



RSpec Testing Framework Technical Documentation

What is RSpec

RSpec is a Behaviour-Driven Development (BDD) testing framework for Ruby on Rails that offers a readable and structured way to write test cases. It enables efficient testing across various components of a Ruby on rails app – controllers, models, views, routes, and helpers.

RSpec's key features:

- Easy to read and understand syntax
- Precise testing of various scenarios
- Comprehensive (unit, function, and integration) testing
- Allows custom and external testing tools integration

RSpec Setup:

This documentation focuses on `rspec-rails` library. `rspec-rails` integrates RSpec with Rails. It allows developers to replace the default Rails testing framework for more flexible testing.

1. Add `rspec-rails` to the Gemfile under the `:development` and `:test` group.

```
gem 'rspec-rails', '~> 6.0'
```

```
Gemfile x
portfolio_app > Gemfile
40
47 ∨ group :development, :test do
48   # See https://guides.rubyonrails.org/debugging_rails_applications.html#debugging
49   gem "debug", platforms: %i[ mri windows ]
50
51   # RSpec
52   gem 'rspec-rails', '~> 6.0'
53
54 end
```

2. Download and install the library to your project.

In your terminal, run

```
bundle install
```

3. Initialize RSpec and generate configuration files

```
rails generate rspec:install
```

The above command will create the following for you:

```
∨ spec
  rails_helper.rb
  spec_helper.rb
```

- a. spec/: Directory for all test files.
- b. spec/spec_helper.rb: Contains basic configuration for RSpec.
- c. spec/rails_helper.rb: Loads the Rails environment and additional setup specific.

How to use RSpec:

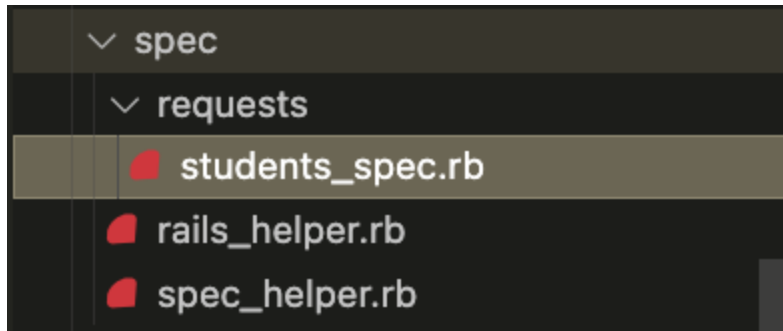
Example

```
rails generate rspec:request student
```

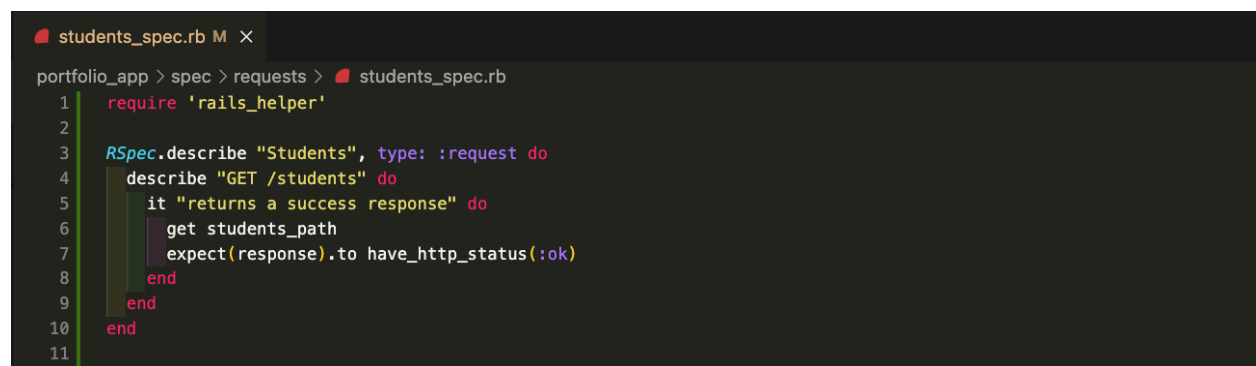
- rails generate - Used to create a boilerplate code for various MVC components.
- rspec:request - This tells Rails to generate request spec, which is used to test HTTP requests of a full stack Ruby app.
- student - This is the resource name where requests will be tested.

The command will generate the following file where all request tests to the student resource will be written.

`spec/requests/students_spec.rb`



It will also contain the basic structure of request tests.



RSpec structure:

```
RSpec.describe "Students", type: :request do
  // All request specs related to the student resource will be
  // written here.
```

```
end
```

Now let's create RSpec test and test it on a local machine.

The following test code checks the `GET /students/:id` endpoint. In total, it has 3 tests.

1. Initially, it creates a temporary Student object testing data.
2. Test #1: Verifies that the endpoint returns a 200 OK status with the correct student details when the student exists.
3. Test #2: Verifies that the endpoint returns a 404 Not Found status when the student does not exist.
4. Test #3: Ensures that both the status codes and response content behave as expected for valid and invalid requests.

```

RSpec.describe "Students", type: :request do

  describe "GET /students/:id" do
    context "when the student exists" do
      let!(:student) { Student.create!(first_name: "Bob", last_name: "John", school_email:
        "gordon@sudenver.edu", major: "Computer Science BS", graduation_date: "2025-05-15") }

      it "returns a 200 OK status" do
        get student_path(student)
        expect(response).to have_http_status(:ok)
      end

      it "includes the correct student details in the response body" do
        get student_path(student)
        expect(response.body).to include("Bob")
        expect(response.body).to include("John")
        expect(response.body).to include("gordon@sudenver.edu")
        expect(response.body).to include("Computer Science BS")
        expect(response.body).to include("2025-05-15")
      end
    end

    context "when the student does not exist" do
      it "returns a 404 Not Found status" do
        get student_path(id: 404)
        expect(response).to have_http_status(:not_found)
      end
    end
  end
end

```

Line-by-line test code explanation:

Test Set up:

- **describe** block: Groups tests for the GET /students/:id endpoint and defines what behavior is expected.
- **context** block: Specifies the scenario where the student exists
- **let!** block: Immediately creates a student record with attributes like name, email, major, and graduation date for use in the tests.

Test #1:

- **it** block: Tests that the endpoint returns a 200 OK status when a valid student ID is requested.
 - ◆ **get** call: Sends a GET request to fetch the student using the student's ID from the student_path helper.
 - ◆ **expect** status: Confirms that the response status is 200 OK.

Test #2:

- **it** block: Verifies that the response body contains the correct student details (name, email, major, graduation date).
 - ◆ **expect** body: Asserts that the response includes specific attribute values, ensuring the data matches the student created.

Test #3:

- **context** block: Tests the scenario where the requested student does not exist.
 - ◆ **it** block: Checks that the endpoint returns a 404 Not Found status when an invalid or non-existent student ID is used.


Now let's run our GET student by ID request test.

But, how to run RSpec tests?

Command	Function
<code>\$ rspec</code>	Used to Run all spec files
<code>\$ rspec spec/requests</code>	Used to Run all spec files in a single directory (<code>requests</code>)
<code>\$ rspec spec/requests/students_spec.rb</code>	Run a single spec file
<code>\$ rspec spec/requests/students_spec.rb:85</code>	Used to Run a single example from a spec file by line number (85)
<code>\$ rspec --help</code>	To see all options for running specs

For our example, we're going to use the single spec file command:

```
rspec spec/requests/students_spec.rb
```

```
PROBLEMS  OUTPUT  TERMINAL  PORTS  DEBUG CONSOLE   docker

# rspec spec/requests/students_spec.rb
...

Finished in 0.50587 seconds (files took 0.97752 seconds to load)
3 examples, 0 failures

#
```

Hooray! All three of our tests have passed!