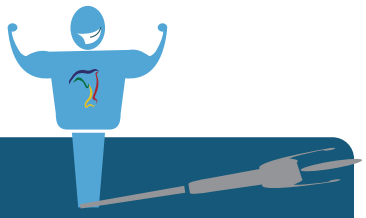
autentia

Avenida de Castilla,1 - Edificio Best Point - Oficina 21B
28830 San Fernando de Henares (Madrid)
tel./fax: +34 91 675 33 06
info@autentia.com - **www.autentia.com**

¿**Qué ofrece** Autentia Real Business Solutions S.L?

Somos su empresa de **Soporte a Desarrollo Informático**.
Ese apoyo que siempre quiso tener...

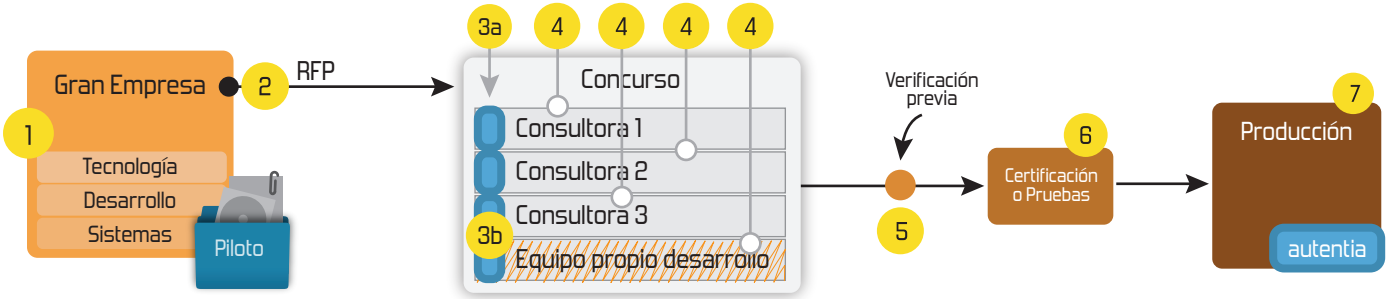
1. Desarrollo de componentes y proyectos a medida



2. Auditoría de código y recomendaciones de mejora

3. Arranque de proyectos basados en nuevas tecnologías

1. Definición de frameworks corporativos.
2. Transferencia de conocimiento de nuevas arquitecturas.
3. Soporte al arranque de proyectos.
4. Auditoría preventiva periódica de calidad.
5. Revisión previa a la certificación de proyectos.
6. Extensión de capacidad de equipos de calidad.
7. Identificación de problemas en producción.



4. Cursos de formación (impartidos por desarrolladores en activo)

- Spring MVC, JSF-PrimeFaces /RichFaces, HTML5, CSS3, JavaScript-jQuery
- Gestor portales (Liferay)
- Gestor de contenidos (Alfresco)
- Aplicaciones híbridas
- Control de autenticación y acceso (Spring Security)
- UDDI
- Web Services
- Rest Services
- Social SSO
- SSO (Cas)
- JPA-Hibernate, MyBatis
- Motor de búsqueda empresarial (Solr)
- ETL (Talend)
- Dirección de Proyectos Informáticos.
- Metodologías ágiles
- Patrones de diseño
- TDD
- Tareas programadas (Quartz)
- Gestor documental (Alfresco)
- Inversión de control (Spring)
- BPM (jBPM o Bonita)
- Generación de informes (JasperReport)
- ESB (Open ESB)

Compartimos nuestro conocimiento en:
www.adictosaltrabajo.com

Para más información visítenos en:
www.autentia.com

Rubén Aguilera Díaz-Heredero

Consultor tecnológico de desarrollo de proyectos informáticos.

Ingeniero en Informática, especialidad en Ingeniería del Software

Puedes encontrarme en Autentia: Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/J2EE

Ver todos los tutoriales del autor

Fecha de publicación del tutorial: 2014-08-07 Tutorial visitado 216 veces Descargar en PDF

Smoke Test implementados con TestNG y Selenium

0. Índice de contenidos.

1. Entorno

Este tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil Mac Book Pro 15" (2,3 Ghz Intel Core i7, 16 GB DDR3)
- Sistema Operativo: Mac OS X Mavericks

2. Introducción

Los smoke tests son un tipo de test que nos permiten verificar que el software en conjunto no "echa humo". Es decir, que podemos completar un recorrido típico de la aplicación sin que se detecte ningún error grave. Por lo tanto solo miran casos positivos y no contemplan los casos en los que los usuarios puedan hacer cosas raras; así que en ningún caso sustituyen a otros tipos de tests más exhaustivos como los unitarios y los de integración.

Se caraterizan por estar automatizados y estar relacionados con las features del proyecto. De hecho, idealmente, contaremos con un smoke test por feature o historia de usuario del proyecto. El cual, en muchas ocasiones, determinará el DONE de una historia.

Podemos utilizarlos para ejecutarlos siempre que queramos estar tranquilos de que la aplicación no está fallando en ningún módulo de la interfaz de usuario como cuando vamos a pasar la aplicación al departamento de pruebas o antes y después de desplegar una release.

Integrados con una herramienta como TestLink puede ser un radiador de información para que el cliente pueda ver el avance de las historias de usuario de un sprint.

3. Vamos al lío

Vamos a crear nuestro primer smoke test. Lo implementaremos con TestNG y Selenium. TestNG es un framework de pruebas basado en JUnit pero al que añade nuevas funcionalidades que lo hacen más poderoso y fácil de utilizar; y Selenium es el framework que nos permite realizar operaciones sobre un navegador para simular la interacción del usuario con la aplicación.

Lo primero que vamos a hacer es crear un proyecto Maven e importarlo en Eclipse.

```
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
Choose a number: 6:
Define value for property 'groupId': : com.autentia.tutoriales
Define value for property 'artifactId': : smoke-tests
Define value for property 'version':  1.0-SNAPSHOT: :
Define value for property 'package':  com.autentia.tutoriales: :
Confirm properties configuration:
groupId: com.autentia.tutoriales
artifactId: smoke-tests
version: 1.0-SNAPSHOT
package: com.autentia.tutoriales
 Y: :
```

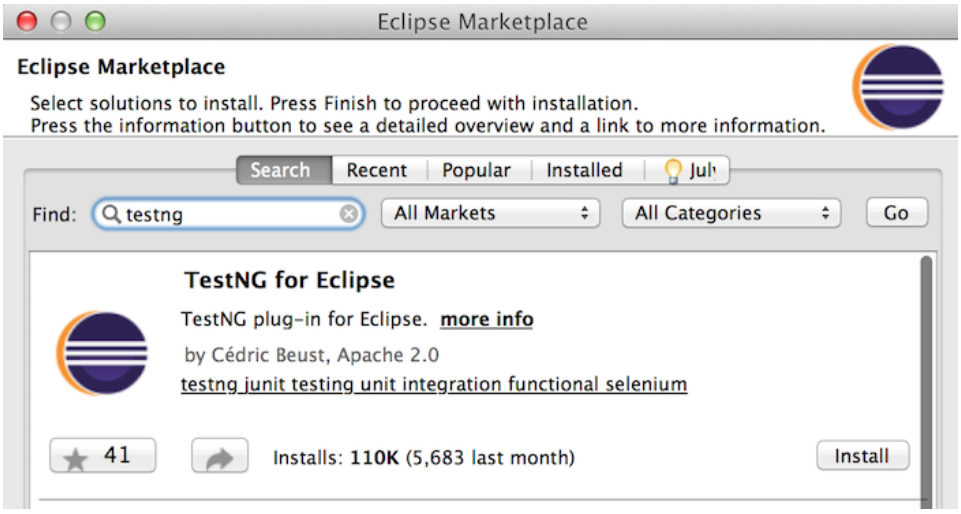
Ahora añadimos las siguientes dependencias en el pom.xml de nuestro proyecto:

```
view plain print ?
01. <dependency>
02.   <groupId>org.testng</groupId>
03.   <artifactId>testng</artifactId>
04.   <version>6.8.7</version>
05.   <scope>test</scope>
06. </dependency>
07. <dependency>
08.   <groupId>org.seleniumhq.selenium</groupId>
09.   <artifactId>selenium-java</artifactId>
10.   <version>2.39.0</version>
11. </dependency>
```

Ya tenemos todo para poder implementar nuestro primer test. Vamos a crear un nuevo test JUnit con el siguiente contenido:

```
view plain print ?
01. package com.autentia.tutoriales;
02.
03. import org.openqa.selenium.WebDriver;
04. import org.openqa.selenium.firefox.FirefoxDriver;
05. import org.testng.Assert;
06. import org.testng.annotations.Test;
07.
08. public class AutentiaWebTest {
09.
10.   @Test
11.   public void getTitleAutentiaTest() {
12.     WebDriver driver = new FirefoxDriver();
13.     driver.get("http://www.autentia.com");
14.     Assert.assertEquals("Autentia | Soporte a desarrollo informático", driver.getTitle());
15.     driver.quit();
16.   }
17.
18. }
```

Para ejecutar el test desde Eclipse antes tenemos que instalar el plugin de TestNG que lo podemos encontrar en el marketplace.



Después de reiniciar para hacer efectiva la instalación del plugin. Vamos a la clase y con botón derecho seleccionamos Run As --> TestNG Test

Al ejecutar el test podremos ver que el navegador automáticamente se abre un momento, se conecta a la página de Autentia y nos devuelve el título de la página que está visualizando. Además muestra en verde el test en la ventana de TestNG

Últimos Tutoriales del Autor

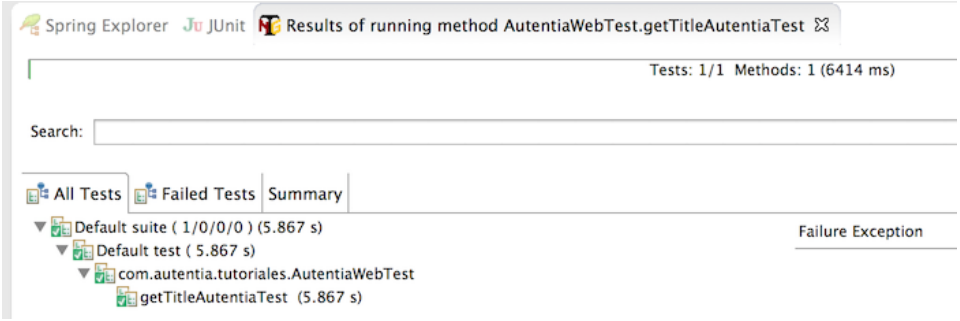
» GitLab: Crear y gestionar nuestro servidor propio de Git

» Crear servidor propio de Git en CentOS 6.5

» Crear un plugin para Android en PhoneGap

» Intercomunicación de aplicaciones en IOS

» Crashlytics en IOS



Este es un ejemplo muy sencillo. El API de Selenium nos permite hacer muchas más cosas como: hacer click en botones, rellenar formularios, recuperar información de la pagina, etc...

Vamos a ilustrarlo con un ejemplo un poco más complejo. Ahora el objetivo es hacer una búsqueda en la página de Autentia por el palabra "formación" y recuperar la última parte del camino de migas que se muestra en la página cuando finaliza la búsqueda.

```
view plain print ?
01.
02. @Test
03. public void fillSearchFormAndExecute(){
04.   WebDriver driver = new FirefoxDriver();
05.   driver.get("http://autentia.com/?s");
06.
07.   WebElement search = driver.findElement(By.id("s"));
08.   search.sendKeys("formación");
09.
10.   search.submit();
11.
12.   //Esperamos 10 segundos o a que devuelva el resultado de la búsqueda
13.   (new WebDriverWait(driver, 10)).until(new ExpectedCondition<Boolean>() {
14.           public Boolean apply(WebDriver d) {
15.               return d.findElement(By.className("trail-end")).getText().startsWith("Búsqueda");
16.           }
17.       });
18.
19.   Assert.assertEquals(driver.findElement(By.className("trail-
    end")).getText(), "Búsqueda de \"formación\"");
20.
21.   driver.quit();
22. }
```

Como se puede leer en el código, ahora nos conectamos a la página de búsqueda, buscamos el input del formulario por su id, lo rellenamos con la palabra "formación" y hacemos submit del formulario. Fijaos que no hemos tenido que buscar el elemento del botón, Selenium lo ha hecho por nosotros al indicarle que queremos hacer submit del formulario que contiene a nuestro elemento.

Dado que tenemos que esperar a que el servidor nos devuelva los resultados de la búsqueda, implementamos una funnción que espera a que pasen 10 segundos o a que se cumpla la condición que le establecemos, que no es otra que se muestra el texto "Búsqueda" en el camino de migas.

Una vez estamos seguros de que la página ha cargado, recuperamos todo el contenido de la parte del camino de migas que viene especificado con la clase "trail-end" y lo comparamos con el texto que esperamos.

Si volvéis ha ejecutar la clase con el nuevo test veréis que ahora el navegador Firefox se abrirá y cerrará dos veces para satisfacer las operaciones de los test y la venta de TestNG nos mostrará que los test han pasado con un color verde.

Esta misma prueba se puede realizar desde la consola con Maven ejecutando "mvn test"

```
-------------------------------------------------------
 T E S T S
-------------------------------------------------------
Running com.autentia.tutoriales.AutentiaWebTest
Configuring TestNG with: org.apache.maven.surefire.testng.conf.TestNG652Configurator@818de3
Tests run: 2, Failures: 0, Errors: 0, Skipped: 0, Time elapsed: 12.264 sec

Results :

Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
```

4. Conclusiones

Como véis los smoke tests no son muy difíciles de implementar y nos aportan un grado más de testabilidad en nuestros proyectos; pero recordad que nunca pueden sustituir a los test unitarios y de integración de nuestras aplicaciones. Os recomiendo que miréis el API de WebDriver y de WebElement para ver todo lo que se puede llegar a hacer.

Cualquier duda o sugerencia en la zona de comentarios.

Saludos.

Por favor, vota +1 o compártelo si te pareció interesante

Anímate y coméntanos lo que pienses sobre este **TUTORIAL:**

» Registrate y accede a esta y otras ventajas **«**