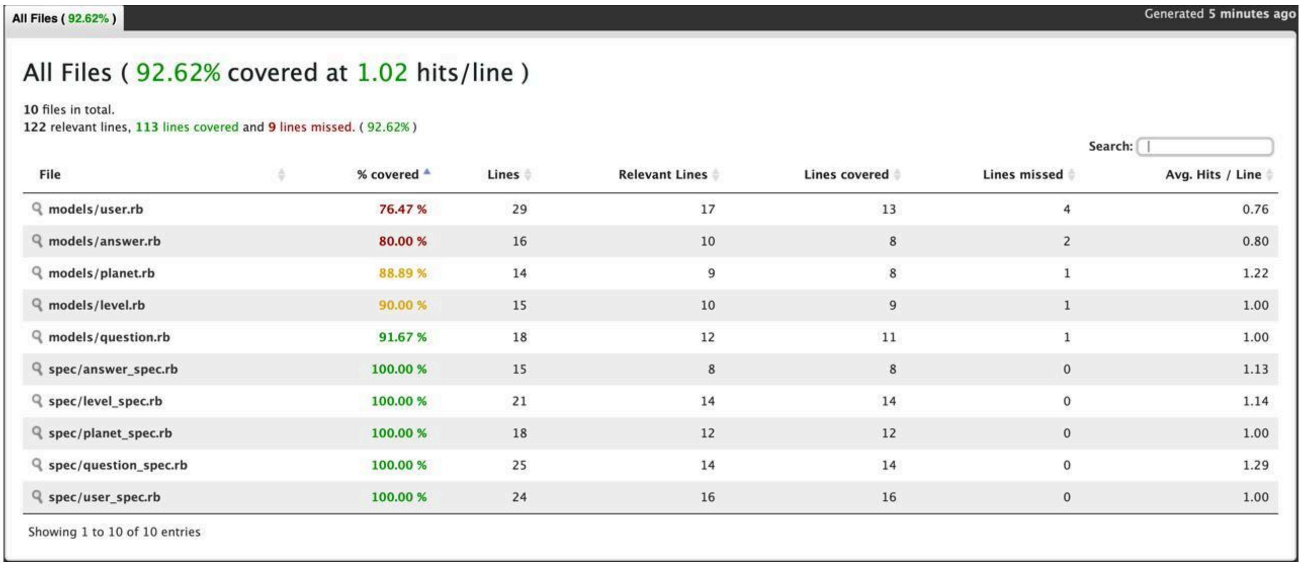
Tarea N°1 - Avaro, Cornejo, Marquez

Antes de realizar las modificaciones en los tests, la cobertura era del 92.62% y en donde menor cobertura había era en los test de user, answer, planet y question.



All Files (92.62%)

Generated 5 minutes ago

All Files (92.62% covered at 1.02 hits/line)

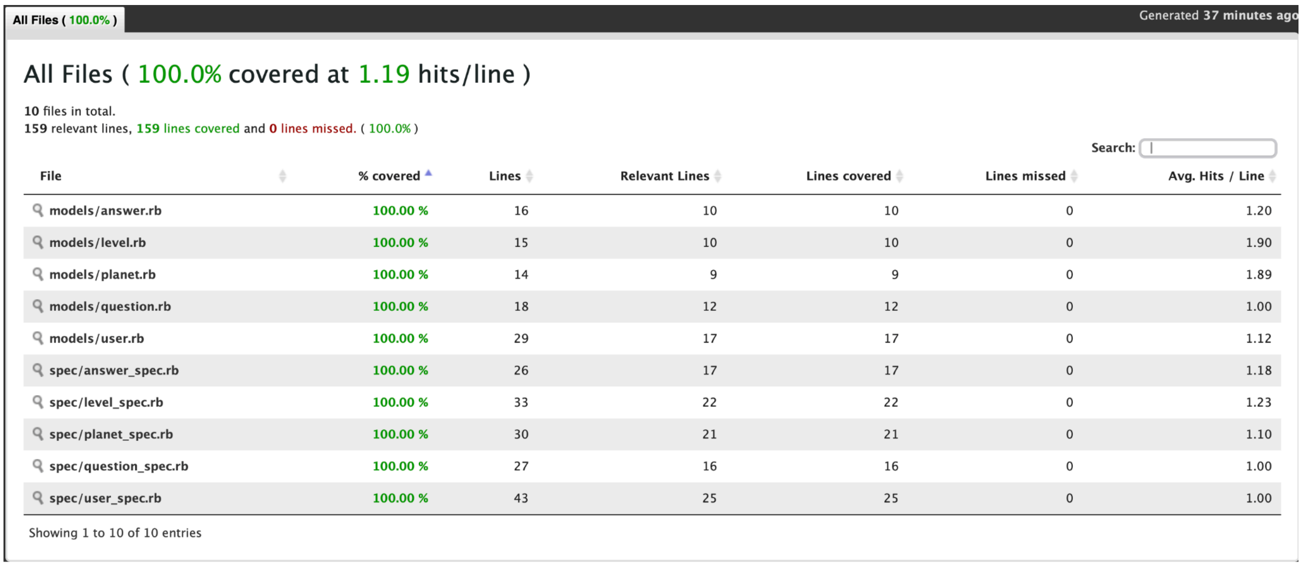
10 files in total.
122 relevant lines, 113 lines covered and 9 lines missed. (92.62%)

Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
models/user.rb	76.47 %	29	17	13	4	0.76
models/answer.rb	80.00 %	16	10	8	2	0.80
models/planet.rb	88.89 %	14	9	8	1	1.22
models/level.rb	90.00 %	15	10	9	1	1.00
models/question.rb	91.67 %	18	12	11	1	1.00
spec/answer_spec.rb	100.00 %	15	8	8	0	1.13
spec/level_spec.rb	100.00 %	21	14	14	0	1.14
spec/planet_spec.rb	100.00 %	18	12	12	0	1.00
spec/question_spec.rb	100.00 %	25	14	14	0	1.29
spec/user_spec.rb	100.00 %	24	16	16	0	1.00

Showing 1 to 10 of 10 entries

Una vez identificados los tests que hacían falta mejorar, implementamos más ítems hasta llegar al 100% de cobertura de los modelos:



All Files (100.0%)

Generated 37 minutes ago

All Files (100.0% covered at 1.19 hits/line)

10 files in total.
159 relevant lines, 159 lines covered and 0 lines missed. (100.0%)

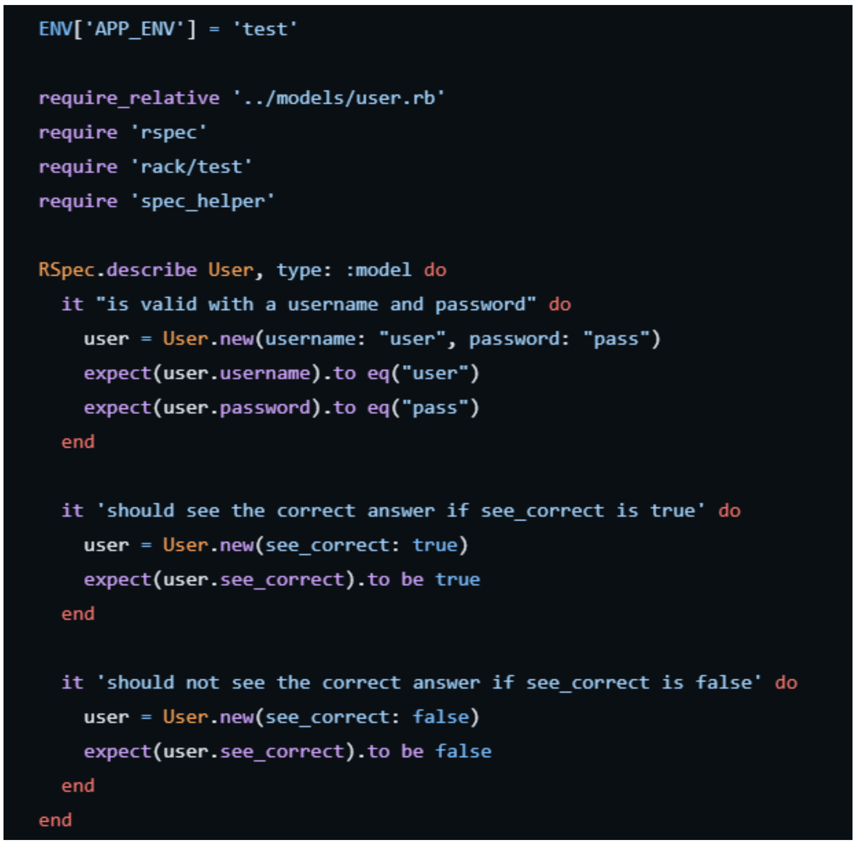
Search:

File	% covered	Lines	Relevant Lines	Lines covered	Lines missed	Avg. Hits / Line
models/answer.rb	100.00 %	16	10	10	0	1.20
models/level.rb	100.00 %	15	10	10	0	1.90
models/planet.rb	100.00 %	14	9	9	0	1.89
models/question.rb	100.00 %	18	12	12	0	1.00
models/user.rb	100.00 %	29	17	17	0	1.12
spec/answer_spec.rb	100.00 %	26	17	17	0	1.18
spec/level_spec.rb	100.00 %	33	22	22	0	1.23
spec/planet_spec.rb	100.00 %	30	21	21	0	1.10
spec/question_spec.rb	100.00 %	27	16	16	0	1.00
spec/user_spec.rb	100.00 %	43	25	25	0	1.00

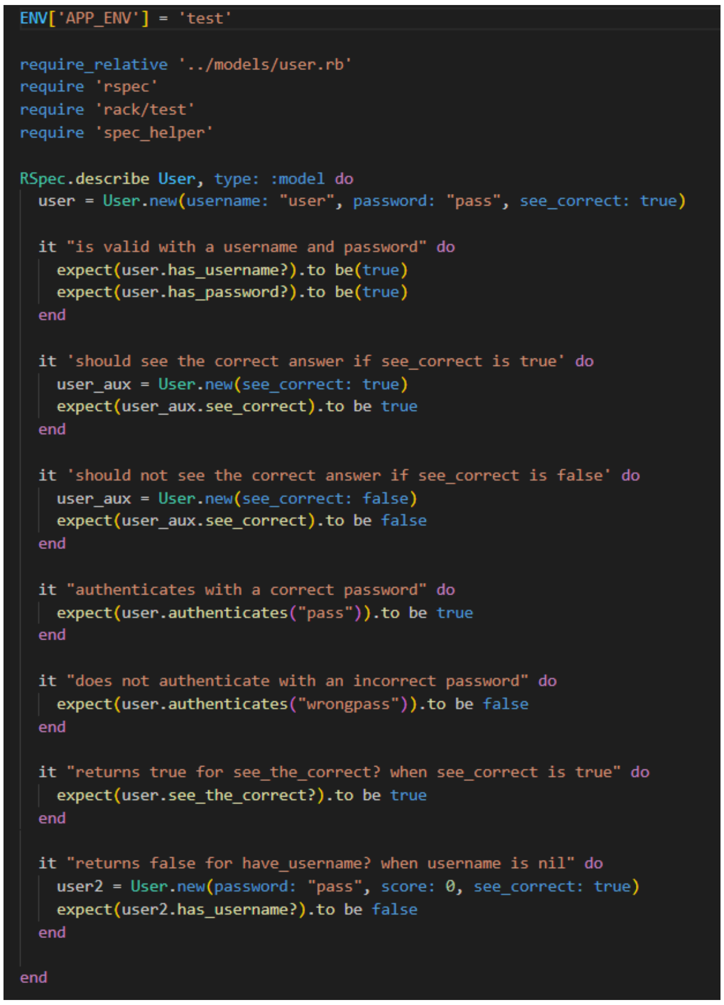
Showing 1 to 10 of 10 entries

Los cambios realizados fueron los siguientes:

Modelo User:



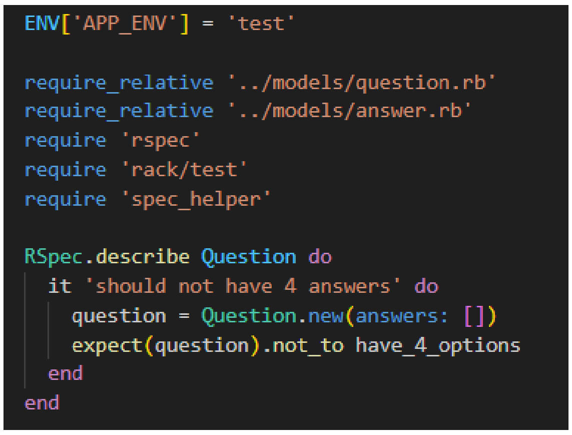
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/user.rb'
require 'rspec'
require 'rack/test'
require 'spec_helper'

RSpec.describe User, type: :model do
  it "is valid with a username and password" do
    user = User.new(username: "user", password: "pass")
    expect(user.username).to eq("user")
    expect(user.password).to eq("pass")
  end

  it 'should see the correct answer if see_correct is true' do
    user = User.new(see_correct: true)
    expect(user.see_correct).to be true
  end

  it 'should not see the correct answer if see_correct is false' do
    user = User.new(see_correct: false)
    expect(user.see_correct).to be false
  end
end
```



```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/user.rb'
require 'rspec'
require 'rack/test'
require 'spec_helper'

RSpec.describe User, type: :model do
  user = User.new(username: "user", password: "pass", see_correct: true)

  it "is valid with a username and password" do
    expect(user.has_username?).to be(true)
    expect(user.has_password?).to be(true)
  end

  it 'should see the correct answer if see_correct is true' do
    user_aux = User.new(see_correct: true)
    expect(user_aux.see_correct).to be true
  end

  it 'should not see the correct answer if see_correct is false' do
    user_aux = User.new(see_correct: false)
    expect(user_aux.see_correct).to be false
  end

  it "authenticates with a correct password" do
    expect(user.authenticates("pass")).to be true
  end

  it "does not authenticate with an incorrect password" do
    expect(user.authenticates("wrongpass")).to be false
  end

  it "returns true for see_the_correct? when see_correct is true" do
    expect(user.see_the_correct?).to be true
  end

  it "returns false for have_username? when username is nil" do
    user2 = User.new(password: "pass", score: 0, see_correct: true)
    expect(user2.has_username?).to be false
  end

end
```

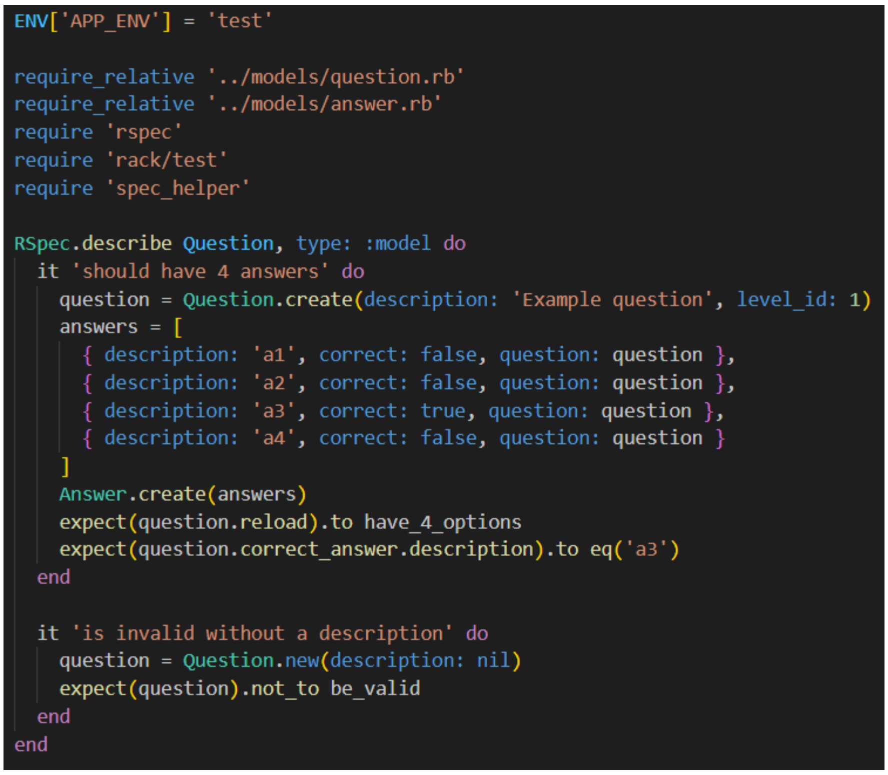
En este modelo lo que agregamos fueron los últimos 4 tests, ya que no se testeaba en su totalidad el modelo, como por ejemplo, el método authenticates.

Modelo Question:



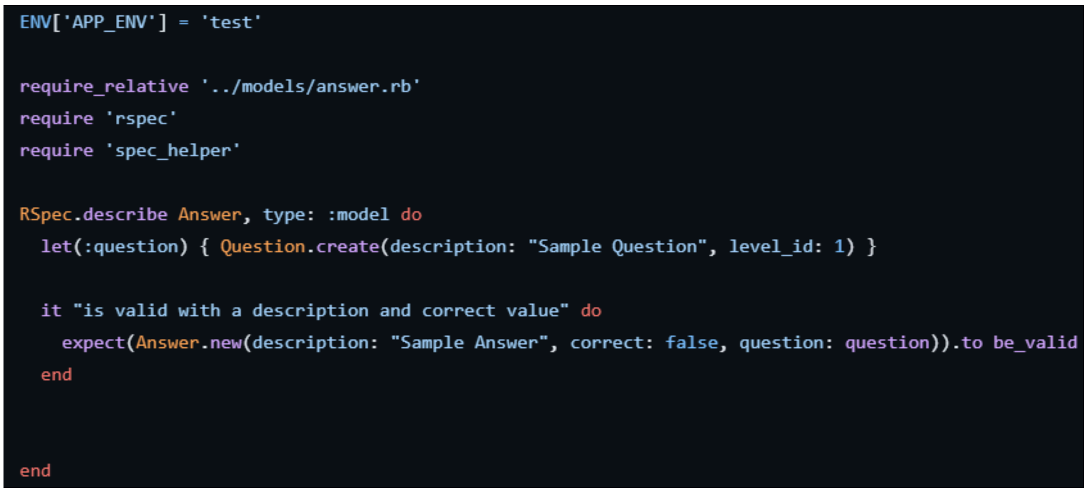
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/question.rb'
require_relative '../models/answer.rb'
require 'rspec'
require 'rack/test'
require 'spec_helper'

RSpec.describe Question do
  it 'should not have 4 answers' do
    question = Question.new(answers: [])
    expect(question).not_to have_4_options
  end
end
```

En este modelo cambiamos la forma en la que se testeaban las opciones de cada pregunta, en un mismo tests testeamos inicialmente que la pregunta tenga 4 opciones, pero luego agregamos que tenga una sola opción correcta. Además agregamos el ítem que prueba que una pregunta no es válida si no tiene descripción, por lo tanto el test quedó de la siguiente manera.



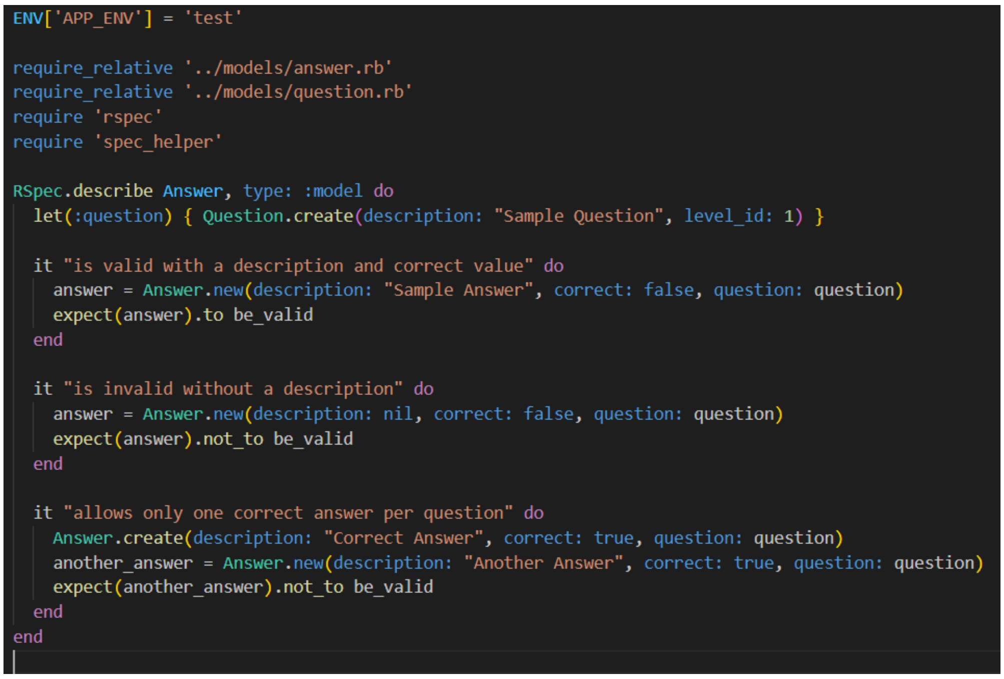
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/question.rb'
require_relative '../models/answer.rb'
require 'rspec'
require 'rack/test'
require 'spec_helper'

RSpec.describe Question, type: :model do
  it 'should have 4 answers' do
    question = Question.create(description: 'Example question', level_id: 1)
    answers = [
      { description: 'a1', correct: false, question: question },
      { description: 'a2', correct: false, question: question },
      { description: 'a3', correct: true, question: question },
      { description: 'a4', correct: false, question: question }
    ]
    Answer.create(answers)
    expect(question.reload).to have_4_options
    expect(question.correct_answer.description).to eq('a3')
  end

  it 'is invalid without a description' do
    question = Question.new(description: nil)
    expect(question).not_to be_valid
  end
end
```

Modelo Answer:



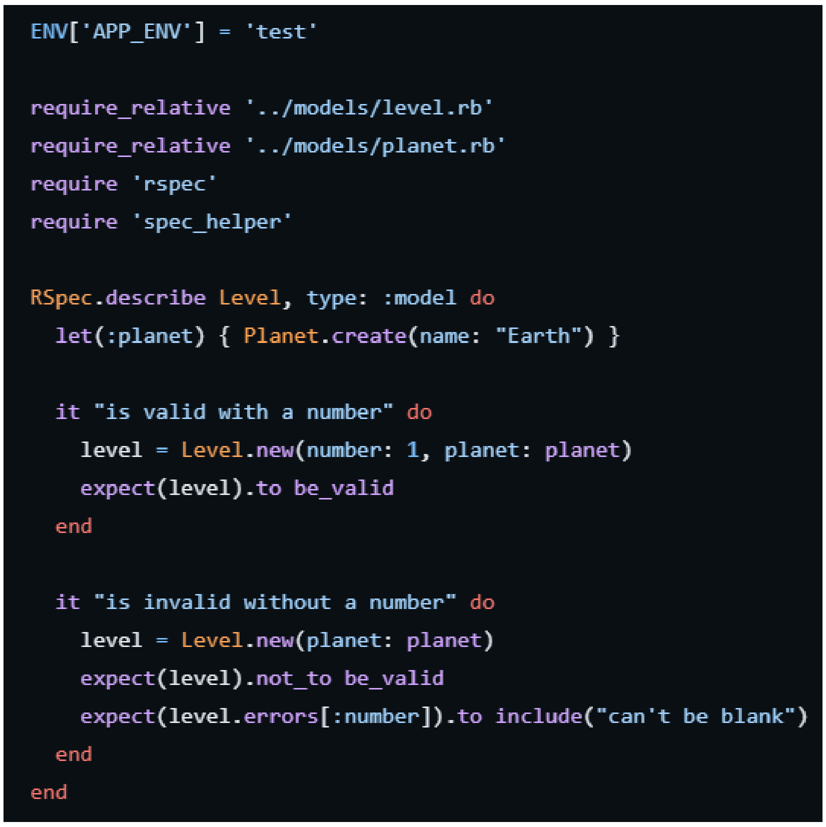
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/answer.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Answer, type: :model do
  let(:question) { Question.create(description: "Sample Question", level_id: 1) }

  it "is valid with a description and correct value" do
    expect(Answer.new(description: "Sample Answer", correct: false, question: question)).to be_valid
  end


end
```

En este caso, no cubrimos el caso en el que haya respuestas sin descripción, por lo tanto lo agregamos. Además agregamos una prueba de manera indirecta para el modelo Question, ya que tenemos una pregunta con una respuesta correcta, cuando intentamos agregar otra respuesta correcta, esa respuesta es inválida. De esta forma, el test quedo asi:



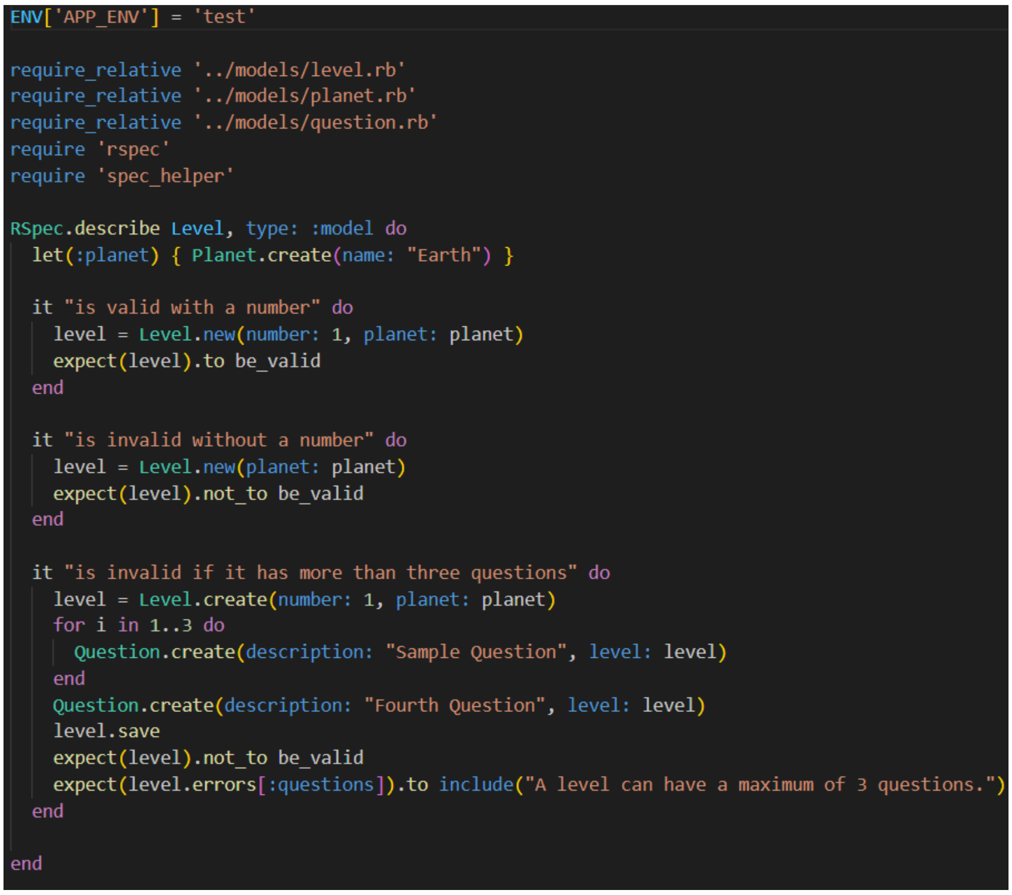
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/answer.rb'
require_relative '../models/question.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Answer, type: :model do
  let(:question) { Question.create(description: "Sample Question", level_id: 1) }

  it "is valid with a description and correct value" do
    answer = Answer.new(description: "Sample Answer", correct: false, question: question)
    expect(answer).to be_valid
  end

  it "is invalid without a description" do
    answer = Answer.new(description: nil, correct: false, question: question)
    expect(answer).not_to be_valid
  end

  it "allows only one correct answer per question" do
    Answer.create(description: "Correct Answer", correct: true, question: question)
    another_answer = Answer.new(description: "Another Answer", correct: true, question: question)
    expect(another_answer).not_to be_valid
  end
end
```

Modelo Level:



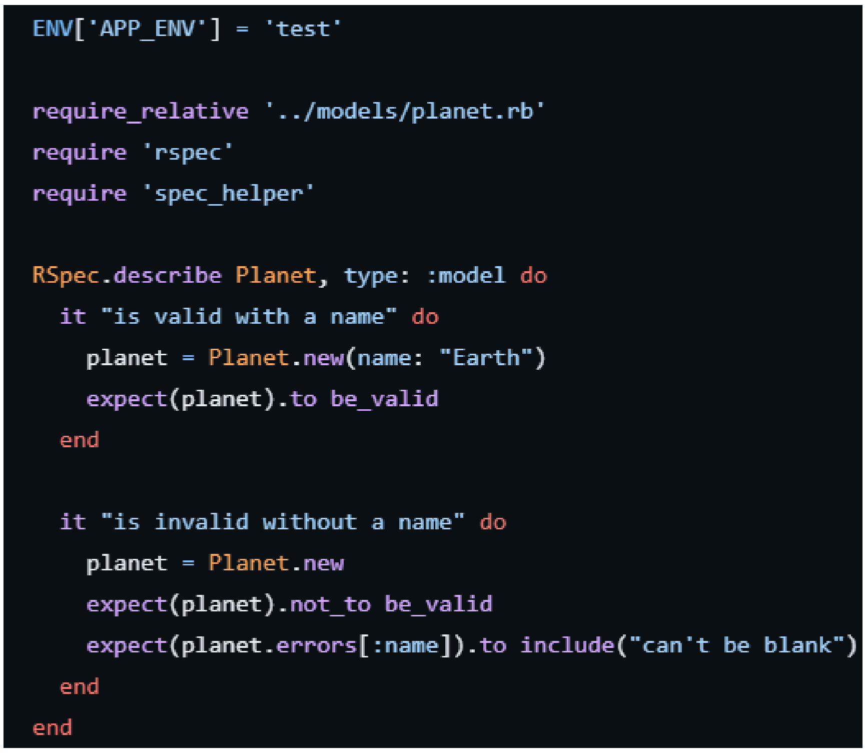
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/level.rb'
require_relative '../models/planet.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Level, type: :model do
  let(:planet) { Planet.create(name: "Earth") }

  it "is valid with a number" do
    level = Level.new(number: 1, planet: planet)
    expect(level).to be_valid
  end

  it "is invalid without a number" do
    level = Level.new(planet: planet)
    expect(level).not_to be_valid
    expect(level.errors[:number]).to include("can't be blank")
  end
end
```

En este modelo no se cubría el caso en el que un nivel tenga más de 3 preguntas (lo cual no es válido), por lo tanto lo agregamos, quedando el test de la siguiente manera:



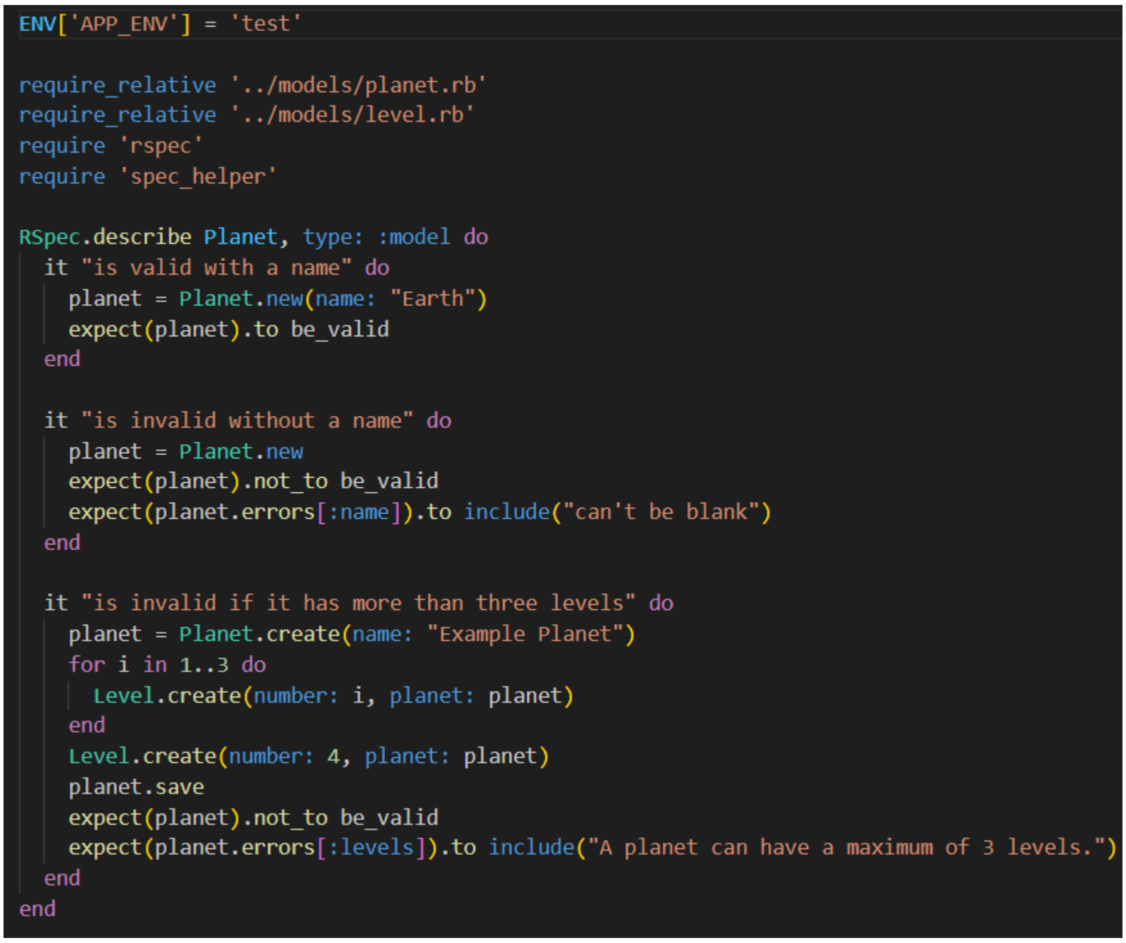
```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/level.rb'
require_relative '../models/planet.rb'
require_relative '../models/question.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Level, type: :model do
  let(:planet) { Planet.create(name: "Earth") }

  it "is valid with a number" do
    level = Level.new(number: 1, planet: planet)
    expect(level).to be_valid
  end

  it "is invalid without a number" do
    level = Level.new(planet: planet)
    expect(level).not_to be_valid
  end

  it "is invalid if it has more than three questions" do
    level = Level.create(number: 1, planet: planet)
    for i in 1..3 do
      Question.create(description: "Sample Question", level: level)
    end
    Question.create(description: "Fourth Question", level: level)
    level.save
    expect(level).not_to be_valid
    expect(level.errors[:questions]).to include("A level can have a maximum of 3 questions.")
  end

end
```

Modelo Planet:



```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/planet.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Planet, type: :model do
  it "is valid with a name" do
    planet = Planet.new(name: "Earth")
    expect(planet).to be_valid
  end

  it "is invalid without a name" do
    planet = Planet.new
    expect(planet).not_to be_valid
    expect(planet.errors[:name]).to include("can't be blank")
  end
end
```

Similar al modelo Level, no se cubría el caso en el que un planeta tenga más de 3 niveles (lo cual no es válido), por lo tanto agregamos este ítem y el test quedo asi:



```ruby
ENV['APP_ENV'] = 'test'

require_relative '../models/planet.rb'
require_relative '../models/level.rb'
require 'rspec'
require 'spec_helper'

RSpec.describe Planet, type: :model do
  it "is valid with a name" do
    planet = Planet.new(name: "Earth")
    expect(planet).to be_valid
  end

  it "is invalid without a name" do
    planet = Planet.new
    expect(planet).not_to be_valid
    expect(planet.errors[:name]).to include("can't be blank")
  end

  it "is invalid if it has more than three levels" do
    planet = Planet.create(name: "Example Planet")
    for i in 1..3 do
      Level.create(number: i, planet: planet)
    end
    Level.create(number: 4, planet: planet)
    planet.save
    expect(planet).not_to be_valid
    expect(planet.errors[:levels]).to include("A planet can have a maximum of 3 levels.")
  end
end
```