<!--Ingeniería de Software IIC2143-->

Ayudantía Testing {

<Por="Vicente Thomas, Cristobal Pérez-Cotapos, Nicolás Barría y Benito Palacios"/>

Contenidos

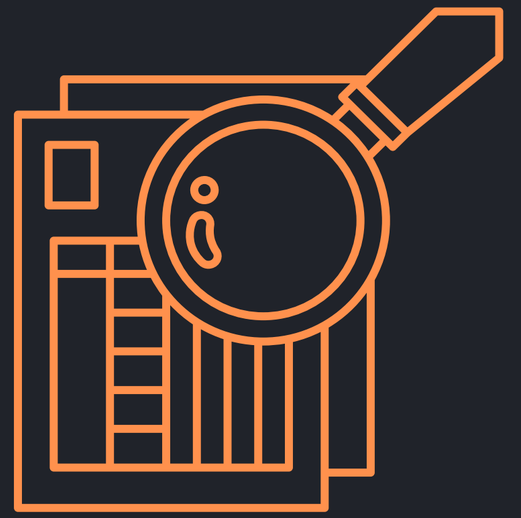
¿Qué es el testing?

> El testing en la programación se refiere al proceso de ejecutar un programa con la intención de encontrar fallas en su funcionamiento, con el objetivo de asegurar su calidad y corregir los problemas identificados.

}

Importancia del testing

- Garantizar la calidad de un software antes de su lanzamiento al mercado.
- Detección de errores y fallos en el funcionamiento del programa antes de que los usuarios finales los experimenten.
- Ayuda a asegurar que el software cumpla con los requisitos y especificaciones definidas.
- Reduce el riesgo de costosas correcciones y re-trabajos en el futuro.



}

Tipos de testing

UNIT

Proceso de probar individualmente cada unidad o componente aislado de un programa para verificar su correcto funcionamiento.

INTEGRATION

Proceso de verificar el correcto funcionamiento de la interacción entre diferentes componentes o módulos de un programa.

SYSTEM

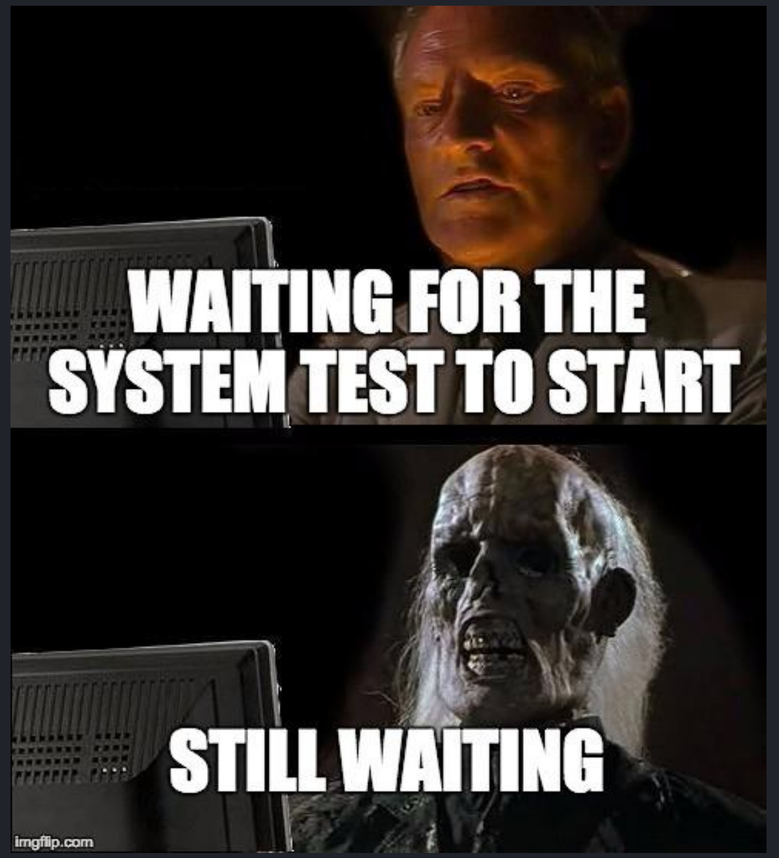
Proceso de probar todo el sistema completo, desde su interfaz hasta su funcionalidad, para asegurar que cumpla con los requisitos y especificaciones

}

Memes...


When your implementation passed integration testing but all unit test cases failed..
9gag


UNIT TESTS PASSING
NO INTEGRATION TESTS
imgflip.com


WAITING FOR THE SYSTEM TEST TO START
STILL WAITING
imgflip.com

}

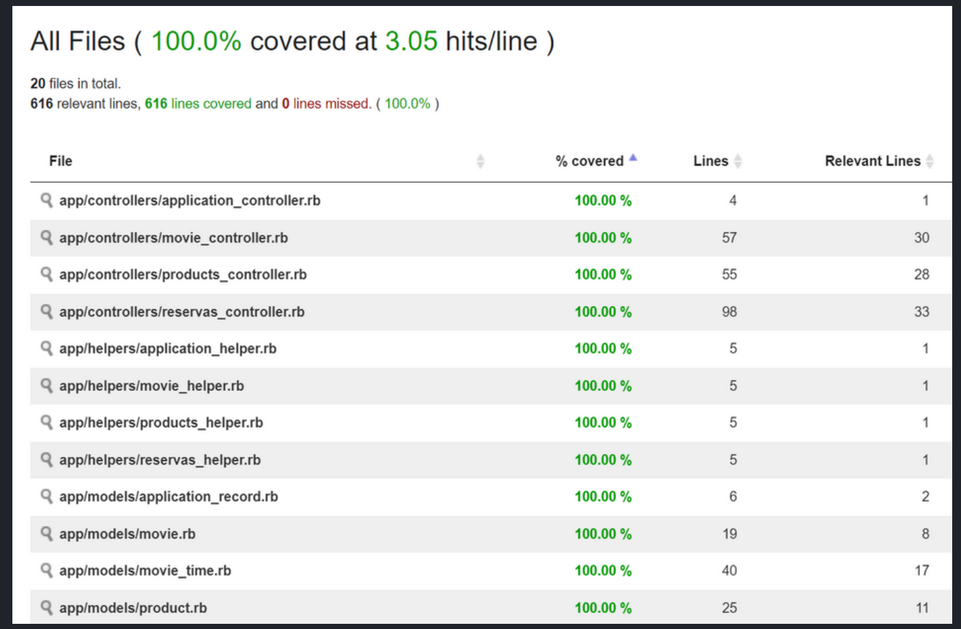
Testing en Rails

Minitest

Gema de testing incluida en Rails que permite escribir pruebas de manera sencilla y eficiente para testing de unidad, integración y aceptación. (FÁCIL Y FLEXIBLE)

Simple COV

Gema utilizada para medir la cobertura de código en Rails y obtener un informe detallado sobre qué líneas de código han sido ejecutadas durante las pruebas de testing.



All Files (100.0% covered at 3.05 hits/line)

20 files in total.
616 relevant lines, **616** lines covered and **0** lines missed. (100.0%)

File	% covered	Lines	Relevant Lines
app/controllers/application_controller.rb	100.00 %	4	1
app/controllers/movie_controller.rb	100.00 %	57	30
app/controllers/products_controller.rb	100.00 %	55	28
app/controllers/reservas_controller.rb	100.00 %	98	33
app/helpers/application_helper.rb	100.00 %	5	1
app/helpers/movie_helper.rb	100.00 %	5	1
app/helpers/products_helper.rb	100.00 %	5	1
app/helpers/reservas_helper.rb	100.00 %	5	1
app/models/application_record.rb	100.00 %	6	2
app/models/movie.rb	100.00 %	19	8
app/models/movie_time.rb	100.00 %	40	17
app/models/product.rb	100.00 %	25	11

}

Minitest

- Integrado de manera nativa en Rails. Al generar los modelos, automáticamente se generan carpetas de test.
- Una de las funciones más importantes es la ***assertion*** (afirmación). Son declaraciones que se buscan verificar para ver si se cumplen o no.

Assert más comúnes:

- assert - Verifica si una expresión es verdadera.
- assert_equal - Verifica si dos valores son iguales.
- assert_nil - Verifica si un valor es nulo.
- assert_includes - Verifica si un valor está incluido en una colección.
- assert_raises - Verifica si se genera una excepción.

Tests

Estructura

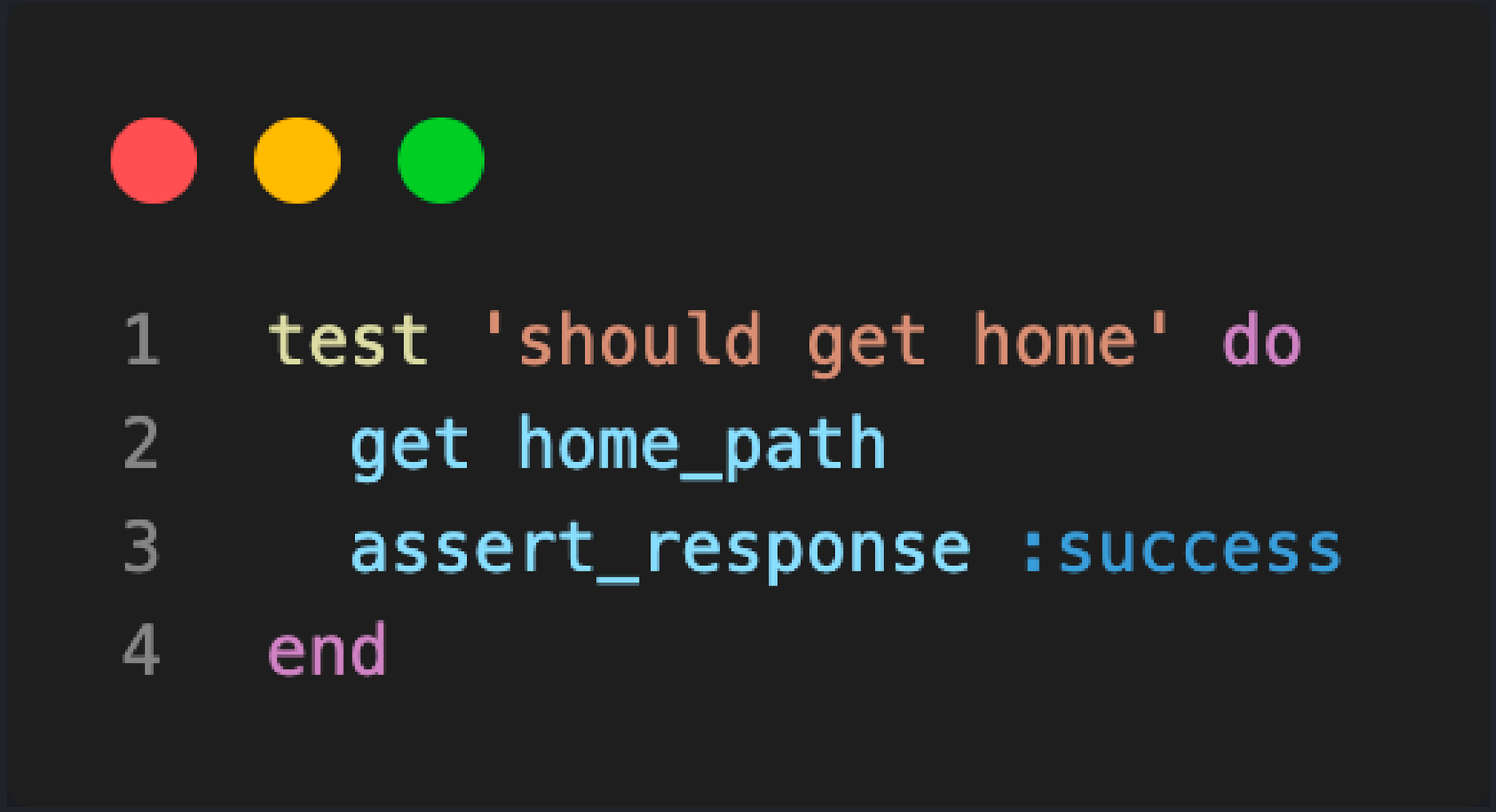
```ruby
class ExampleTest < ActionDispatch::IntegrationTest
  def setup
    'create all objects needed for tests'
  end

  def teardown
    'destroy objects created in setup'
  end

  test 'description' do
    'do something'
    'assert something'
  end
```

}

Ejemplo Controlador

```ruby
class MovieControllerTest < ActionDispatch::IntegrationTest
  def setup
    @movie_params = {
      title: 'Test Movie',
      age_restriction: true,
      language: 'Español'
    }
    @movie_time_params = { movie_time: {
      branch: 'Santiago',
      time: 'TANDA',
      date_start: Date.new(2022, 10, 19),
      date_end: Date.new(2022, 10, 31),
      room: 1
    } }
  end
```

Ejemplo Controlador



```ruby
test 'should get home' do
  get home_path
  assert_response :success
end
```

Ejemplo Controlador

```ruby
test 'should get list_by_date with age restriction' do
    get movies_by_date_path(
      date: Date.new(2022, 10, 19),
      language: 'Español',
      branch: 'Santiago',
      age_restriction: '1'
    )
    assert_response :success
  end
```

Ejemplo Controlador

```
1  test 'should create a Movie with valid attributes' do
2    assert_difference('Movie.count', 1) do
3      post create_movie_path, params: @movie_params
4    end
5    assert_redirected_to movie_new_path
6  end
```

Ejemplo Controlador

```ruby
test 'should not create a Movie with invalid title' do
  @movie_params[:title] = nil
  assert_no_difference('Movie.count') do
    post create_movie_path, params: @movie_params
  end
  assert_redirected_to movie_new_path
end
```

Ejemplo Modelo

```ruby
class MovieTest < ActiveSupport::TestCase
  def setup
    @movie = Movie.new(
      title: 'Example Movie',
      age_restriction: true,
      language: 'Español'
    )
  end
```

Ejemplo Modelo

```
1   test 'A movie is valid with valid params' do
2     assert @movie.valid?
3   end
4
```

Ejemplo Modelo

```ruby
test 'A movie is invalid if age_restriction is nil' do
  @movie.age_restriction = nil
  assert @movie.invalid?
end
```

Simple COV - Revisar Coverage

Pasos a seguir:

1. Agregar la gema al gemfile
2. Haz bundle install
3. Requerir simplecov en el helper y hacerle start
4. Ejecuta los test con rails test
5. Abrir coverage/index.html en navegador

```ruby
# Gemfile
gem 'simplecov', require: false, group: :test
```

```ruby
# test/test_helper.rb
require 'rails/test_help'
require 'simplecov'
SimpleCov.start
```

<!--Ingeniería de Software IIC2143-->

Ejemplo{

<manos_a_la_obra=True>

}