

Test-driven Development no Rails

Começando seu projeto com o pé direito

2007, Nando Vieira – <http://simplesideias.com.br>

O que iremos ver?

```
slides = Array.new
```

```
slides << "O que é TDD?"
```

```
slides << "Por que testar?"
```

```
slides << "Um pequeno exemplo"
```

```
slides << "TDD no Rails"
```

```
slides << "Fixtures"
```

```
slides << "Testes unitários"
```

```
slides << "Testes funcionais"
```

```
slides << "Testes de integração"
```

```
slides << "Mocks & Stubs"
```

```
slides << "Dúvidas?"
```

O que é TDD?

- Test-driven Development ou Desenvolvimento Guiado por Testes
- Uma técnica de desenvolvimento de software
- Ficou conhecida como um aspecto do XP

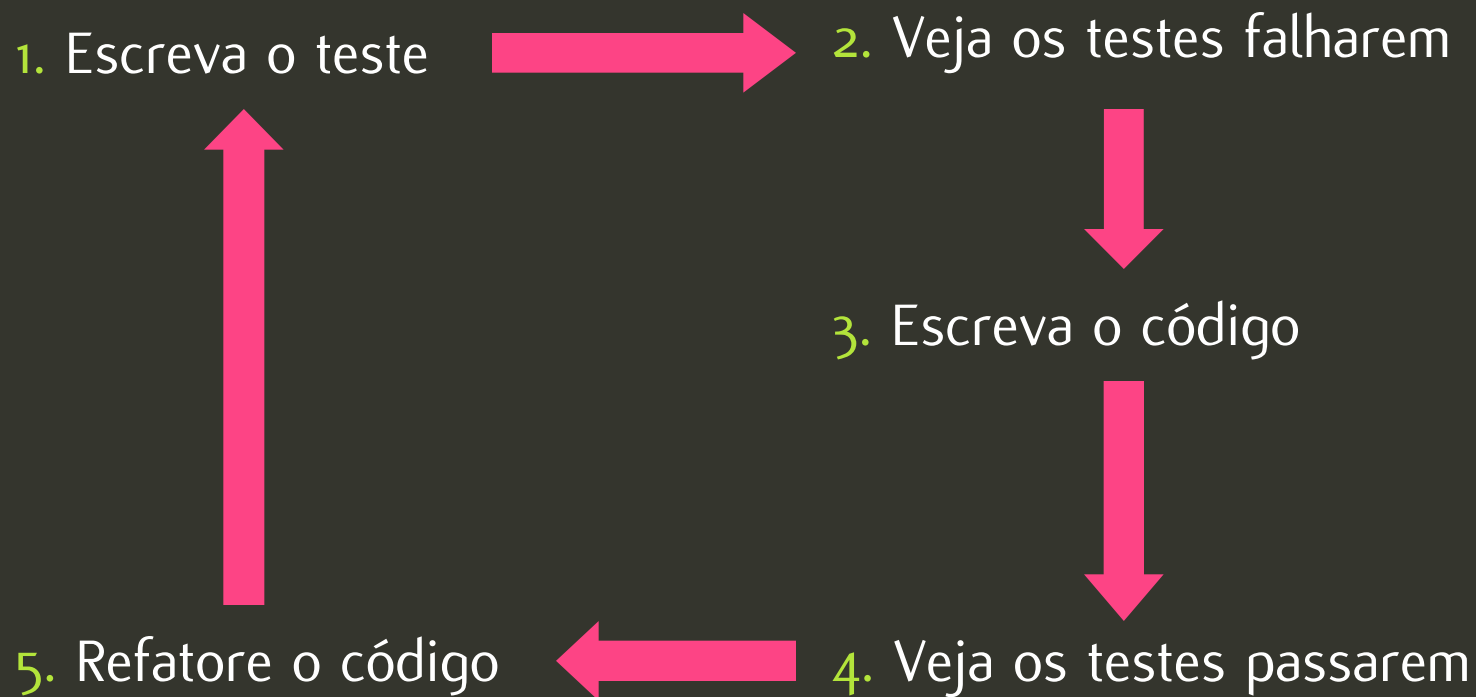
O que é TDD?

Kent Beck (TDD by Example, 2003):

1. Escreva um teste que falhe antes de escrever qualquer código
2. Elimine toda duplicação (refactoring)
3. Resposta rápida a pequenas mudanças
4. Escreva seus próprios testes

O que é TDD?

Processo:



Porque testar

“ Isso não me impediu de cometer a tresloucada loucura de publicar minha aplicação utilizando Rails 1.2.3 apenas um dia após o lançamento deles. Devo estar ficando doido mesmo.
Ou isso ou tenho testes.

Thiago Arrais – Motiro, no Google Groups Ruby-Br

Porque testar

Você:

- escreverá códigos **melhores**
- escreverá códigos **mais rapidamente**
- identificará erros **mais facilmente**
- não usará **F5 nunca mais!**

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

test_calculator.rb

```
class TestCalculator < Test::Unit::TestCase
  def setup
    @calc = Calculator.new
  end

  def teardown
    @calc = nil
  end

  def test_sum
    assert_equal(2, @calc.sum(1, 1), "1 + 1 = 2")
  end

  def test_subtract
    assert_equal(1, @calc.subtract(2, 1), "2 - 1 = 1")
  end

  def test_multiply
    assert_equal(4, @calc.multiply(2, 2), "2 x 2 = 4")
  end

  def test_divide
    assert_equal(2, @calc.sum(4, 2), "4 / 2 = 2")
    assert_raise(ZeroDivisionError) { @calc.divide(2, 0) }
  end
end
```

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

execute os testes:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

execute os testes: todos eles devem falhar

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
EEEE
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NameError: uninitialized constant TestCalculator::Calculator
```

```
test.rb:24:in `setup'
```

```
...
```

```
4 tests, 0 assertions, 0 failures, 4 errors
```

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

execute os testes: todos eles devem falhar

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
EEEE
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NameError: uninitialized constant TestCalculator::Calculator
```

```
test.rb:24:in `setup'
```

```
...
```

```
4 tests, 0 assertions, 0 failures, 4 errors
```



4 erros: nosso código não passou no teste

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o primeiro
método:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o primeiro método: rode os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
EEE.
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NoMethodError: undefined method `divide' for #<Calculator:
0x2e2069c>
```

```
test.rb:44:in `test_divide'
```

```
...
```

```
4 tests, 1 assertions, 0 failures, 3 errors
```

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o primeiro método: rode os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
EEE.
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NoMethodError: undefined method `divide' for #<Calculator:
0x2e2069c>
```

```
test.rb:44:in `test_divide'
```

```
...
```

```
4 tests, 1 assertions, 0 failures, 3 errors
```



1 asserção: nosso código passou no teste

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n2 - n1
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o método
seguinte:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n2 - n1
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o método seguinte: rode os testes

```
$~ ruby test_calculator.rb
```

Loaded suite test

Started

EF.E

Finished in 0.0 seconds.

1) Failure:

test_subtract(TestCalculator) [test.rb:35]:

2 - 1 = 1.

<1> expected but was

<-1>.

...

4 tests, 1 assertions, 1 failures, 2 errors

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n2 - n1
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o método seguinte: rode os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
EF.E
```

```
Finished in 0.0 seconds.
```

```
1) Failure:
```

```
test_subtract(TestCalculator) [test.rb:35]:
```

```
2 - 1 = 1.
```

```
<1> expected but was
```

```
<-1>.
```

```
...
```

```
4 tests, 1 assertions, 1 failures, 2 errors
```



1 falha: nosso código não passou no teste

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n2 - n1
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o método seguinte: rode os testes

```
$~ ruby test_calculator.rb
```

Loaded suite test

Started

EF.E

Finished in 0.0 seconds.

1) Failure:

test_subtract(TestCalculator) [test.rb:35]:

2 - 1 = 1.

<1> expected but was

<-1>.

...

4 tests, 1 assertions, 1 failures, 2 errors



1 falha: nosso código não passou no teste

devia ser $n1 - n2$

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

corrija o método que falhou:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

corrija o método que falhou: rode os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
E..E
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NoMethodError: undefined method `divide' for #<Calculator:
0x2e2069c>
```

```
test.rb:44:in `test_divide'
```

```
...
```

```
4 tests, 2 assertions, 0 failures, 2 errors
```

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

corrija o método que falhou: rode os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
E..E
```

```
Finished in 0.0 seconds.
```

```
1) Error:
```

```
test_divide(TestCalculator):
```

```
NoMethodError: undefined method `divide' for #<Calculator:
0x2e2069c>
```

```
test.rb:44:in `test_divide'
```

```
...
```

```
4 tests, 2 assertions, 0 failures, 2 errors
```



2 asserções: nosso código passou no teste

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

continue o processo:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

continue o processo: execute os testes

```
$~ ruby test_calculator.rb
```

...

4 tests, 3 assertions, 0 failures, 1 errors

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

continue o processo: execute os testes

```
$~ ruby test_calculator.rb
```

...

4 tests, 3 assertions, 0 failures, 1 errors



só mais um: ufa!

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o último método:

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o último método: execute os testes

```
$~ ruby test_calculator.rb
```

Loaded suite test

Started

....

Finished in 0.0 seconds.

4 tests, 5 assertions, 0 failures, 0 errors

Um pequeno exemplo

calculator.rb

```
class Calculator
  def sum(n1, n2)
    n1 + n2
  end

  def subtract(n1, n2)
    n1 - n2
  end

  def multiply(n1, n2)
    n1 * n2
  end

  def divide(n1, n2)
    n1 / n2
  end
end
```

escreva o último método: execute os testes

```
$~ ruby test_calculator.rb
```

```
Loaded suite test
```

```
Started
```

```
....
```

```
Finished in 0.0 seconds.
```

```
4 tests, 5 assertions, 0 failures, 0 errors
```



5 asserções, nenhuma falha/erro: código perfeito!

TDD no Rails

- Testar uma aplicação no Rails é muito simples
- Os testes ficam no diretório `test`
- Os arquivos de testes são criados para você
- Apenas um comando: `rake test`
- Dados de testes: `fixtures`
- Testes unitários, funcionais, integração e performance
- Mocks & Stubs

Fixtures

- Conteúdo inicial do modelo
- Ficam sob o diretório test/fixtures
- SQL, CSV ou YAML

Modelo Artist



Tabela artists



Fixture artists.yml

Fixtures

test/fixtures/artists.yml

nofx:

id: 1

name: NOFX

url: http://nofxofficialwebsite.com

created_at: <%= Time.now.strftime(:db) %>

millencolin:

id: 2

name: Millencolin

url: http://millencolin.com

created_at: <%= 3.days.ago.strftime(:db) %>

Testes unitários

- Testes unitários validam modelos
- Ficam sob o diretório `test/unit`
- Arquivo gerado pelo Rails: `<model>_test.rb`

`script/generate model Artist`



`test/unit/artist_test.rb`

Testes unitários

test/unit/artist_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'
```

```
class ArtistTest < Test::Unit::TestCase
```

```
  fixtures :artists
```

```
  # Replace this with your real tests.
```

```
  def test_truth
```

```
    assert true
```

```
  end
```

```
end
```

Testes unitários

test/unit/artist_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'
```

```
class ArtistTest < Test::Unit::TestCase
```

```
  fixtures :artists
```

```
  def test_should_require_name
```

```
    artist = create(:name => nil)
```

```
    assert_not_nil artist.errors.on(:name), ":name should have had an error"
```

```
    assert !artist.valid?, "artist should be invalid"
```

```
  end
```

```
  private
```

```
  def create(options={})
```

```
    Artist.create({
```

```
      :name => 'Death Cab For Cutie',
```

```
      :url => 'http://deathcabforcutie.com'
```

```
    }).merge(options)
```

```
  end
```

```
end
```

Testes unitários

app/models/artist.rb

```
class Artist < ActiveRecord::Base  
end
```

execute os testes:

Testes unitários

app/models/artist.rb

```
class Artist < ActiveRecord::Base  
end
```

execute os testes: rake test:units

```
$~ rake test:units  
Loaded suite test  
Started  
F  
Finished in 0.094 seconds.
```

1) Failure:

```
test_should_require_name(ArtistTest) [./test/  
unit/artist_test.rb:8]:  
:name should have had an error.  
<nil> expected to not be nil.
```

1 tests, 1 assertions, 1 failures, 0 errors

Testes unitários

```
app/models/artist.rb
```

```
class Artist < ActiveRecord::Base  
end
```

execute os testes: rake test:units

```
$~ rake test:units  
Loaded suite test  
Started  
F  
Finished in 0.094 seconds.
```

1) Failure:

```
test_should_require_name(ArtistTest) [./test/  
unit/artist_test.rb:8]:  
:name should have had an error.  
<nil> expected to not be nil.
```

1 tests, 1 assertions, 1 failures, 0 errors



o teste falhou: era esperado!

Testes unitários

app/models/artist.rb

```
class Artist < ActiveRecord::Base  
  validates_presence_of :name  
end
```

adicione a validação:

Testes unitários

app/models/artist.rb

```
class Artist < ActiveRecord::Base
  validates_presence_of :name
end
```

adicione a validação: execute os testes

```
$~ rake test:units
```

```
Loaded suite test
```

```
Started
```

```
.
```

```
Finished in 0.062 seconds.
```

```
1 tests, 2 assertions, 0 failures, 0 errors
```

Testes unitários

app/models/artist.rb

```
class Artist < ActiveRecord::Base
  validates_presence_of :name
end
```

adicione a validação: execute os testes

```
$~ rake test:units
```

```
Loaded suite test
```

```
Started
```

```
.
```

```
Finished in 0.062 seconds.
```

1 tests, 2 assertions, 0 failures, 0 errors



nenhuma falha: voila!

Testes funcionais

- Testes funcionais validam controles
- Ficam sob o diretório `test/functional`
- Template e/ou requisição
- Arquivo gerado pelo Rails: `<controller>_test.rb`

`script/generate controller artists`



`test/functional/artists_controller_test.rb`

Testes funcionais

```
test/functional/artists_controller_test.rb
```

```
require File.dirname(__FILE__) + '/../test_helper'
require 'artists_controller'

# Re-raise errors caught by the controller.
class ArtistsController; def rescue_action(e) raise e end; end

class ArtistsControllerTest < Test::Unit::TestCase
  def setup
    @controller = ArtistsController.new
    @request     = ActionController::TestRequest.new
    @response    = ActionController::TestResponse.new
  end

  # Replace this with your real tests.
  def test_truth
    assert true
  end
end
```

Testes funcionais

```
test/functional/artists_controller_test.rb
```

...

```
class ArtistsControllerTest < Test::Unit::TestCase
  def setup
    @controller = ArtistsController.new
    @request     = ActionController::TestRequest.new
    @response    = ActionController::TestResponse.new
  end

  def test_index_page
    get index_path
    assert_response :success
    assert_template 'index'

    assert_select 'h1', 'Bands that sound like...'
    assert_select 'form', {:count => 1, :method} => 'get' do
      assert_select 'input#name', :count => 1
      assert_select 'input[type=submit]', :count => 1
    end
  end
end
```

Testes funcionais

```
test/functional/artists_controller_test.rb
```

...

```
class ArtistsControllerTest < Test::Unit::TestCase
```

```
  def setup
```

```
    @controller = ArtistsController.new
```

```
    @request = ActionController::TestRequest.new
```

```
    @response = ActionController::TestResponse.new
```

```
  end
```

```
  def test_index_page
```

```
    get index_path
```

```
    assert_response :success
```

```
    assert_template 'index'
```

← requisição: verificando a resposta

```
    assert_select 'h1', 'Bands that sound like...'
```

```
    assert_select 'form', { :count => 1, :method => 'get' } do
```

```
      assert_select 'input#name', :count => 1
```

```
      assert_select 'input[type=submit]', :count => 1
```

← template: verificando o markup

```
    end
```

```
  end
```

```
end
```

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

execute os testes:

```
1) Error:
```

```
test_index_page(ArtistsControllerTest):
```

```
ActionController::UnknownAction: No action responded to index
```

```
...
```

```
./test/functional/artists_controller_test.rb:15:in `test_index_page'
```

```
1 tests, 0 assertions, 0 failures, 1 errors
```

Testes funcionais

```
~$ rake test:functionals
```

execute os testes: eles devem falhar

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

```
1) Error:
```

```
test_index_page(ArtistsControllerTest):
```

```
ActionController::UnknownAction: No action responded to index
```

```
...
```

```
./test/functional/artists_controller_test.rb:15:in `test_index_page'
```

```
1 tests, 0 assertions, 0 failures, 1 errors
```

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

```
1) Error:
```

```
test_index_page(ArtistsControllerTest):
```

```
ActionController::UnknownAction: No action responded to index
```

```
...
```

```
./test/functional/artists_controller_test.rb:15:in `test_index_page'
```

```
1 tests, 0 assertions, 0 failures, 1 errors
```



1 erro: ele já era esperado

execute os testes: eles devem falhar

Testes funcionais

```
~$ rake test:functionals
```

execute os testes: eles devem falhar

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

```
1) Error:
```

action: nós ainda não a criamos

```
test_index_page(ArtistsControllerTest):
```



```
ActionController::UnknownAction: No action responded to index
```

```
...
```

```
./test/functional/artists_controller_test.rb:15:in `test_index_page'
```

```
1 tests, 0 assertions, 0 failures, 1 errors
```



1 erro: ele já era esperado

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

crie o template index.rhtml:

1) Failure:

```
test_index_page(ArtistsControllerTest)
```

```
[selector_assertions.rb:281:in `assert_select'
```

```
./test/functional/artists_controller_test.rb:19:in `test_index_page']:
```

```
Expected at least 1 elements, found 0.
```

```
<false> is not true.
```

```
1 tests, 3 assertions, 1 failures, 0 errors
```

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

crie o template index.rhtml: execute os testes

1) Failure:

```
test_index_page(ArtistsControllerTest)
```

```
[selector_assertions.rb:281:in `assert_select'
```

```
./test/functional/artists_controller_test.rb:19:in `test_index_page']:
```

```
Expected at least 1 elements, found 0.
```

```
<false> is not true.
```

```
1 tests, 3 assertions, 1 failures, 0 errors
```

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

1) Failure:

```
test_index_page(ArtistsControllerTest)
```

```
[selector_assertions.rb:281:in `assert_select'
```

```
./test/functional/artists_controller_test.rb:19:in `test_index_page']:
```

```
Expected at least 1 elements, found 0.
```

```
<false> is not true.
```

```
1 tests, 3 assertions, 1 failures, 0 errors
```

crie o template index.rhtml: execute os testes

← 1 erro: ele já era esperado

Testes funcionais

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
E
```

```
Finished in 0.094 seconds.
```

crie o template index.rhtml: execute os testes

1) Failure: **template: nenhum código HTML**

```
test_index_page(ArtistsControllerTest)
```



```
[selector_assertions.rb:281:in `assert_select'
```

```
./test/functional/artists_controller_test.rb:19:in `test_index_page']:
```

```
Expected at least 1 elements, found 0.
```

```
<false> is not true.
```

```
1 tests, 3 assertions, 1 failures, 0 errors
```



1 erro: ele já era esperado

Testes funcionais

app/views/artists/index.rhtml

```
<h1>Bands that sound like...</h1>  
<% form_tag 'view', {:method => :get} do %>  
  <p>  
    <%= text_field_tag :name %>  
    <%= submit_tag 'View' %>  
  </p>  
<% end %>
```

código HTML:

Testes funcionais

```
app/views/artists/index.rhtml
```

```
<h1>Bands that sound like...</h1>  
<% form_tag 'view', {:method => :get} do %>  
  <p>  
    <%= text_field_tag :name %>  
    <%= submit_tag 'View' %>  
  </p>  
<% end %>
```

código HTML: execute os testes

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
.
```

```
Finished in 0.094 seconds.
```

```
1 tests, 9 assertions, 0 failures, 0 errors
```

Testes funcionais

```
app/views/artists/index.rhtml
```

```
<h1>Bands that sound like...</h1>  
<% form_tag 'view', { :method => :get } do %>  
  <p>  
    <%= text_field_tag :name %>  
    <%= submit_tag 'View' %>  
  </p>  
<% end %>
```

código HTML: execute os testes

```
~$ rake test:functionals
```

```
Loaded suite test
```

```
Started
```

```
.
```

```
Finished in 0.094 seconds.
```

```
1 tests, 9 assertions, 0 failures, 0 errors
```



yep: tudo certo!



erros: nenhum para contar história

Testes de integração

- Testes de integração validam (duh) a integração entre diferentes controllers
- Ficam sob o diretório `test/integration`
- Rails não gera arquivo automaticamente: `você escolhe!`

```
script/generate integration_test artist_stories
```



```
test/integration/artist_stories_test.rb
```

Testes de integração

test/integration/artist_stories_test.rb

```
require File.dirname(__FILE__) + '/../test_helper'
```

```
class ArtistStoriesTest < ActionController::IntegrationTest
```

```
  # fixtures :your, :models
```

```
  # Replace this with your real tests.
```

```
  def test_truth
```

```
    assert true
```

```
  end
```

```
end
```

Testes de integração

```
test/integration/artist_stories_test.rb
```

```
require File.dirname(__FILE__) + '/../test_helper'

class ArtistStoriesTest < ActionController::IntegrationTest
  fixtures :artists, :albums

  def test_user_viewing_artist_and_album
    # user accessing index page
    get "/"
    assert_response :success
    assert_template "index"

    # searching for an artist
    get artist_search_path(artists(:millencolin).slug)
    assert_response :redirect
    assert_redirected_to "/artists/#{artists(:millencolin).slug}"

    # viewing album "life on a plate"
    get album_path(albums(:life_on_a_plate))
    assert_response :success
    assert_template "view"
  end
end
```

Testes de integração

```
test/integration/artist_stories_test.rb
```

```
require File.dirname(__FILE__) + '/../test_helper'
```

```
class ArtistStoriesTest < ActionController::IntegrationTest  
  fixtures :artists, :albums
```

```
  def test_user_viewing_artist_and_album
```

```
    # user accessing index page
```

```
    get "/"
```

```
    assert_response :success
```

```
    assert_template "index"
```

← controller: /

```
    # searching for an artist
```

```
    get artist_search_path(artists(:millencolin).slug)
```

```
    assert_response :redirect
```

```
    assert_redirected_to "/artists/#{artists(:millencolin).slug}"
```

← controller: /artists/search

```
    # viewing album "life on a plate"
```

```
    get album_path(albums(:life_on_a_plate))
```

```
    assert_response :success
```

```
    assert_template "view"
```

← controller: /albums/

```
  end
```

```
end
```

Testes de integração

- Testes funcionais em um nível mais alto: DSL
- Leitura mais fácil, impossível!
- Use o método `open_session`

```
open_session do |session|  
  # Do everything you need!  
end
```

Testes de integração

```
test/integration/artist_stories_test.rb
```

```
class ArtistStoriesTest < ActionController::IntegrationTest
  fixtures :artists, :albums
```

```
  def test_artist_story
    artist = artists(:millencolin)

    user = new_session
    user.search_artist artist
    user.view_album artist.albums.first
  end
```

```
private
  def new_session
    open_session do |session|
      def session.search_artist(artist)
        get search_artist(artist.name)
        assert_response :redirect
        assert_redirected_to artist_path(artist.slug)
      end

      def session.view_album(album)
        get album_path(album.slug)
        assert_response :success
        assert_template 'view'
      end
    end
  end
end
```

Testes de integração

```
test/integration/artist_stories_test.rb
```

```
class ArtistStoriesTest < ActionController::IntegrationTest
  fixtures :artists, :albums
```

```
  def test_artist_story
    artist = artists(:millencolin)
```

```
    user = new_session
    user.search_artist artist
    user.view_album artist.albums.first
  end
```



```
user = new_session
user.search_artist artist
user.view_album artist.albums.first
```

```
private
```

```
  def new_session
```

```
    open_session do |session|
```

```
      def session.search_artist(artist)
```

```
        get search_artist(artist.name)
```

```
        assert_response :redirect
```

```
        assert_redirected_to artist_path(artist.slug)
```

```
      end
```

```
      def session.view_album(album)
```

```
        get album_path(album.slug)
```

```
        assert_response :success
```

```
        assert_template 'view'
```

```
      end
```

```
    end
```

```
  end
```

```
end
```

Testes de integração

```
test/integration/artist_stories_test.rb
```

```
class ArtistStoriesTest < ActionController::IntegrationTest
  fixtures :artists, :albums
```

```
  def test_artist_story
    artist = artists(:millencolin)
```

```
    user = new_session
    user.search_artist artist
    user.view_album artist.albums.first
  end
```

```
user = new_session
user.search_artist artist
user.view_album artist.albums.first
```



```
private
  def new_session
    open_session do |session|
      def session.search_artist(artist)
        get search_artist(artist.name)
        assert_response :redirect
        assert_redirected_to artist_path(artist.slug)
      end
```



melhor, impossível!

```
      def session.view_album(album)
        get album_path(album.slug)
        assert_response :success
        assert_template 'view'
      end
    end
  end
end
```

Mocks & Stubs

- Códigos que eliminam o acesso a um recurso
- Ficam sob o diretório `test/mocks/test`
- Deve ter o mesmo nome do arquivo e estrutura da classe/módulo que você quer substituir

`RAILS_ROOT/lib/launch_missile.rb`



`test/mocks/<RAILS_ENV>/launch_missile.rb`

Mocks & Stubs

lib/feed.rb

```
require 'open-uri'
```

```
class Feed
```

```
  def load(url)
```

```
    open URI.parse(url).read
```

```
  end
```

```
end
```

Mocks & Stubs

lib/feed.rb

```
require 'open-uri'
```

```
class Feed
```

```
  def load(url)
```

```
    open URI.parse(url).read
```

```
  end
```

```
end
```



não é o ideal:

- a) pode ser lento;
- b) pode não estar disponível;
- c) imagine se fosse transação de pagamento!

Mocks & Stubs

```
test/mocks/test/feed.rb
```

```
require 'open-uri'
```

```
class Feed
```

```
  def load(url)
```

```
    File.new("#{RAILS_ROOT}/test/fixtures/feed.xml").read
```

```
  end
```

```
end
```

Mocks & Stubs

```
test/mocks/test/feed.rb
```

```
require 'open-uri'
```

```
class Feed
```

```
  def load(url)
```

```
    File.new("#{RAILS_ROOT}/test/fixtures/feed.xml").read
```

```
  end
```

```
end
```



muito mais fácil e rápido: faça quantos pagamentos quiser!

Dúvidas?