

Taller IS 2023 - Actividad Nro: 1



Materia: Ingeniería de Software

Profesores: Marcela Daniele - Marcelo Uva - Ariel Arsaute - Daniela Solivellas

Alumnos: Balestra Edgar Agustin - Bernardi Quiroga Matias - Olivo Lucas Gabriel

Metodología

La metodología que utilizamos para medir la cobertura de nuestro código fue hacer uso de “test unitarios” o “test de unidad” escritos en Ruby usando el framework RSpec. Esta metodología implica aislar componentes individuales del código y comprobarlos de manera independiente para asegurarse de que funcionen correctamente.

La cobertura fue medida utilizando la herramienta de cobertura simpleCov

Test para Model User

Durante el final del primer cuatrimestre, implementamos pruebas sobre el modelo User, en el que este debía contener un nombre y email únicos, o sea que no puede haber otro de usuario que utilice alguno de estos, y también debía contar con una password. Además, en el modelo de usuario se encontraban dos funciones que son utilizadas en Schema para sumarle o restarle puntos al usuario adecuadamente cuando responde una pregunta.

Nuestros primeros tests sólo abarcaban la parte de validación del usuario, cubriendo solo el 33,33% de cobertura del modelo User, ya que no teníamos casos de prueba para cubrir las funciones de puntos del usuario. Una cobertura muy baja de acuerdo al modelo que teníamos implementado.

All Files (73.33%)

Generated about a minute ago

All Files (73.33% covered at 0.73 hits/line)

4 files in total.
45 relevant lines, 33 lines covered and 12 lines missed. (73.33%)

Search:

File	% covered [▲]	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
models/user.rb	33.33 %	29	18	6	12	0.33
config/environment.rb	100.00 %	5	3	3	0	1.00
models/init.rb	100.00 %	1	1	1	0	1.00
spec/models/user_spec.rb	100.00 %	37	23	23	0	1.00

Showing 1 to 4 of 4 entries

Generated by simplecov v0.22.0 and simplecov-html v0.12.3 using RSpec

models/user.rb

33.33% lines covered

18 relevant lines. 6 lines covered and 12 lines missed.

```
1. class User < ActiveRecord::Base
2.
3.   validates :name, presence: true, uniqueness: true
4.   validates :email, presence: true, uniqueness: true
5.   validates :password, presence: true
6.
7.   def sum_points (question)
8.     if question.difficulty.to_i == 1
9.       self.points += 10
10.     elsif question.difficulty.to_i == 2
11.       self.points += 20
12.     else
13.       self.points += 30
14.     end
15.     self.update(points: self.points)
16.   end
17.
18.   def rest_points (question)
19.     if question.difficulty.to_i == 1
20.       self.points -= 10
21.     elsif question.difficulty.to_i == 2
22.       self.points -= 20
23.     else
24.       self.points -= 30
25.     end
26.     self.update(points: self.points)
27.   end
28.
29. end
```

models/user.rb

```
er / spec / models / user_spec.rb
# spec/models/user_spec.rb
require 'sinatra/activerecord'
require_relative '../models/init.rb'

describe User do

  it "is a valid user" do
    user = User.new(name: "romanino", email: "rique1me10torero@gmail.com", password: "avostegusta?")
    user.reppw = "avostegusta?"
    expect(user).to be_valid
  end

  it "is invalid without a username" do
    user = User.new(email: "romi10@gmail.com", password: "bocal23")
    user.reppw = "bocal23"
    expect(user).not_to be_valid
  end

  it "is invalid without an email" do
    user = User.new(name: "marten", password: "bocaaa")
    user.reppw = "bocaaa"
    expect(user).not_to be_valid
  end

  it "is invalid without a password" do
    user = User.new(name: "liomessi10", email: "soydeboca@gmail.com")
    expect(user).not_to be_valid
  end

  it "is invalid with a duplicate email address" do
    existing_user = User.create(name: "pichonadvincula", email: "peru@example.com", password: "zambradios")
    user = User.new(name: "cheftravel", email: "peru@example.com", password: "latercera")
    expect(user).not_to be_valid
    existing_user.destroy
  end

  it "is invalid with a duplicate username" do
    existing_user = User.create(name: "MarcosRED", email: "bocaelmasgrande@example.com", password: "matafuegos123")
    user = User.new(name: "MarcosRED", email: "tefuistealab@mail.com", password: "password")
    expect(user).not_to be_valid
    existing_user.destroy
  end

end
```

Posteriormente incluimos en el modelo de Users que el nombre del usuario y la contraseña tenga un rango de cuantos caracteres pueden tener, y que el email debe tener incluido un @, con un carácter antes y uno después.

Se agregó además que cuando un usuario ingrese una contraseña, esta debe coincidir con una ya existente.

Los test para cubrir los casos de prueba de las funciones de puntos aún no están implementados, teniendo en total un 47,83% de cobertura del modelo.

All Files (82.61%) Generated 3 minutes ago

All Files (82.61% covered at 1.06 hits/line)

4 files in total.
69 relevant lines, 57 lines covered and 12 lines missed. (82.61%)

Search:

File	% covered [▲]	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
models/user.rb	47.83 %	38	23	11	12	1.17
config/environment.rb	100.00 %	5	3	3	0	1.00
models/init.rb	100.00 %	1	1	1	0	1.00
spec/models/user_spec.rb	100.00 %	66	42	42	0	1.00

Showing 1 to 4 of 4 entries

Generated by [simplecov](#) v0.22.0 and [simplecov-html](#) v0.12.3 using [RSpec](#)

models/user.rb

47.83% lines covered

23 relevant lines, 11 lines covered and 12 lines missed.

1.	class User < ActiveRecord::Base	1
2.	validates :name, presence: true, uniqueness: true, length: { in: 3..20 }	1
3.	validates :email, presence: true, uniqueness: true, format: { with: /\A[\w\.\s]+\@[^\s]+\z/, message: "debe ser una dirección de correo electrónico válida" } # verifica que haya un caracter antes y dsp del @	1
4.	validates :password, presence: true, length: { minimum: 4, too_short: "te quieres ir hackeado?" }	1
5.		
6.	validate :password_match	1
7.		
8.	attr_accessor :reppw	1
9.		
10.	def password_match	1
11.	if password != reppw	12
12.	errors.add(:password_confirmation, "debe coincidir con la contraseña")	6
13.	end	
14.	end	
15.		
16.	def sum_points(question)	1
17.	if question.difficulty.to_i == 1	
18.	self.points += 10	
19.	elsif question.difficulty.to_i == 2	
20.	self.points += 20	
21.	else	
22.	self.points += 30	
23.	end	
24.	self.update(points: self.points)	
25.	end	
26.		
27.	def rest_points(question)	1
28.	if question.difficulty.to_i == 1	
29.	self.points -= 10	
30.	elsif question.difficulty.to_i == 2	
31.	self.points -= 20	
32.	else	
33.	self.points -= 30	
34.	end	
35.	self.update(points: self.points)	
36.	end	
37.		
38.	end	

models/user.rb

```

it "is invalid if pws not matches" do
  user = User.new(name: "MarcosRED", email: "tefuistealab@mail.com", password: "password", reppw: "paswor")
  expect(user).not_to be_valid
end

it "is valid if pws matches" do
  user = User.new(name: "marquitos", email: "tefuistealab@gmail.com", password: "paswor")
  user.reppw = "paswor"
  expect(user).to be_valid
end

it "is invalid with a email without @" do
  user = User.new(name: "koka", email: "mail.com", password: "password")
  expect(user).not_to be_valid
end

it "is invalid with a too short pw" do
  user = User.new(name: "changuito", email: "elchango7eballos@hotmail.com", password: "aaa")
  user.reppw = "aaa"
  expect(user).not_to be_valid
end

```

Luego ampliamos nuestros tests para abordar los casos relacionados con la puntuación del usuario, evaluando cómo se asignaban los puntos al usuario en función de la dificultad de las preguntas y si sus respuestas eran correctas o incorrectas. Esta ampliación nos permitió alcanzar una cobertura completa del 100% para nuestro modelo User.

All Files (100.0%) Generated 3 minutes ago

All Files (100.0% covered at 1.45 hits/line)

4 files in total.
110 relevant lines, 110 lines covered and 0 lines missed. (100.0%)

File	% covered ^	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
config/environment.rb	100.00 %	5	3	3	0	1.00
models/init.rb	100.00 %	1	1	1	0	1.00
models/user.rb	100.00 %	43	26	26	0	2.88
spec/models/user_spec.rb	100.00 %	117	80	80	0	1.00

Showing 1 to 4 of 4 entries

Generated by [simplecov](#) v0.22.0 and [simplecov-html](#) v0.12.3 using [RSpec](#)

```

1. class User < ActiveRecord::Base
2.   validates :name, presence: true, uniqueness: true, length: { in: 3..20 }
3.   validates :email, presence: true, uniqueness: true, format: { with: /\A(?:[a-z0-9]+(?:\.[a-z0-9]+)*)\z/, message: "debe ser una dirección de correo electrónico válida" } # verifica que haya un caracter antes y desp del @
4.   validates :password, presence: true, length: { minimum: 4, too_short: "te quieres ir hackeado?" }
5.
6.   validate :password_match
7.   after_initialize :set_default_points
8.
9.   attr_accessor :reppw
10.
11.   def password_match
12.     if password != reppw
13.       errors.add(:password_confirmation, "debe coincidir con la contraseña")
14.     end
15.   end
16.
17.   def set_default_points
18.     self.points ||= 0
19.   end
20.
21.   def sum_points(question)
22.     if question.difficulty.to_i == 1
23.       self.points += 10
24.     elsif question.difficulty.to_i == 2
25.       self.points += 20
26.     else
27.       self.points += 30
28.     end
29.     self.update(points: self.points)
30.   end
31. end

```

```

31.
32. def rest_points(question)
33.   if question.difficulty.to_i == 1
34.     self.points -= 10
35.   elsif question.difficulty.to_i == 2
36.     self.points -= 20
37.   else
38.     self.points -= 30
39.   end
40.   self.update(points: self.points)
41. end
42.
43. end

```

models/user.rb

```

#TESTS for points tratament
it "updates points with a question of difficulty 1" do
  user = User.new(name: "colo", email: "colobarco@gmail.com", password: "aguantebocal23")
  user.reppw = "aguantebocal23"
  question = double("¿Cual es el manjar tipico de Argentina?", difficulty: 1)
  user.sum_points(question)
  expect(user.points).to eq(10)
end

it "updates points with a question of difficulty 2" do
  user = User.new(name: "lucasjanson", email: "jansonson@mail.com", password: "bocaaaaa12")
  user.reppw = "bocaaaaa12"
  question = double("¿Qué tipo de madera es tradicionalmente utilizada para el fuego del asado?", difficulty: 2)
  user.points = 100
  user.sum_points(question)
  expect(user.points).to eq(120)
end

it "updates points with a question of difficulty 3" do
  user = User.new(name: "Carlos", email: "apache@mail.com", password: "bocaaaaa12")
  user.reppw = "bocaaaaa12"
  question = double("¿Qué rol cumple la vainilla en su preparación?", difficulty: 3)
  user.sum_points(question)
  expect(user.points).to eq(30)
end

it "updates points negatively with a question of difficulty 1" do
  user = User.new(name: "hamburguesitafabra", email: "ffabra@example.com", password: "liberenAlPanitaVilla123")
  user.reppw = "liberenAlPanitaVilla123"
  question = double("¿Cuáles son los ingredientes para preparar dulce de leche?", difficulty: 1)
  user.rest_points(question)
  expect(user.points).to eq(-10)
end

it "updates points negatively with a question of difficulty 2" do
  user = User.new(name: "javiG0Dcia", email: "javmancol@mail.com", password: "aBreyLeFaltaTodavia123")
  user.reppw = "aBreyLeFaltaTodavia123"
  question = double("¿Cuál es el proceso principal para lograr un buen asado argentino?", difficulty: 2)
  user.rest_points(question)
  expect(user.points).to eq(-20)
end

it "updates points negatively with a question of difficulty 3" do
  user = User.new(name: "Roncaglia", email: "troncaglia2@mail.com", password: "oPasaLaPelotaoPasaElJugador")
  user.reppw = "oPasaLaPelotaoPasaElJugador"
  user.points = 100
  question = double("¿Por qué el Dulce de Leche se debe revolver constantemente durante su preparación?", difficulty: 3)
  user.rest_points(question)
  expect(user.points).to eq([100-30])
end

```

Test para Model Question

En nuestro modelo de Question, una pregunta debe contener texto, un nivel de dificultad y una respuesta correcta asociada; para ello realizamos una serie de casos de pruebas en los que logramos obtener un 100% de cobertura sobre el modelo.

```
models/question.rb
100.0% lines covered
7 relevant lines, 7 lines covered and 0 lines missed.

1. require 'active_record'
2. require_relative 'answer'
3.
4. class Question < ActiveRecord::Base
5.   has_many :answers
6.
7.   validates :text, presence: true
8.   validates :levels_id, presence: true, numericality: { only_integer: true, greater_than: 0 }
9.   validates :answer_id, presence: true, numericality: { only_integer: true, greater_than: 0 } # has a correct associated answer
10.
11.
12. end
```

```
cheftravel > models > question.rb > Question
1 require 'active_record'
2 require_relative 'answer'
3
4 class Question < ActiveRecord::Base
5   has_many :answers
6
7 end
```

```

cheftravel > spec > models > question_spec.rb
1 require 'sinatra/activerecord'
2 require_relative '../models/init.rb'
3
4 describe Question do
5
6   it 'is a valid question' do
7     question = Question.new(text: "Estas leyendo?", levels_id: 1, answer_id: 2)
8     expect(question).to be_valid
9   end
10
11   it 'is invalid with no text' do
12     question = Question.new(levels_id: 1, answer_id: 2)
13     expect(question).not_to be_valid
14   end
15
16   it 'is invalid with no correct answer associated' do
17     question = Question.new(text: "Que te parece?", levels_id: 1)
18     expect(question).not_to be_valid
19   end
20
21   it 'is invalid with negative id of answer associated' do
22     question = Question.new(text: "Seguis leyendo?", levels_id: 1, answer_id: -2)
23     expect(question).not_to be_valid
24   end
25
26 end

```

```

cheftravel > models > question.rb > ...
1 require 'active_record'
2 require_relative 'answer'
3
4 class Question < ActiveRecord::Base
5   has_many :answers
6   validates_associated :answers
7
8   validates :text, presence: true
9   validates :levels_id, presence: true, numericality: { only_integer: true, greater_than: 0 }
10  validates :answer_id, presence: true, numericality: { only_integer: true, greater_than: 0 } # has a correct associated answer
11 end
12
13 ,

```

Conclusión

En términos generales, nuestra metodología se basó en la aplicación de pruebas unitarias utilizando el framework RSpec en Ruby y la herramienta simpleCov para medir la cobertura de código. El objetivo era asegurarnos de que cada componente del código se evaluará de manera independiente para confirmar su correcto funcionamiento.

Nos centramos en la aplicación de pruebas unitarias y la mejora continua de la cobertura de código para garantizar la calidad y confiabilidad de nuestra aplicación, garantizando que todas las funcionalidades estén correctamente probadas y funcionando según lo previsto.