

¿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

Tareas programadas (Quartz)
Gestor documental (Alfresco)
Inversión de control (Spring)

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

BPM (jBPM o Bonita)
Generación de informes (JasperReport)
ESB (Open ESB)

AdictosAlTrabajo

Terrakas 1x05
¡¡Ya está en la web!! :-)
terrakas.com



autentia
Soporte a desarrollo informático
Hosting patrocinado por
enREDados

Entra en Adictos a través de  

E-mail

Contraseña

Entrar

[Deseo registrarme](#)
[Olvidé mi contraseña](#)

[Inicio](#) [Quiénes somos](#) [Formación](#) [Comparador de salarios](#) [Nuestro libro](#) [Más](#)



» Estás en: [Inicio](#) [Tutoriales](#) Sonar y Javascript: obteniendo la cobertura de nuestro código



Miguel Arlandy Rodríguez

Consultor tecnológico de desarrollo de proyectos informáticos.

Puedes encontrarme en [Autentia](#): Ofrecemos servicios de soporte a desarrollo, factoría y formación

Somos expertos en Java/JEE



[Ver todos los tutoriales del autor](#)



Fecha de publicación del tutorial: 2013-01-17

Tutorial visitado 1 veces [Descargar en PDF](#)

Sonar y Javascript: obteniendo la cobertura de nuestro código.

0. Índice de contenidos.

- 1. Introducción.
- 2. Entorno.
- 3. Instalando y configurando el plugin de Javascript.
- 4. Analizando el proyecto con Maven.
- 5. Analizando el proyecto con Sonar Runner.
- 6. Referencias.
- 7. Conclusiones.

1. Introducción

Entre las muchas funcionalidades que ofrece Sonar está la de medir el código cubierto por nuestros tests unitarios (cobertura). Cualquiera que haya trabajado con Sonar, Java y Maven sabe que la forma de medir la cobertura de nuestro código es muy sencilla, de hecho, suele bastar con escribir nuestros test y analizar el proyecto con Sonar.

En el tutorial "JsTestDriver: Testea tu código Javascript" vimos lo fácil que era realizar test unitarios en Javascript. Siendo así, ¿podemos medir la cobertura de nuestro código Javascript con Sonar?

En este tutorial veremos cómo medir el código Javascript cubierto por nuestros tests con Sonar y su correspondiente plugin.

2. Entorno.

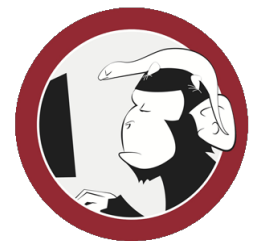
El tutorial está escrito usando el siguiente entorno:

- Hardware: Portátil MacBook Pro 15' (2.2 Ghz Intel Core I7, 8GB DDR3).
- Sistema Operativo: Mac OS Mountain Lion 10.8
- Entorno de desarrollo: [IntelliJ Idea 11.1 Ultimate](#).
- Sonar 3.4
- Javascript Sonar plugin v1.2

3. Instalando y configurando el plugin de Javascript.

Lo primero que debemos hacer para medir el código cubierto por nuestros tests es instalar el plugin de Javascript para Sonar. Es muy sencillo, antes de nada accederemos a la aplicación con un usuario administrador. Después **System > Update Center > Available Plugins** y seleccionamos e instalamos el plugin de Javascript.

Catálogo de servicios Autentia



Síguenos a través de:



Últimas Noticias

» ¿Qué es aportar valor como técnico/programador?

» Como arrancar un nuevo proyecto e integrar el agilidad en una organización

» Hoy es el primer día para cambiar tu sector

» AdictosAlTrabajo os desea
¡¡¡FELICES FIESTAS!!!

» ESTRENO de Terrakas 1x05: "Un lugar en el mundo"

[Histórico de noticias](#)

Últimos Tutoriales

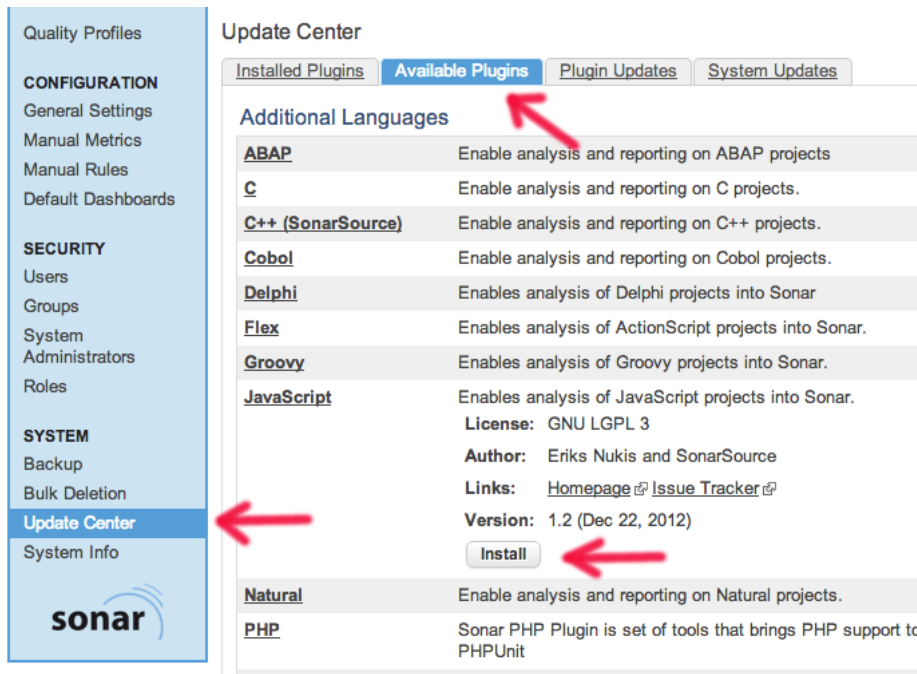
» ZKPushState: Manejar el API de Historial de navegación en HTML5 directamente desde Java

» [Modela tu mercado en base a la demanda y no a la oferta.](#)

» Mirar el todo: llevar el agilidad a las grandes organizaciones.

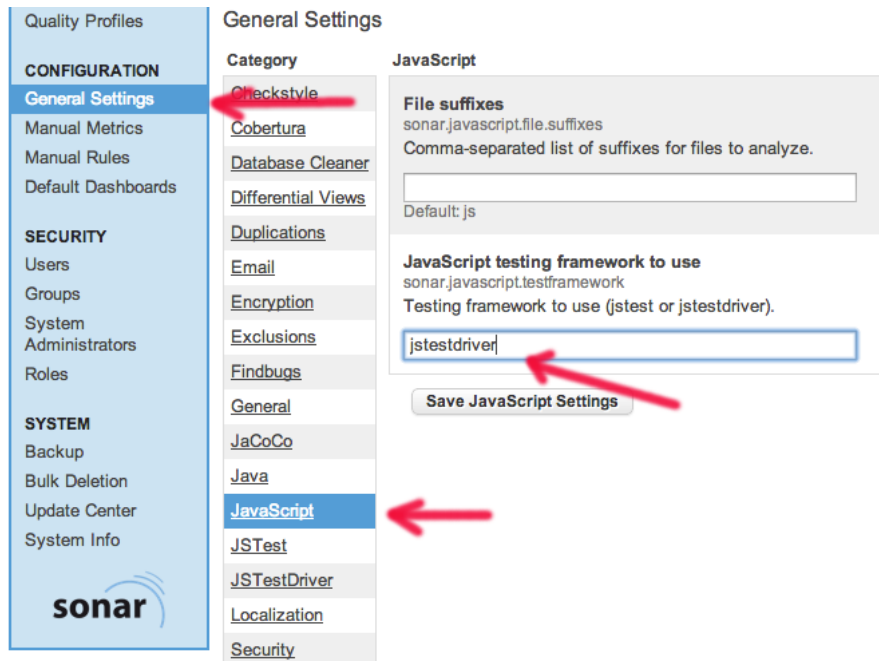
» Creación de un gif animado con Adobe Photoshop

» Lanzando nuestros tests de jasmine-node con IntelliJ IDEA



Después reiniciamos Sonar.

Nuevamente accedemos con un usuario administrador y, esta vez, configuraremos el plugin para que utilice JsTestDriver como framework de tests. Para ello **Configuration > General Settings > Javascript** y nos aseguramos de que el framework que utilizaremos para test sea JsTestDriver, tal y como se muestra en la siguiente figura:



Pues ya tendríamos todo listo para empezar a medir la cobertura de nuestros proyectos Javascript con Sonar.

En el tutorial "JsTestDriver: testea tu código Javascript" utilizamos un objeto llamado Validador que comprobaba que los valores de los campos de un formulario fuesen correctos. Utilizaremos ese mismo ejemplo para medir su cobertura.

Podemos medir la cobertura de nuestro proyecto de dos formas: con Maven o con Sonar Runner. A continuación veremos las dos formas.

4. Analizando el proyecto con Maven.

Tenemos nuestro proyecto organizado de la siguiente manera:

Últimos Tutoriales del Autor

» Sonar y Total Quality: midiendo la calidad total de nuestros proyectos

» Servicios REST documentados y probados con Swagger

» Creación de plantillas DSL con Drools

» Introducción a Drools.

» Jugando con JSON en Java y la librería Gson

Últimas ofertas de empleo

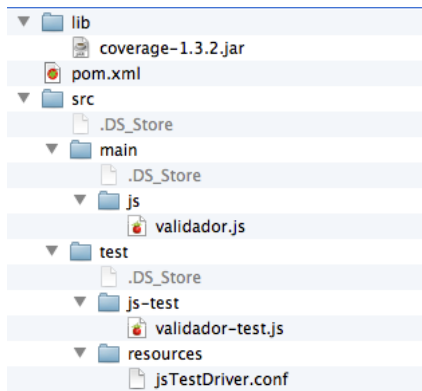
2011-09-08
Comercial - Ventas - MADRID.

2011-09-03
Comercial - Ventas - VALENCIA.

2011-08-19
Comercial - Compras - ALICANTE.

2011-07-12
Otras Sin catalogar - MADRID.

2011-07-06
Otras Sin catalogar - LUGO.



Como podemos ver, en el directorio **src/main/js/** tenemos el fichero **validador.js** que contiene el código de producción y que será el testearmos. Del mismo modo, en el directorio **src/test/js-test/** tenemos el fichero **validador-test.js** que contiene los test que aplican sobre **validador.js**.

Como vimos en el tutorial "JsTestDriver: testea tu código Javascript", para analizar un proyecto con JsTestDriver es necesario un fichero **jsTestDriver.conf** donde indicaremos donde tenemos nuestro código de producción (código a testear) y nuestros tests. Además debemos hacer uso del **plugin de cobertura** de JsTestDriver para que nos mida el porcentaje de código cubierto por los tests.

El fichero **jsTestDriver.conf** quedaría de la siguiente forma:

```
1 server: http://localhost:9876
2
3 load:
4   - src/main/js/*.js
5
6 test:
7   - src/test/js-test/*.js
8
9 plugin:
10  - name: "coverage"
11    jar: "lib/coverage-1.3.2.jar"
12    module: "com.google.jstestdriver.coverage.CoverageModule"
```

Como vemos, el plugin de cobertura está en una librería **coverage-1.3.2.jar** que está en el directorio **lib/** de nuestro proyecto. Esta librería puede descargarse desde la [página de JsTestDriver](#).

Pues lo siguiente que faltaría es configurar nuestro **pom.xml** para que haga uso del plugin de JsTestDriver de forma que, se lancen los tests automáticamente y se genere un informe con la cobertura.

El **pom.xml** tendría el siguiente aspecto:

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema"
3   <modelversion>4.0.0</modelversion>
4
5   <groupid>com.autentia.tutoriales</groupid>
6   <artifactid>sonar-javascript-maven</artifactid>
7   <version>1.0-SNAPSHOT</version>
8
9   <name>Sonar Javascript y Maven</name>
10
11   <properties>
12     <project.build.sourceencoding>UTF-8</project.build.sourceencoding>
13     <sonar.language>js</sonar.language>
14     <sonar.dynamicanalysis>reuseReports</sonar.dynamicanalysis>
15     <sonar.javascript.jstestdriver.reportsfolder>target/jstestdriver</sonar.javascript.js
16     <path.to.browser>/Applications/Firefox.app/Contents/MacOS/firefox</path.to.browser>
17   </properties>
18
19   <build>
20     <sourcedirectory>src/main/js</sourcedirectory>
21     <testsourceDirectory>src/test/js-test</testsourceDirectory>
22
23     <plugins>
24       <plugin>
25         <groupid>com.googlecode.jstd-maven-plugin</groupid>
26         <artifactid>jstd-maven-plugin</artifactid>
27         <version>1.3.2.5</version>
28         <configuration>
29           <verbose>true</verbose>
30           <browser>${path.to.browser}</browser>
31           <port>9876</port>
32           <testoutput>target/jstestdriver</testoutput>
33         </configuration>
34         <executions>
35           <execution>
36             <id>run-tests</id>
37             <goals>
38               <goal>test</goal>
39             </goals>
40           </execution>
41         </executions>
42       </plugin>
43     </plugins>
44   </build>
45
46   <!-- JsTestDriver Dependencies -->
47   <dependencies>
48     <dependency>
49       <groupid>com.googlecode.jstd-maven-plugin</groupid>
50       <artifactid>jstd-maven-plugin</artifactid>
51       <version>1.3.2.5</version>
52       <scope>test</scope>
53     </dependency>
54   </dependencies>
55
```

```

56 <repositories>
57 <repository>
58 <id>jstd-maven-plugin google code repo</id>
59 <url>http://jstd-maven-plugin.googlecode.com/svn/maven2</url>
60 </repository>
61 </repositories>
62 <pluginrepositories>
63 <pluginrepository>
64 <id>jstd-maven-plugin google code repo</id>
65 <url>http://jstd-maven-plugin.googlecode.com/svn/maven2</url>
66 </pluginrepository>
67 </pluginrepositories>
68 </project>

```

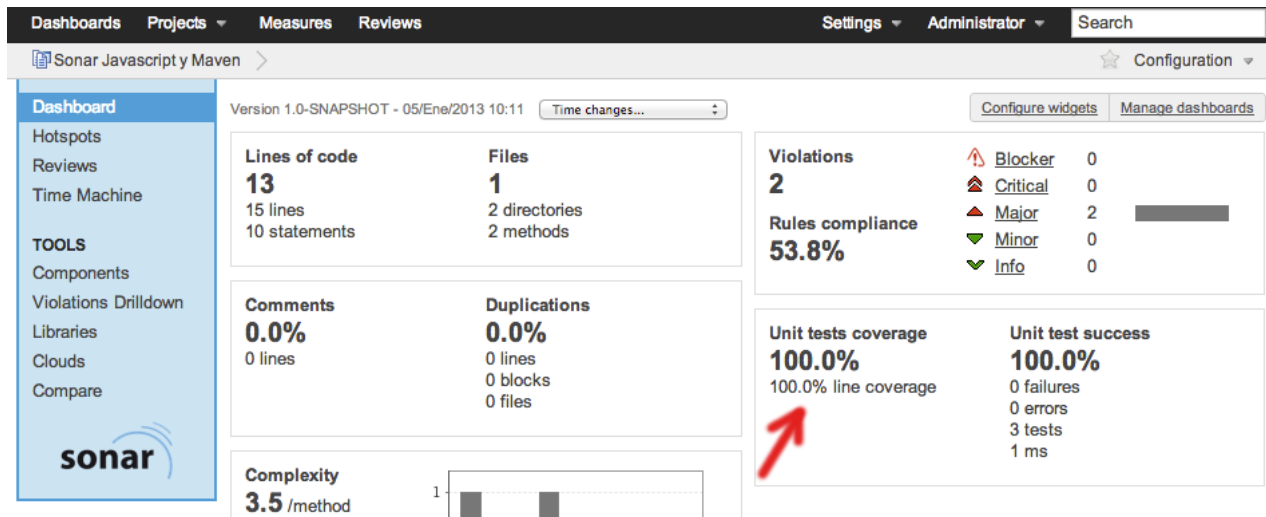
De esta forma, cuando lleguemos a la fase de test de Maven (ver [ciclo de vida de Maven](#)), JsTestDriver entrará en acción: correrá los tests y generará un informe con la cobertura que dejará en el directorio **target/jstestdriver/**. Dicho informe será consumido por Sonar cuando analice el proyecto.

Por tanto, lo único que nos falta es ejecutar los siguientes comandos (en el directorio raíz del proyecto):

```

1 mvn test
2 mvn sonar:sonar

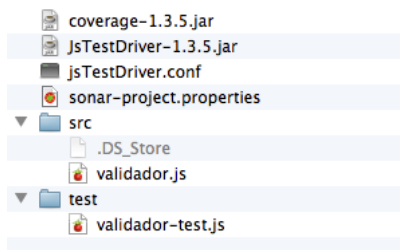
```



5. Analizando el proyecto con Sonar Runner.

Y el que no quiera utilizar Maven puede hacerlo con Sonar Runner. Quien no tenga ni idea de qué es Sonar Runner puede echarle un ojo a [este tutorial mi compi Rubén](#).

Tenemos estructurado el proyecto de la siguiente manera:



Como vemos, ya no tenemos fichero pom.xml. En su lugar tenemos el fichero **sonar-project.properties** que será usado por Sonar Runner. Este sería su aspecto:

```

1 # project metadata (required)
2 sonar.projectKey=com.autentia.tutoriales.sonarjavascriptsonarrunner
3 sonar.projectName=Sonar Javascript y Sonar Runner
4 sonar.projectVersion=1.0
5
6 # path to source directories (required)
7 sonar.sources=src
8
9 # path to test source directories (optional)
10 sonar.tests=test
11
12 # Language
13 sonar.language=js
14
15 # Advanced parameters
16 sonar.javascript.jstestdriver.reportsfolder=jstestdriver
17 sonar.dynamicAnalysis=reuseReports
18
19 # Encoding of the source files
20 sonar.sourceEncoding=UTF-8

```

Y nuestro fichero **jsTestDriver.conf** cambia ligeramente ya que hemos cambiado las rutas de nuestros fuentes y tests, además de la ruta del plugin de cobertura.

```

1 server: http://localhost:9876
2
3 load:
4   - src/*.js
5 test:
6   - test/*.js
7
8 plugin:
9   - name: "coverage"

```

```
10 | jar: "coverage-1.3.5.jar"
11 | module: "com.google.jstestdriver.coverage.CoverageModule"
```

Además, hemos incluido la librería **JsTestDriver-1.3.5.jar** ya que ahora no se lanza con Maven. Ya explicamos cómo lanzar JsTestDriver de forma manual [en este tutorial](#). Lanzamos el siguiente comando:

```
1 | java -jar JsTestDriver-1.3.5.jar --port 9876 --tests all --browser "/Applications/Firefox?"
```

Esto sería lo equivalente al comando **mvn test** del punto anterior. Por último lanzamos el análisis con Sonar Runner de la siguiente forma:

```
1 | sonar-runner
```

Recordemos que para que este comando nos funcione debemos tener [instalado y configurado Sonar Runner](#).

Y si todo ha ido bien, tendremos un resultado similar al que obtuvimos con Maven :-)

6. Referencias.

- [Plugin de Javascript para Sonar](#)

7. Conclusiones.

En este tutorial hemos visto lo fácil que es medir el código de nuestro proyecto cubierto por tests, una alternativa muy interesante para no descuidar esta faceta durante el desarrollo de nuestros proyectos.

Espero que este tutorial os haya sido de ayuda. Un saludo.

Miguel Arlandy

marlandy@autentia.com

Twitter: [@m_arlandy](#)

A continuación puedes evaluarlo:

[Regístrate para evaluarlo](#)

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

Share |

0

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

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



Esta obra está licenciada bajo licencia [Creative Commons de Reconocimiento-No comercial-Sin obras derivadas 2.5](#)

IMPULSA

Impulsores

Comunidad

¿Ayuda?

0 personas han traído clicks a esta página

sin clicks

+ + + + + + + +

powered by [karmacracy](#)

Copyright 2003-2013 © All Rights Reserved | [Texto legal y condiciones de uso](#) | [Banners](#) | [Powered by Autentia](#) | [Contacto](#)

W3C XHTML 1.0 W3C CSS XML RSS XML RSD