

Selenium



Barbeito Rodríguez, Alejandro

Índice

1. Uso de pruebas automáticas
2. Selenium
3. JUnit + Selenium
4. CodeceptJS + WebDriverIO
5. Conclusión

1. Uso de pruebas automáticas

Realizar test o pruebas a nuestras aplicaciones **no es una parte opcional** del desarrollo y es normal que a medida que añadamos nuevas características a nuestro proyecto las probemos en el momento, por lo que también tendría que ser normal realizar todas las pruebas anteriores para comprobar que todo el funcionamiento sigue correcto y que las partes nuevas no interfieran con ya implementadas. Hacer todo este trabajo con cierta frecuencia puede **requerir bastante tiempo**, por no decir que es posible cometer errores por el tedio de la repetición y las prisas. Es por ello que en la actualidad se crearon herramientas especializadas en la realización de estas tareas de forma automatizada y que se adapten a distintos campos.

1. Uso de pruebas automáticas

Como se acaba de mencionar existe mucho software que pueda adaptarse a nuestras necesidades, pero en este documento nos centraremos en 2 frameworks que pueden emplear junto **Selenium**:

JUnit → es una herramienta que permite extender las pruebas grabadas con Selenium IDE para que puedan ser ejecutadas con cualquier navegador.

CodeceptJS → Nos permite hacer uso de WebDriverIO (entre otras cosas), que nos permite realizar pruebas de Selenium mediante código javascript.

2. Selenium



Selenium es un **entorno de pruebas de software** para aplicaciones **basadas en la web**. Selenium provee una herramienta de **grabar/reproducir** para crear pruebas sin usar un lenguaje de scripting para pruebas (Selenium IDE). Incluye también un lenguaje específico de dominio para pruebas (Selenese) para escribir pruebas en un amplio número de lenguajes de programación populares incluyendo Java, C#, Ruby, Groovy, Perl, Php y Python (aunque también hay maneras de realizar estas pruebas mediante lenguaje javascript como vamos a comprobar). Las pruebas pueden ejecutarse entonces usando la mayoría de los navegadores web modernos en diferentes sistemas operativos como Windows, Linux y OSX.

3. JUnit + Selenium



JUnit es un framework Java que permite la realización de la ejecución de clases de manera controlada, para poder comprobar que los métodos realizan su cometido de forma correcta.

Este ya viene integrado en IDEs como **Eclipse** o **Netbeans** por lo que no es necesario realizar ningún tipo de instalación adicional.

3. JUnit + Selenium

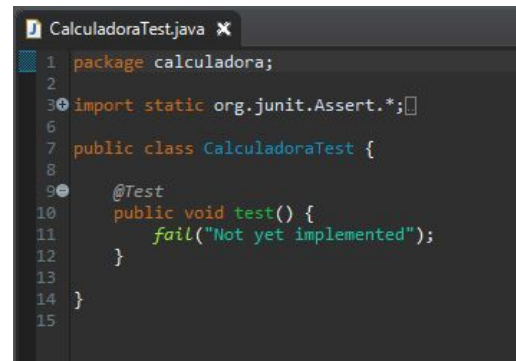
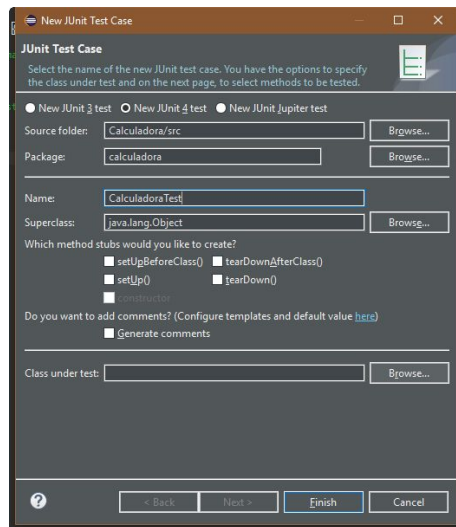
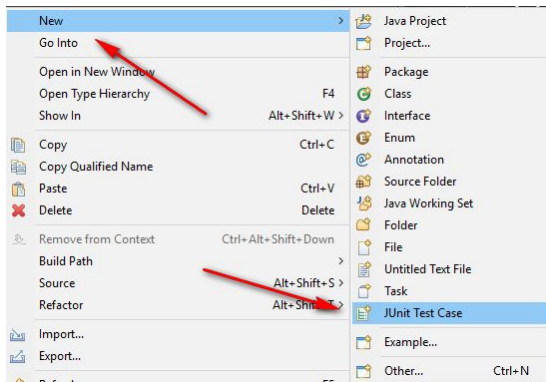
Antes de explicar como hacer test basado en web, vamos a ver un ejemplo básico de JUnit con una aplicación en Java.

Ejemplo básico →

```
Calculadora.java x
1 package calculadora;
2
3 public class Calculadora {
4
5     public static int suma(int a, int b) {
6         return a + b;
7     }
8
9     public static int resta(int a, int b) {
10        return a - b;
11    }
12
13 }
14
```

3. JUnit + Selenium

Una vez tengamos nuestro ejemplo de código Java crearemos en nuestro proyecto un fichero “**JUnit Test Case**”. Este no deja de ser una plantilla de un archivo Java para usar como base para realizar nuestros tests.



3. JUnit + Selenium

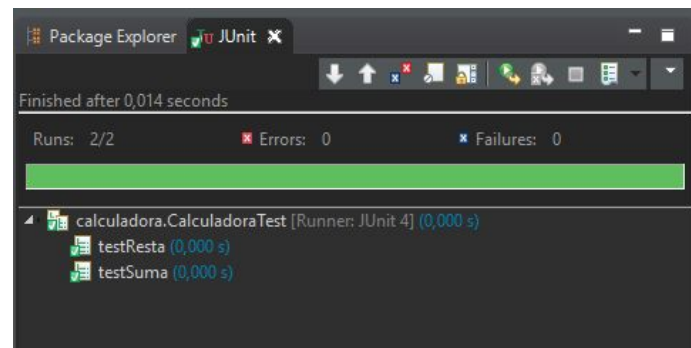
Cada test que queremos realizar irá precedido de la etiqueta “**@Test**” y para la realización de estas emplearemos las funciones importadas de “Assert”, todas estas se pueden encontrar escribiendo “**assert**” y empleado el autocompletado del IDE para localizar la función que se adecue a la situación.

En este caso queremos ejecutar las funciones con unos valores y comprobar si el resultado es igual al esperado.

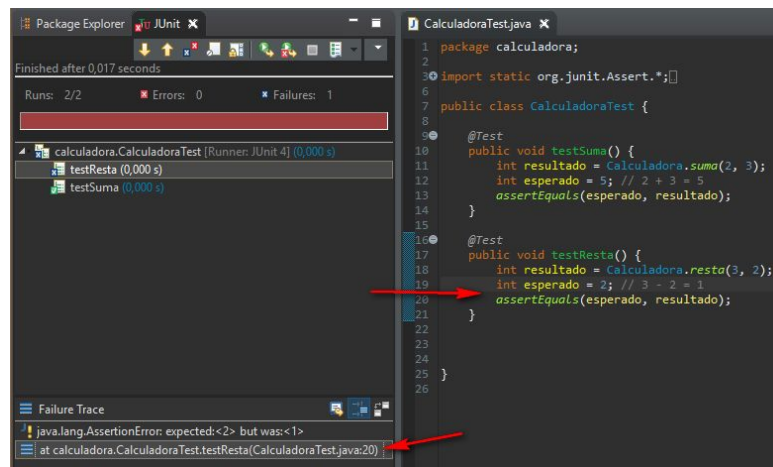
```
*CalculadoraTest.java x
1 package calculadora;
2
3+ import static org.junit.Assert.*;
6
7 public class CalculadoraTest {
8
9 @Test
10 public void testSuma() {
11     int resultado = Calculadora.suma(2, 3);
12     int esperado = 5; // 2 + 3 = 5
13     assertEquals(esperado, resultado);
14 }
15
16 @Test
17 public void testResta() {
18     int resultado = Calculadora.resta(3, 2);
19     int esperado = 1; // 3 - 2 = 1
20     assertEquals(esperado, resultado);
21 }
22
23
24
25 }
26
```

3. JUnit + Selenium

Una vez acabada esta parte solo queda ejecutarlo como un “**JUnit Test**”. Esto nos generará una nueva **vista** con los resultados.



En caso de que el resultado no fuera el esperado se nos notificará donde esta el error



3. JUnit + Selenium



Ahora que sabemos como funciona más o menos JUnit pasemos a la parte de Selenium.

Para realizar una grabación de las funciones que queremos comprobar usaremos una extensión llamada **Katalon Recorder**, un IDE para Selenium que podemos encontrar como extensión o plugin en navegadores como Chrome o Firefox.

3. JUnit + Selenium

The screenshot shows the Selenium IDE interface. The top toolbar contains buttons for 'New', 'Record', 'Play', 'Play Suite', 'Play All', 'Pause', and 'Export'. The main area is divided into a 'Test Suites' pane on the left and a 'Command' table on the right. The 'Test Suites' pane shows a tree view with 'Untitled Test Suite*' and a sub-suite 'prueba *'. The 'Command' table lists the following steps:

Command	Target	Value
open	http://localhost:3000/	
click	name=nombre	
type	name=nombre	test
click	name=sala	
type	name=sala	1
click	id=enterChat	
type	id=txtMensaje	test
click	id=sendTxt	

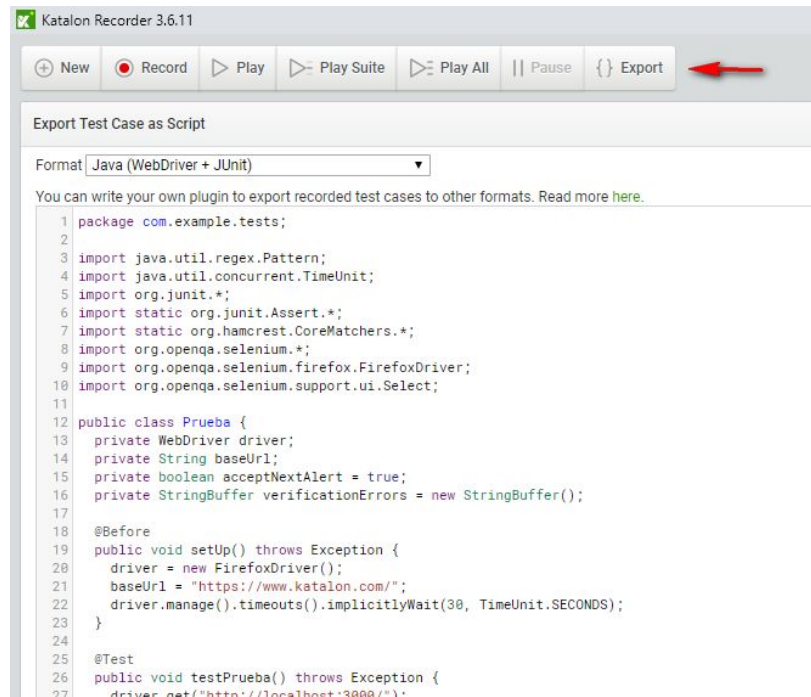
Below the table are input fields for 'Command', 'Target', and 'Value'. A summary bar at the bottom of the main area shows 'Passed: 1' and 'Failed: 0'. The bottom panel contains a log with the following entries:

```
[info] Executing: | click | name=sala | |
[info] Executing: | type | name=sala | 1 |
[info] Executing: | click | id=enterChat | |
[info] Executing: | type | id=txtMensaje | test |
[info] Executing: | click | id=sendTxt | |
[info] Time: Wed Nov 07 2018 17:04:08 GMT+0100 (hora estándar de Europa central) Timestamp: 1541606648872
[info] Test case passed
```

3. JUnit + Selenium

Una vez realizadas las grabaciones podemos exportar estas en el formato de JUnit.

Igualmente lo más seguro es que tengamos que ajustar tanto el nombre del paquete y la clase, así como importar las librerías necesarias.



```
1 package com.example.tests;
2
3 import java.util.regex.Pattern;
4 import java.util.concurrent.TimeUnit;
5 import org.junit.*;
6 import static org.junit.Assert.*;
7 import static org.hamcrest.CoreMatchers.*;
8 import org.openqa.selenium.*;
9 import org.openqa.selenium.firefox.FirefoxDriver;
10 import org.openqa.selenium.support.ui.Select;
11
12 public class Prueba {
13     private WebDriver driver;
14     private String baseUrl;
15     private boolean acceptNextAlert = true;
16     private StringBuffer verificationErrors = new StringBuffer();
17
18     @Before
19     public void setUp() throws Exception {
20         driver = new FirefoxDriver();
21         baseUrl = "https://www.katalon.com/";
22         driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
23     }
24
25     @Test
26     public void testPrueba() throws Exception {
27         driver.get("http://localhost:3000/");
```

3. JUnit + Selenium

Apuntes para hacer que funcione correctamente:

- Importar las librerías de Selenium para Java

<https://www.seleniumhq.org/download/>

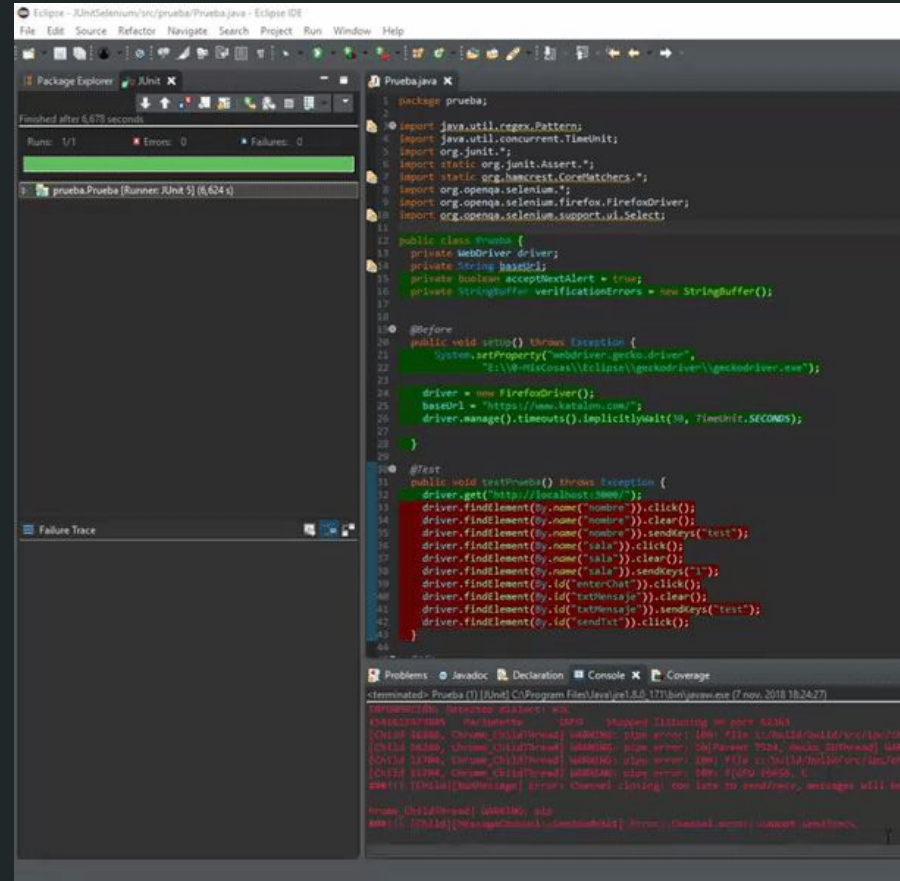
- Descargar e incorporar GeckoDriver

<https://www.softwaretestinghelp.com/geckodriver-selenium-tutorial/>

3. JUnit + Selenium

Ejemplo de ejecución →

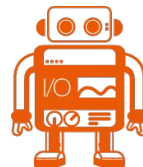
Nota: en caso de no poder visualizar el gif clicar [aquí](#) para ver el recurso asociado.



4. CodeceptJS + WebDriverIO



+



CodeceptJS es un framework de pruebas end-to-end con WebDriver (u otras). Resume la interacciones del navegador que pueden realizar mediante pasos simples que están escritos desde la perspectiva del usuario.

Para la instalación de esta requerimos de **Node.js** y **npm**.



Una vez instalados estos instalaremos los siguientes **paquetes** de forma global:

```
npm install -g codeceptjs webdriverio selenium-standalone
```


4. CodeceptJS + WebDriverIO

Una vez realizado esto, desde la terminal y desde el directorio en el que queremos almacenar los ficheros, iniciamos el CodeceptJS mediante:

```
codeceptjs init
```

Esto nos hará unas preguntas para definir la configuración inicial del entorno para las pruebas.

```
E:\0-MisCosas\Proyectos\test>codeceptjs init

Welcome to CodeceptJS initialization tool
It will prepare and configure a test environment for you

Installing to E:\0-MisCosas\Proyectos\test
? Where are your tests located? ./*_test.js
? What helpers do you want to use? WebDriverIO
? Where should logs, screenshots, and reports to be stored? ./output
? Would you like to extend I object with custom steps? Yes
? Do you want to choose localization for tests? English (no localization)
? Where would you like to place custom steps? ./steps_file.js
Configure helpers...
? [WebDriverIO] Base url of site to be tested http://localhost:3000
? [WebDriverIO] Browser in which testing will be performed chrome
Steps file created at E:\0-MisCosas\Proyectos\test\steps_file.js
Config created at E:\0-MisCosas\Proyectos\test\codecept.json
Directory for temporary output files created at `output`
Almost done! Create your first test by executing `codeceptjs gt` (generate test) command

E:\0-MisCosas\Proyectos\test>
```

4. CodeceptJS + WebDriverIO

Como nos indicaba en la diapositiva anterior, al finalizar podemos crear un ejemplo mediante el siguiente comando:

```
codeceptjs gt
```

Este será el aspecto del archivo generado:

```
E:\0-MisCosas\Proyectos\test>codeceptjs gt
Creating a new test...
-----
? Filename of a test prueba
? Feature which is being tested Prueba
Test for prueba was created in E:\0-MisCosas\Proyectos\test\prueba_test.js

E:\0-MisCosas\Proyectos\test>
```

```
prueba_test.js
1
2   Feature('Prueba');
3
4   Scenario('test something', (I) => {
5
6   });
7
```

4. CodeceptJS + WebDriverIO

¿Cómo definimos las pruebas que queremos que realice? Pues de una forma sencilla, ya que el código que debemos introducir se asemeja a la definición de qué queremos hacer. Este sería un ejemplo:

```
prueba_test.js
1
2 Feature('Prueba');
3
4 Scenario('test something', (I) => {
5   I.amOnPage('/');
6   I.fillField('nombre', 'test');
7   I.fillField('sala', '1');
8   I.click('#enterChat');
9   I.see('Sala de chat');
10  I.fillField('#txtMensaje', 'test');
11  I.click('#sendTxt');
12  I.see('test', '#divChatbox');
13 });
14
```

Como se puede deducir:

- I.amOnPage** → define a qué directorio nos queremos mover
- I.fillField** → nos rellenara un campo existente con ese nombre
- I.click** → simulará que clicamos en un elemento
- I.see** → buscará un texto o un elemento donde le indiquemos

4. CodeceptJS + WebDriverIO

Los ejecutaremos mediante:

```
codeceptjs run --steps
```

Nota: iniciar el selenium en otra terminal mediante:

```
selenium-standalone start
```

4. CodeceptJS + WebDriverIO

Este proceso lo realizará una simulación real en un navegador y de una forma muy rápida. Una vez finalizado podemos ver el resultado:

```
E:\0-MisCosas\Proyectos\test>codeceptjs run --steps
CodeceptJS v1.4.3
Using test root "E:\0-MisCosas\Proyectos\test"

Prueba --
  test something
    I am on page "/"
    I fill field "nombre", "test"
    I fill field "sala", "1"
    I click "#enterChat"
    I see "Sala de chat"
    I fill field "#txtMensaje", "test"
    I click "#sendTxt"
    I see "test", "#divChatbox"
  ✓ OK in 1950ms

OK 1 passed // 5s
E:\0-MisCosas\Proyectos\test>
```

¿Que pasa si hay un error en alguno de los procesos?
Que la simulación se detendrá y nos notificará donde:

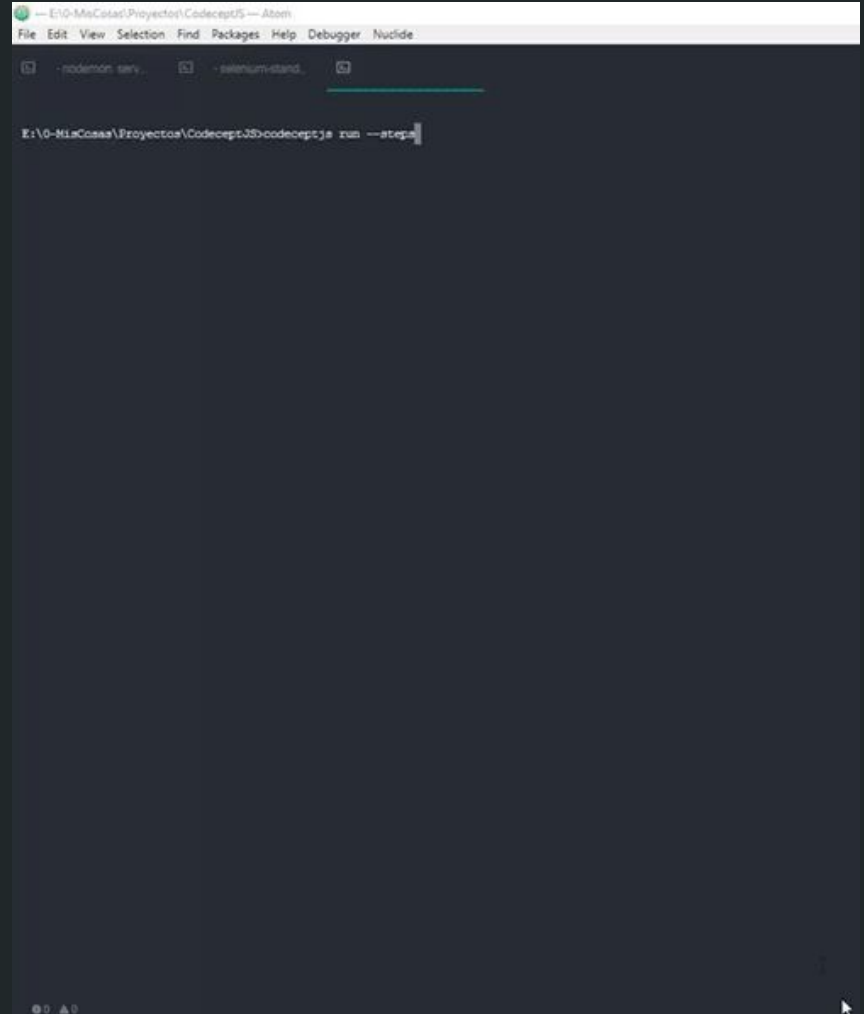
```
E:\0-MisCosas\Proyectos\test>codeceptjs run --steps
CodeceptJS v1.4.3
Using test root "E:\0-MisCosas\Proyectos\test"

Prueba --
  test something
    I am on page "/"
    I fill field "fail", "test"
  ✗ FAILED in 1277ms
```

4. CodeceptJS + WebDriverIO

Ejemplo de ejecución →

Nota: en caso de no poder visualizar el gif clicar [aquí](#) para ver el recurso asociado.



4. CodeceptJS + WebDriverIO

Esto es solo una parte sencilla de lo que se puede hacer con CodeceptJS.

Entre otras cosas más podemos hacer se encuentran: guardar un registro, realizar capturas de pantalla, simular múltiples sesiones o hacer que espere a que ocurran ciertos eventos que requieran tiempo entre otras cosas.

5. Conclusión

En la actualidad aplicaciones como estas nos ahorran mucho tiempo a la hora de probar nuestras aplicaciones, no solo realizando las pruebas en sí, sino que, también preparando los procesos que estas deben seguir.