ver la salida de RSpec que indica si todas las pruebas pasaron o no.