Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computación

Avantek Software

"Estrategia Operativa para pruebas de Automatización y Rendimiento"

Practica de especialidad para optar por el titulo de ingeniería en computación con el grado académico de bachillerato.

Gilberto Martínez Taleno

San Carlos, Junio 2012

Resumen Ejecutivo

La automatización de pruebas de software es un proceso que ha crecido a gran escala en los últimos años debido a las ventajas que presenta, situación por la cual Avantek Software se ha interesado en desarrollar una estrategia de automatización que permita llevar el proceso a nivel operativo.

Para llevar dicho proceso a nivel operativo se desarrolló un repositorio de scripts utilizando un lenguaje de programación orientado a objetos con el objetivo de realizar validaciones y chequeos para incluirlos en una plataforma de automatización de pruebas, la cual también fue desarrollada y cuyo principal objetivo es convertirse en el framework de automatización de la empresa.

Dicho framework se desarrolló utilizando Java como lenguaje de programación y Eclipse como entorno de desarrollo, además de incluir las herramientas Junit, Selenium Web Driver y Selenium Grid.

Junit es un framework que permite ejecutar clases Java de manera controlada para evaluar si el funcionamiento de cada uno de los métodos de una clase se comportan de la manera esperada, por otra parte Selenium Web Driver es un conjunto de clases que provee funciones que se encargan de interactuar con el navegador, permitiendo el envío y recepción de acciones y datos. Con el objetivo de aprovechar estas características se desarrollaron métodos que incluyen funciones de Selenium Web Driver que contienen pasos específicos a ejecutar sobre el navegador web, con funciones de Junit para determinar si el comportamiento es el esperado. Cada uno de esos métodos, que reciben el nombre de casos de prueba, son ejecutados de manera local, pero mediante la herramienta Selenium Grid se puede realizar una ejecución distribuyéndolos entre distintas computadoras.

Durante el desarrollo del proyecto fue necesario incluir nuevas herramientas para poder ejecutar pruebas sobre algunos navegadores, esas herramientas, llamadas controladores hacen interfaz entre el framework y el navegador, y se detallarán en los siguientes capítulos.

Palabras Claves: Framework, automatización, clases, controlador, interfaz, navegador.

Contenido

Capítulo 1
Descripción del problema

1.1 Contexto del proyecto

1.1.1 Antecedentes de la empresa

Avantek Software es una compañía de alta tecnología con Sede en San Carlos, Alajuela, Costa Rica. El modelo de negocios se basa en las tecnologías de información y los servicios ofrecidos incluyen desarrollo de software, outsourcing y personalización de software ya desarrollado.

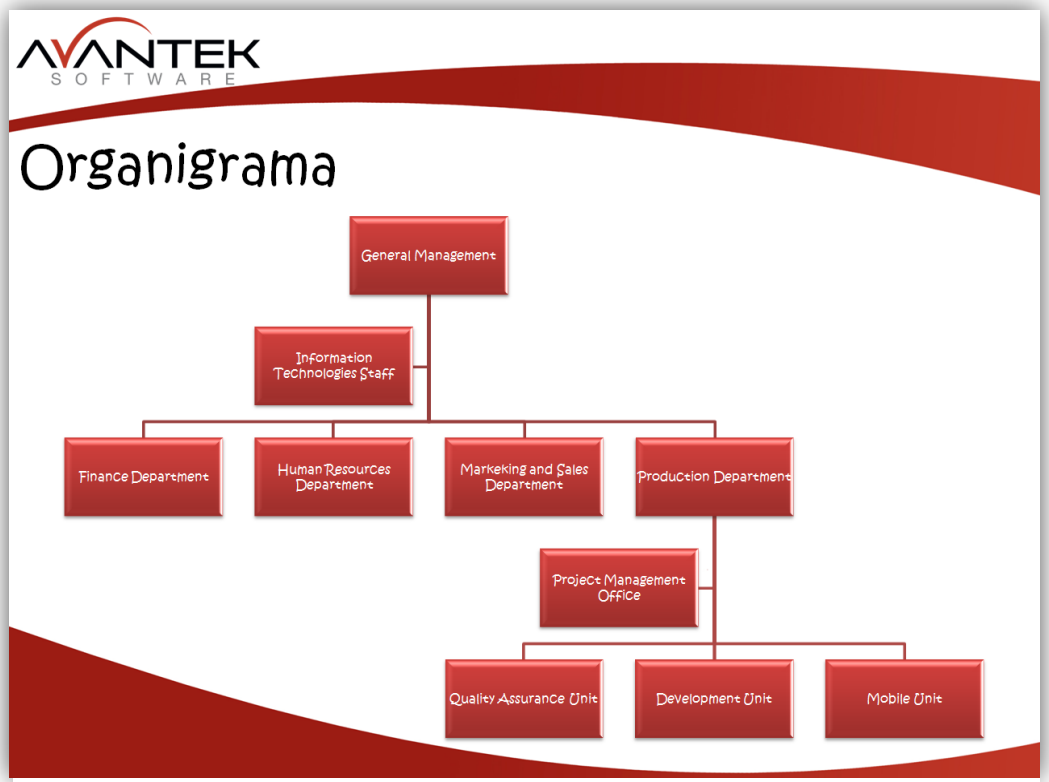


Imagen 1. Estructura Organizacional Avantek Software

1.1.2 Antecedentes del Proyecto

En los últimos años el crecimiento de las empresas dedicadas al desarrollo de software ha sido a gran escala, el mercado se ha diversificado y han surgido nuevas oportunidades, la alta competencia ha provocado la constante investigación y el desarrollo de nuevas técnicas enfocadas optimizar procesos, que disminuyan costos e incrementen el rendimiento de las aplicaciones a desarrollar.

El departamento de QA ha surgido debido a la necesidad de asegurar que el software desarrollado satisfaga tanto las expectativas del cliente como las de la empresa. Desarrolla una serie de actividades que van desde el inicio del proyecto, hasta su finalización, las cuales incluyen procesos de control de calidad, pruebas de software, análisis de pruebas y Gestión de configuración de software, además define procedimientos, métricas y coordina revisión de pares.

Las pruebas de software son organizadas mediante casos de pruebas, los cuales son un conjunto de condiciones o variables bajo las cuales se determinará si el requerimiento a probar de la aplicación es satisfactorio o no. Son utilizadas con el propósito de mejorar la calidad mediante la realización de tareas de verificación y validación, permitiendo conocer la calidad del producto y simplificando la detección y corrección de posibles fallos.

Dichos beneficios se pueden aprovechar para actuar a tiempo y así reducir costos de mantenimiento y de soporte hacia usuarios insatisfechos, además de la obtención de información útil que ayudará a futuros proyectos.

Existen diversas estrategias de pruebas de software, entre las cuales se encuentran:

a) Pruebas unitarias.
b) Pruebas de integración.
c) Pruebas de sistema.
d) Pruebas de aceptación y validación.

Las pruebas unitarias se enfocan en probar funcionalidades específicas del sistema, el ámbito esta dado por la interfaz del modulo, el impacto de los datos globales en el

modulo, las condiciones límite, las estructuras de datos locales, los caminos de ejecución y los caminos de manejo de errores.

Debido al gran crecimiento y la alta competitividad del mercado las herramientas utilizadas para pruebas también han evolucionado ayudando a reducir costos en procesos que solían consumir muchos recursos. La automatización de pruebas es un claro ejemplo de dicha evolución.

El proyecto a desarrollar se enfoca en crear un repositorio de scripts con métodos genéricos utilizando las herramientas Selenium Web Driver y JUnit mediante Java, los cuales permitan la creación casos de pruebas automáticos con el fin de colaborar en la productividad mediante la automatización del proceso pruebas de Avantek Software.

Selenium Web Driver es una tecnología libre desarrollada con el objetivo de automatizar pruebas brindando una serie de funcionalidades que permiten el manejo total del navegador, tanto en ambientes web como móvil. Por su parte JUnit pone a disposición un entorno con una gran cantidad de métodos que permiten desarrollar, ejecutar y obtener resultados de casos de pruebas automáticos.

Avantek necesita incrementar y mejorar la eficiencia en la prestación de servicios en el área de aseguramiento de la calidad del software, por lo que contar con una mejor estrategia operativa de automatización de casos de prueba y ejecución de pruebas de rendimiento vendrán a ayudar a la eficiencia y eficacia de sus servicios, mediante la mejora de los procesos ya establecidos.

El desarrollo e implementación de este plan operativo facilitara la creación de una cartelera más amplia en los servicios de la calidad del software, evolucionando de pruebas manuales a las pruebas automáticas, y de este modo ahorrando tiempo y recursos que podrán ser utilizados en otras actividades o proyectos.

1.2 Descripción del problema

1.2.1 Enunciado del problema

El proyecto consiste en desarrollar un repositorio de scripts (Utilizando clases y objetos) de validación y chequeo para una plataforma de automatización de pruebas, la cual se convertirá en el framework de automatización de Avantek Software.

Esta plataforma necesitará un núcleo, el cual será llamado *AvanticaWebDriver* y será el encargado de crear la instancia del navegador web y comunicarse con él. Entre sus principales funciones debe estar el enviar y recibir datos del navegador, realizar esperas por elementos específicos, ejecutar scripts para modificar, crear, eliminar objetos y obtener respuestas, simular movimientos del mouse, simular comportamiento de teclas. En fin brindar las funcionalidades que logren simular posibles comportamientos de un usuario real en la página web.

La clase anterior necesita una estructura para funcionar correctamente, dicha estructura debe ser lo suficientemente flexible para manejar instancias del *AvanticaWebDriver* en tiempo de corrida (con el objetivo de no perder el hilo de ejecución) y agregar conjuntos de casos de pruebas en clases Java. Esas clases son llamadas suites, y deben comunicarse con clases intermedias que realizarán llamadas a los métodos presentes en el núcleo de la aplicación.

Este repositorio será la base para toda estrategia de automatización dentro de Avantek. Adicionalmente, se deberá crear una metodología de pruebas de desempeño y carga (benchmarking).

Avantek Software requiere llevar el proceso de pruebas de desempeño y carga al área operativa, dicho proceso permite conocer el comportamiento los programas ante envíos de información masivos y constantes durante periodos prolongados de tiempo, lo que permitirá conocer la verdadera capacidad de la aplicación, corregir anomalías y así asegurar un mayor nivel de robustez en sus productos.

Actualmente el plan que abarca esta área no se está implementando por lo que la metodología permitirá mejorarlo y llevarlo al nivel operativo.

1.2.2 Enunciado de la solución

Se debe crear un repositorio de scripts reutilizables para la plataforma de automatización de pruebas utilizando Selenium WebDriver y JUnit bajo el lenguaje de programación Java en el entorno de desarrollo Eclipse.

Selenium WebDriver provee funcionalidades que permiten el control del navegador Web, por su parte JUnit da soporte para crear casos de pruebas automáticos. Ambas herramientas son indispensables para el desarrollo de la solución la cual consiste en una estructura de clases que va a permitir la adición de clases de pruebas personalizadas correspondientes a escenarios de páginas de la aplicación a probar.

Cada página web posee elementos que van a ser objeto de pruebas durante las ejecuciones de las *suites*, en muchas ocasiones será necesario enviar valores a los elementos de la prueba, esos valores estarán localizados un archivo de propiedades, cuya extensión debe ser *.properties*.

Java da soporte para el paso de mensajes entre clases y archivos *.properties* por lo que se debe crear una clase de comunicación, la cual contendrá los métodos utilizados para enviar los datos solicitados a las clases correspondientes.

Los casos de pruebas automáticos correspondientes a una página web estarán localizados en una clase específica, cada uno utilizará la anotación *@Test* y el método *assert* de JUnit. El primero indicará que el método siguiente es un caso de prueba, y el segundo determinará si el caso pasó o falló.

Ejemplo de casos de prueba utilizando JUnit en Java.

```
      @Test
       public void pruebaAbrirPublicaciones(){
         pub.AbrirDialog ("dialog1");
          Assert.assertTrue(pub.VerificarDialogAbierto
("dialog1"));
```

Cuadro 2. Caso de prueba utilizando JUnit en Java

Cada caso de prueba realiza llamadas a funciones que contienen los pasos que va a seguir dicha prueba. Esas funciones están localizadas en una clase diferente (cada clase de pruebas debe contener su correspondiente clase de funciones de pasos), la cual va a ser intermediaria entre el driver (clase controladora del navegador) y la clase que contiene las pruebas.

Se debe crear una clase que llame a las suites de pruebas requeridas, y así crear una suite de clases.

Por otro lado clase controladora del navegador va a utilizar las funcionalidades de Selenium Web Driver para desarrollar métodos que ejecuten acciones desde las más simples hasta las más complejas.

La segunda parte de la práctica consiste en la creación de un plan operativo de benchmarking. Dicho plan será el resultado de una serie de análisis los cuales deben incluir:

a) Creación de plantillas de documentación de pruebas de desempeño y carga.
b) Creación de procesos de planeación para la realización de planes de pruebas de desempeño y carga.
c) Creación de procesos de ejecución para pruebas de desempeño y carga.
d) Creación de procesos de cierre de pruebas de desempeño y carga.

Además se debe realizar minuciosos análisis sobre herramientas utilizadas para este fin. Dicho análisis debe incluir ventajas, desventajas, características, modos de obtención de datos y despliegue e interpretación de datos. Con el objetivo de determinar las herramientas a utilizar para este propósito.

1.2.3 Stakeholders

- ✓ Rodrigo Vargas
  - Unidad: PMO.
  - Cargo: Gerente de Producción.

- ✓ Mario Núñez

- Unidad: QA.
- Cargo: Líder de Aseguramiento de la Calidad.

✓ Francisco Barquero
- Unidad: QA
- Cargo: Líder del grupo de especialización de pruebas Automáticas.

✓ Gilberto Martínez
- Unidad: QA.
- Cargo: Desarrollador.

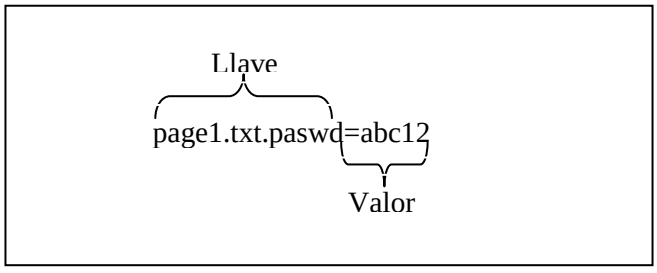
1.2.4 Solución Propuesta

Archivo .properties

Cada página web posee elementos y propiedades, las cuales van a ser objeto de pruebas en cada ejecución de los casos de pruebas automatizados y serán accedidos e identificados a través del driver de selenium. Para lograr acceder a esos elementos es necesario brindar una localización que será proporcionada en la declaración del caso de prueba.

En muchas ocasiones será necesario enviar valores a los elementos a prueba, esos valores estarán localizados un archivo de propiedades, cuya extensión es .properties.

El archivo de propiedades es utilizado para almacenar una llave (key) y un valor (value) separados por un signo de igual. Donde llave va a brindar una descripción y valor va a contener el dato a utilizar que será utilizado para establecer comunicación con el objeto.



Cuadro 2. Formato del archivo properties.

Especificación de Clases

El Framework de automatización va a estar compuesto por las siguientes clases las cuales se pueden apreciar integradas en la *imagen 4*:

a) BaseTest
b) BasePage
c) AvanticaWebDriver
d) TestSuite
e) LoginLogout
f) LoginTest
g) LoginPage
h) Messages
i) DriverManager

Las clases anteriores proporcionan los medios necesarios para iniciar a automatizar casos de prueba, los cuales van a utilizar dos clases adicionales, una de ellas va a contener la declaración de cada uno de los casos de prueba. En la otra clase se van a desarrollar los métodos correspondientes de los escenarios a probar, será la clase intermedia que utilizará las funciones declaradas del núcleo *AvanticaWebDriver*.

En la *imagen 4* se puede apreciar las clases adicionales, las cuales son:

a) ModuleXFunctionalTest
b) ModuleXPage

ModuleXFunctionalTest representa las clases que van a contener los casos de pruebas, que por lo general van a estar asociados a una página web específica de la aplicación. Cada uno de los casos de pruebas dentro de estas clases va a estar compuestos por una serie llamadas a funciones diseñadas para probar escenarios específicos de la página. Esas funciones van a estar localizados en las clases ModuleXPage, las cuales estarán conformadas por las funciones que realizan los pasos requeridos para ejecutar dichos escenarios, esta clase es intermedia, utilizada para enviar instrucciones a la clase *AvanticaWebDriver*, y recibir datos para enviarlos a su correspondiente clase de pruebas.

Cada una de las clases *ModuleXFunctionalTest* debe incluirse en *TestSuite* para su ejecución.

Clase BaseTest

Esta clase se utiliza para definir las funciones que se deben ejecutar antes de iniciar con las pruebas, por ejemplo, si se necesita realizar un ingreso a la aplicación, en esta clase se realizan las llamadas a la función correspondiente que realiza esa operación. Lo mismo ocurre con el cierre de sesión.

Estos métodos requieren anotaciones para su ejecución, situación por la cual se deben importar las siguientes librerías de JUnit: *org.junit.After* y *org.junit.Before*. Estas anotaciones, respectivamente van a indicar qué método se ejecutará antes y después de cada caso de prueba. Por Ejemplo:

a) El *@before* y el *@after*.

Clase BasePage

Esta clase es utilizada para compartir variables globales que deben ser heredadas a las clases intermedias, como por ejemplo una variable global que indique el nombre del archivo de propiedades.

Clase DriverManager

DriverManager contiene una instancia estática de la clase *AvanticaWebDriver*, durante la primera llamada crea la instancia, que es utilizada para realizar las interacciones con el browser.

Clase TestSuite

El TestSuite se encarga de ejecutar clases que contienen casos de prueba, su declaración es de la siguiente manera:

```
@RunWith(Suite.class)

@Suite.SuiteClasses({LastCustomer.class

})
```

Cuadro 3. Declaración de un Test Suite

El código anterior utiliza las funcionalidades de JUnit para realizar las llamadas las clases con las pruebas. En el espacio *LastCustomer.class* se indican las clases a ejecutar. Se pueden agregar las requeridas separándolas mediante comas.

Clase LoginLogout

Esta clase contiene las funciones que se utilizan para realizar el inicio y el cierre de sesión en la aplicación, y es utilizada por la clase llamada *LoginPage* la cual realiza llamadas a dichas funciones.

Clase LoginPage

Esta clase contiene los escenarios que serán utilizados por la clase que ejecuta las pruebas relacionadas al inicio y cierre de sesión, necesita una instancia de la clase *LoginLogout* para acceder a los métodos que realizan dichas operaciones.

Clase LoginTest

Existen diversas verificaciones que se deben realizar en los campos de inicio de sesión, esta clase tiene asociado los casos de pruebas que realizan llamadas a los escenarios localizados en la clase *LoginPage* para realizar dichas verificaciones.

Clase Messages

Esta clase se encarga de realizar la comunicación con el archivo de propiedades (archivo que va a contener los mensajes que serán enviados a determinada página de la aplicación a prueba). Tal comunicación se realiza mediante una función que recibe el nombre del archivo y el nombre de la llave a acceder. Retorna el valor correspondiente a la llave en formato String. Esta clase es importada, no instanciada con el objetivo de mejorar el rendimiento.

Librerías en Java:

a) *java.util.Properties*
b) *java.io.File*
c) *java.io.FileInputStream*
d) *java.io.IOException*

Clase AvanticaWebDriver

Esta clase va contener las funciones genéricas utilizadas durante la ejecución de los casos de prueba automatizados. Son funciones básicas, que realizan acciones puntuales sobre el navegador importando librerías de Selenium Web Driver:
Librerías requeridas en Java:

a) org.openqa.selenium.;*
b) org.openqa.selenium.interactions.Actions;
c) org.openqa.selenium.support.ui.ExpectedCondition;
d) org.openqa.selenium.support.ui.WebDriverWait;

Clases de ModuleXFunctionalTest

En estas clases se van a encontrar las pruebas que se van a ejecutar. La agrupación de estas pruebas por lo general se asocia con escenarios de una página y heredan las características de *BaseTest.* La declaración de cada método de prueba va antecedida por la anotación *@Test* para indicar que se desea ejecutar esa prueba, en caso contrario se utiliza *@Ignore.*
Cada caso de prueba se puede utilizar métodos de comprobación para verificar si la operación se realizó como se esperaba, esos métodos, llamados *assert* nos permiten realizar esas verificaciones y de acuerdo al resultado, el estado será fallido o aprobado.
Librerías requeridas en Java:

a) *junit.framework.Assert.*
b) *org.junit.Test.*

Clases de ModuleXPage

Estas clases van a estar conformadas por funciones que representarán escenarios de pruebas y realizarán llamadas a funciones básicas y puntuales localizadas en la clase *AvanticaWebDriver*.

Las funciones de esta clase por lo general van a retornar un valor booleano que será utilizado para realizar los *asserts* en la clase correspondiente ModuleX*FunctionalTest* hereda los atributos de *BasePage.*

1.2.5 Requerimientos no funcionales

a) Desempeño
b) Usabilidad
c) Escalabilidad
d) Mantenibilidad
e) Flexibilidad

1.3 Análisis de los Riesgos

Tabla 3. análisis de riesgos

Riesgo	Estrategia de Mitigación	Impacto
Conocimientos insuficientes en las herramientas a utilizar pueden causar retrasos en el desarrollo del proyecto.	Preparar capacitaciones con personal especializado en el área, para reducir tiempo dedicado a la investigación.	Alto
Falta de una especificación detallada en los requerimientos del proyecto puede provocar desvíos de acuerdo a los objetivos propuestos.	Constantes reuniones con el líder de QA para determinar el rumbo del proyecto.	Alto
Nuevas versiones liberadas de las herramientas a utilizar pueden causar problemas de configuración y compatibilidad.	Realizar la configuración del entorno utilizando versiones estables de las herramientas a utilizar.	Medio
Grupo de especialización de automatización relativamente nuevo, pocas personas con experiencia y con poca disponibilidad.	Definir lapsos en los cuales la o las personas especializadas puedan evacuar dudas sobre problemas surgidos durante el desarrollo.	Medio
Aparición de dependencias inesperadas, esto requeriría una revisión y rediseño del proyecto.	Reunión con los expertos en automatización del equipo para definir el diseño a seguir.	Alto

1.4 Objetivos y Alcances del sistema

1.4.1 Objetivo General

Desarrollar la estrategia operativa de scripts para generación de Casos de Pruebas Automáticos y estrategia operativa para la ejecución e implementación de las pruebas de Rendimiento.

1.4.2 Objetivos Específicos

a) Analizar el proceso de pruebas automáticas a desarrollar visualizando múltiples escenarios.

b) Diseñar el proceso de pruebas automáticas mediante la creación de un modelo conceptual.
c) Documentar el proceso de configuración de Selenium WebDriver, JUnit y Eclipse para un ambiente de desarrollo de pruebas automáticas y ejecución pruebas de Rendimiento.
d) Desarrollar Scripts genéricos en Selenium Web Driver para pruebas automáticas.
e) Documentar mejores prácticas, diseño y estándares de programación para pruebas automáticas.
f) Analizar diversas herramientas de desempeño y carga.
g) Documentar características, ventajas y desventajas de herramientas de desempeño y carga.
h) Crear una metodología para pruebas de desempeño y carga.
i) Documentar estándares de ejecución y diseño de pruebas de rendimiento.

1.4.3 Alcances

El proyecto cubre las etapas de:

a) Desarrollo de los requerimientos de la aplicación.
b) Análisis y diseño de la aplicación.
c) Documentación del entorno de desarrollo
d) Configuración de entorno de desarrollo.
e) Desarrollo de scripts genéricos usando Selenium WebDriver y JUnit.
f) Realización de Pruebas (informales)
g) Documentación del framework a desarrollar.
h) Análisis de herramientas de desempeño y carga.
i) Documentación de características de herramientas de desempeño y carga.
j) Desarrollo de metodología para pruebas de desempeño y carga.

Capítulo 2
Solución Implementada

2.1 Arquitectura conceptual

El Framework de automatización pone a disposición un entorno en donde se pueden desarrollar casos de pruebas para su ejecución automática sobre un navegador web determinado.

Dicho Framework tiene la capacidad de ejecutar las pruebas localmente mediante la utilizac.-ión de Selenium WebDriver o de manera distribuida utilizando Selenium Grid, con el objetivo de realizar un procesamiento paralelo para disminuir los tiempos de ejecución.

Selenium Grid, es una herramienta de Selenium [1] que se encarga de distribuir las pruebas entre los diferentes nodos registrados en un nodo principal, también llamado hub, en la sección Componentes y Subsistemas se brinda una descripción más detallada.

El siguiente diagrama muestra la arquitectura del Framework desarrollado cuando se está ejecutando localmente mediante Selenium WebDriver sin hacer uso del Grid:

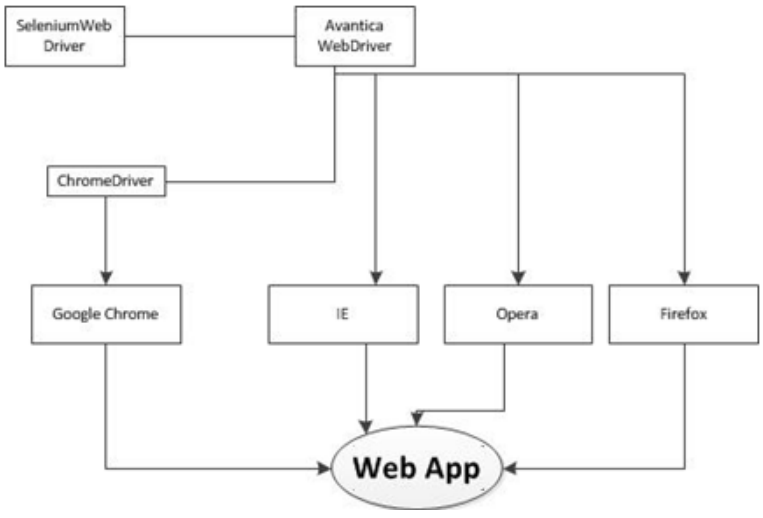


Imagen 1. Framework ejecutándose localmente.

La integración entre Avantica WebDriver y Selenium WebDriver permite realizar las operaciones sobre los navegadores web en los cuales se correrán los casos de prueba. Para

el caso de Google Chrome es necesario utilizar un controlador que se encargará de manejarlo, llamado ChromeDriver desarrollado por el equipo de Chromium en conjunto con el equipo de Selenium [1].

La arquitectura anterior muestra los componentes utilizados por el Framework Avantica WebDriver durante la ejecución de casos de prueba de manera local, la siguiente arquitectura muestra los componentes que requiere el Framework cuando está utilizando Selenium Grid para la ejecución de las pruebas en multilples nodos o computadoras:

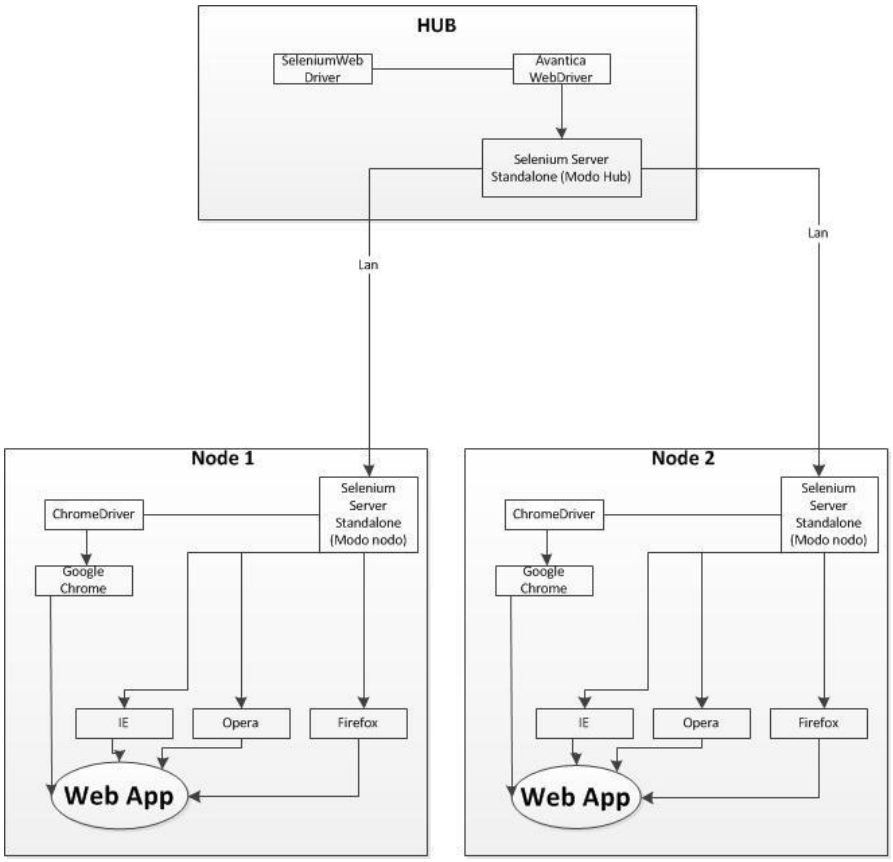


Imagen 2. Framework ejecutándose mediante Selenium Grid.

En el Hub se inicializa el Selenium Server Standalone (El cual administra la distribución de casos de prueba) con la configuración requerida quedando a la espera de nuevos nodos; en el momento en que se configura un nuevo nodo con la dirección IP del hub, éste lo reconoce e inmediatamente envía carga para que sea procesada si es el caso.

En cada nodo es necesario que esté corriendo el Selenium Server Standalone configurado en modo nodo, este proceso es detallado en la sección Componentes y Subsistemas.

2.2 Diagrama de Clases

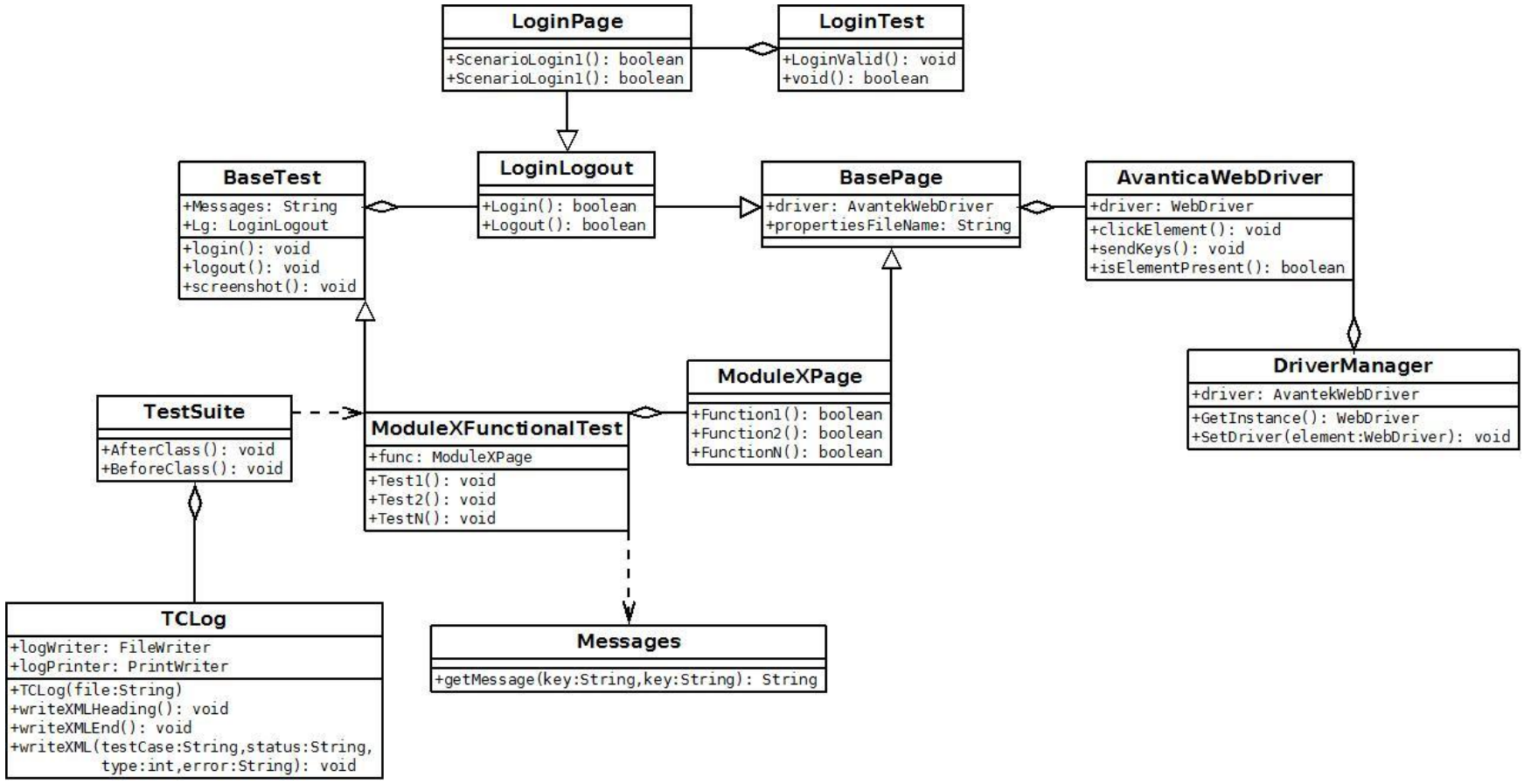


Imagen 3. Diagrama de clases de Avantica WebDriver.

2.3 Interfaz de Usuario

El Framework no posee una interfaz gráfica de usuario, sino hace uso de las herramientas que pone a disposición JUnit para la visualización del comportamiento de cada uno de los casos de prueba en el entorno de desarrollo Eclipse [2].

JUnit se encuentra incluido en las últimas versiones de Eclipse, y su uso se realiza importando las librerías que habilitan la declaración de los casos de prueba (**`org.junit.Test`**) o la declaración de las suites (**`org.junit.runner.RunWith`** y **`org.junit.runners.Suite`**) las cuales ponen a disposición la etiqueta *@Test* que indicarán la ubicación del método de prueba [5]. *Véase imagen 4.*

```
@Test
public void TC29_cancelCopyControlAction() {
    final WebElement wait = driver.findElement(By.id("apiRequestLoader"));
    driver.waitAndClick(By.xpath(".//*[@id='navigation']/div[1]/ul/li[4]"));
    driver.waitForCondition(new ExpectedCondition<Boolean>(){@Override public Boolean apply (WebDriver d){
        return wait.getAttribute("style").equals("display: none;");}});
```

Imagen 4. Declaración de un caso de prueba.

La ejecución de las pruebas se logra dando clic derecho sobre la clase suite o bien sobre el método de prueba y seleccionando la opción correr como *JUnit Test. Ver imagen 5*. De este modo JUnit ejecuta el navegador y abre la ventana en Eclipse, la cual mostrará el progreso y los resultados que se van obteniendo. *Véase imagen 6.*

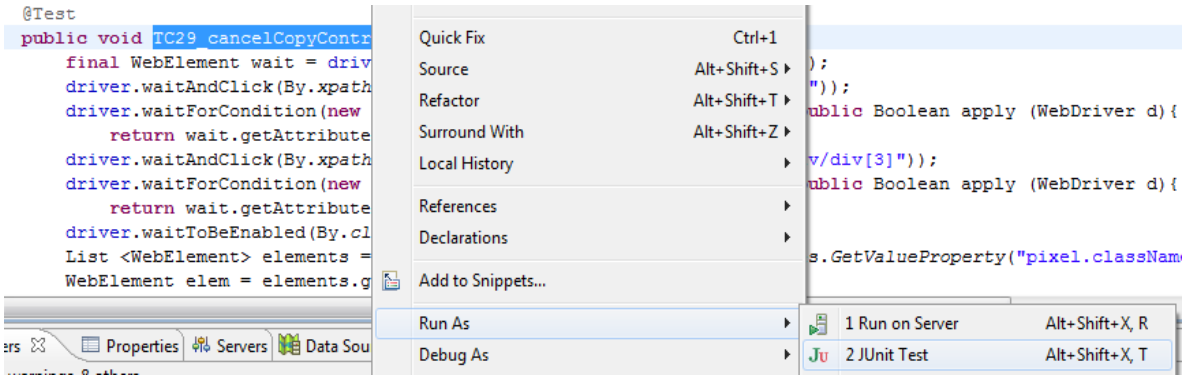


Imagen 5. Ejecución de un caso de prueba mediante JUnit.

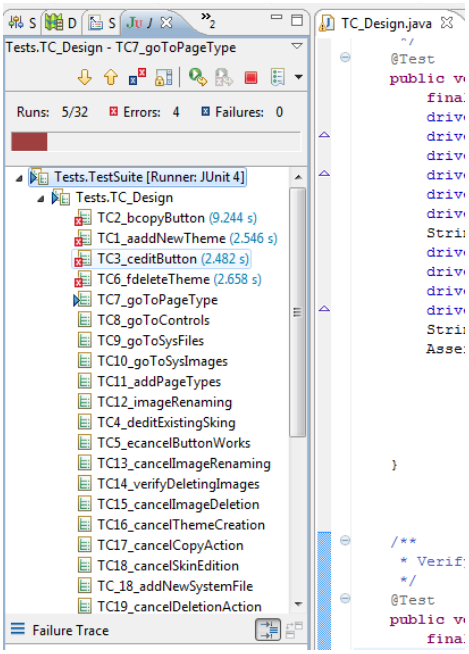


Imagen 6 Ejecución de una suite de pruebas.

Mediante el uso de ciertas funciones se reciben entradas que son utilizadas para determinar si el caso automatizado de prueba ha fallado o si ha sido ejecutado correctamente. Si el caso ha fallado, se indica con color azul, si el caso es aprobado, se indica con color verde y si ha ocurrido un error durante el proceso, se marca con rojo. *Véase imagen 7.*

Un error ocurre cuando el flujo normal del caso automatizado de prueba sufre alguna anomalía, y las operaciones que lo componen no pueden acceder a los componentes requeridos o bien, se ha sobrepasado el tiempo establecido para la espera de un elemento web.

Para acceder al log de cada caso de prueba que ha sido ejecutado y que ha fallado o ha terminado en error, basta con hacer clic sobre su respectivo ícono en la ventana que JUnit ha abierto en Eclipse IDE, (En la *imagen 7* se puede apreciar un caso de prueba fallado que ha sido seleccionado) de esta manera la traza del error es desplegado con el objetivo de encontrar su origen lo más pronto posible.

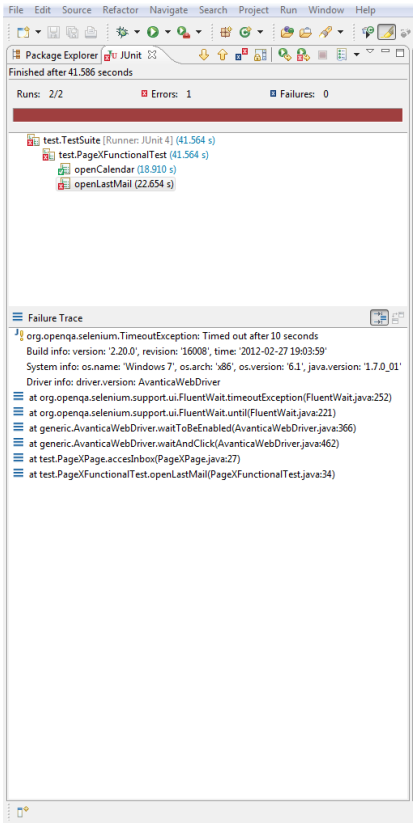


Imagen 7 Resultado de ejecución de una Suite de pruebas.

Adicionalmente se ha construido una clase, la cual puede ser visualizada en el diagrama de clases, llamada *TCLog*, la cual provee funciones necesarias para la escritura de un archivo el cual va a contener el resultado de todos los casos de prueba ejecutados en formato XML. Para cada caso de prueba se escribe el nombre, el estado, y si ha fallado se escribe la ruta por la que ha pasado el método, hasta su fallo, en inglés conocida como *Stack Trace*. Este archivo puede ser visualizado en cualquier navegador web.

```
<?xml version="1.0" encoding="UTF-8"?>
- <results>
  - <testcase external_id="TC3">
      <result>p</result>
      <notes>Automatic Run</notes>
      <execution_type>1</execution_type>
    </testcase>
  - <testcase external_id="TC1">
      <result>f</result>
      <notes>Automatic Run Failed, Cause: Not Avai
```

Imagen 8 - Fragmento del archivo XML con resultados de una suite de pruebas.

2.4 Componentes y servicios

El Framework de automatización, además de la librería SeleniumWebDriver, incluye otros componentes de Selenium que proveen funcionalidades que son utilizadas para realizar diversas tareas, dichos componentes son:

2.4.1 Controlador De Google Chrome

Selenium WebDriver soporta una gran variedad de navegadores en varios sistemas operativos ejecutando instrucciones de manera nativa, pero debido a diversas circunstancias el browser Google Chrome necesita utilizar un componente adicional para su funcionamiento óptimo en este entorno.

Este componente es el encargado de poner en ejecución el navegador para que Selenium WebDriver pueda accederlo y ejecutar instrucciones nativamente sobre él. Para llevar a cabo esto, es necesario realizar una configuración previa a la ejecución de los casos de prueba, la cual consiste en agregar al constructor de la clase AvanticaWebDriver la propiedad del sistema que indicará donde estará localizado el componente, en este caso fue probado en Windows, por lo que el componente tiene una extensión *.exe*, dicha propiedad es declarada de la siguiente manera:

```
System.setProperty(ChromeDriverService.CHROME_DRIVER_EXE_PROPERTY,
"chromedriver.exe");
driver = new ChromeDriver();
```

Cuadro 1. Configuración del navegador Google Chrome

Con la instrucción anterior el Framework estará configurado para ejecutar pruebas bajo el navegador Google Chrome, independientemente si se ejecuta en el grid o de manera local, cabe resaltar que las extensiones del componente *chromedriver* puede variar dependiendo del sistema operativo donde se ejecute.

2.4.1 Selenium Grid

Selenium Grid es una tecnología que permite la ejecución de pruebas en diferentes sistemas operativos y navegadores de manera paralela siempre que tengan instalados del JDK de java.

No es necesario realizar cambios en códigos con pruebas desarrolladas con Selenium WebDriver, lo que brinda flexibilidad y facilita su implementación, además realiza una distribución balanceada de carga entre todos los nodos registrados y utilizando las funciones de WebDriver establece una comunicación nativa con el navegador Web a utilizar.

Para implementar dicha herramienta en un entorno de pruebas es necesario descargar un componente denominado Selenium Server Standalone, el cual va a ser fundamental para la distribución de los casos de prueba entre las diferentes maquinas registradas.

La siguiente imagen muestra la distribución de casos de prueba a través del Selenium Grid:

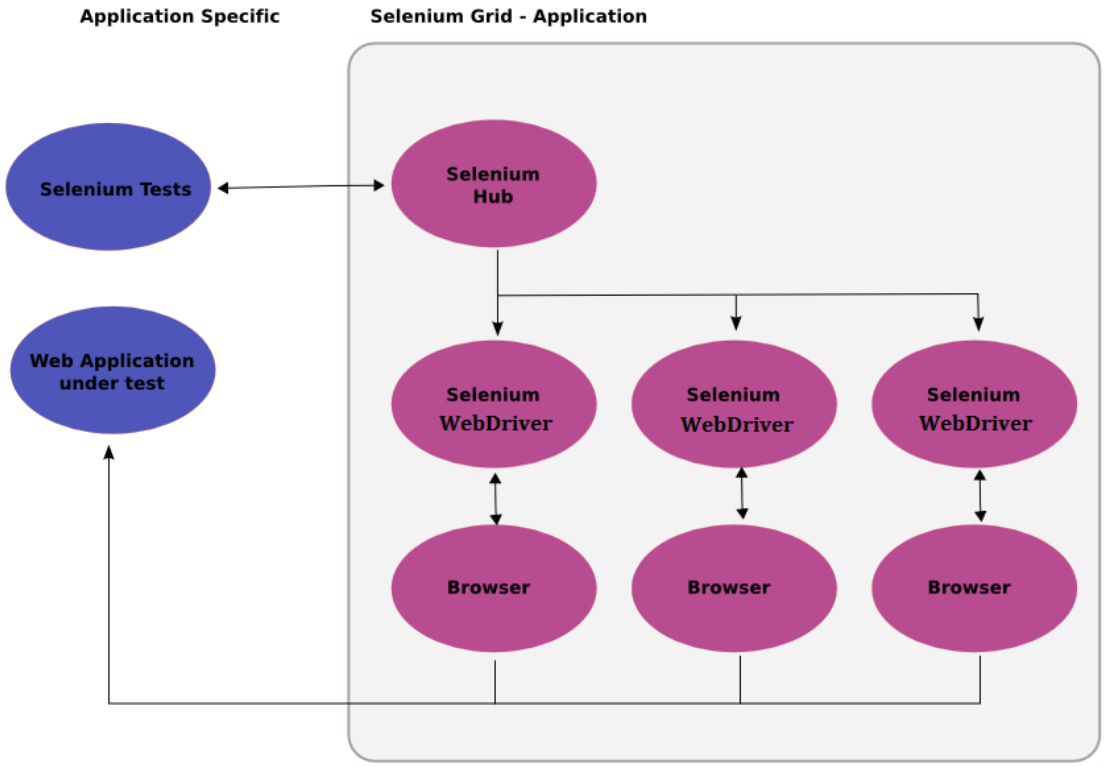


Imagen 9 Distribución de Casos de prueba a través del Grid

Los casos de prueba se van a ubicar en clases específicas del Framework de automatización AvanticaWebDriver, son agrupados en suites y únicamente estarán localizados en el hub, en el momento que se ejecuta el suite el hub toma dichas pruebas,

las organiza en una cola y las distribuye de manera balanceada a través de todos los nodos registrados.

Cuando una prueba finaliza, el nodo encargado de realizar la tarea envía la respuesta al hub, el cual se comunicará con JUnit enviando información que necesita para determinar si la prueba ha fallado o a sido exitosa, este proceso se realiza internamente.

Para su configuración se requiere un nodo principal, el cual distribuirá los casos de prueba a sus nodos secundarios registrados, que podrán estar en Linux, Windows, OS X y Solaris.

La configuración del nodo principal esta dado por los siguientes pasos:

a) Descargar la última versión del selenium server standalone disponible en la página oficial de Selenium.

b) Seguidamente hay que proceder a abrir una terminal y desplazarse hasta la carpeta donde se encuentra el archivo descargado.

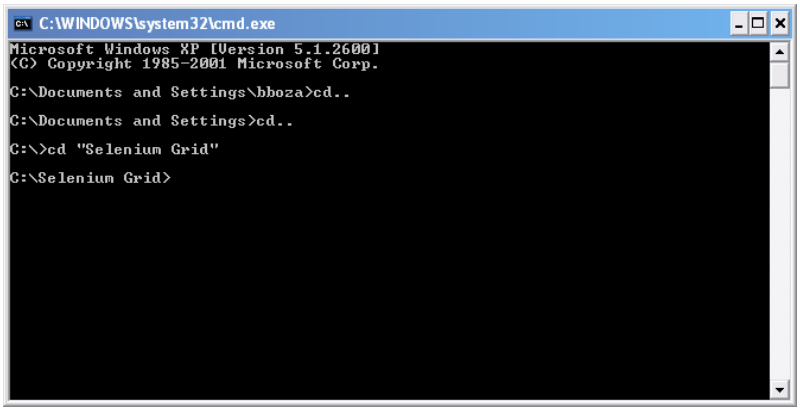


Imagen 10 Terminal Windows.

c) En este punto lo que procede es inicializar el servidor con la siguiente instrucción: *java -jar selenium-server-standalone-2.21.0.jar -role hub.* Como resultado, la consola ejecuta una serie de comandos que habilitan el servidor, el cual puede ser visualizado copiando la siguiente dirección en cualquier navegador web: http://localhost:4444/grid/console. Es importante mencionar que para que el servidor sea reconocido desde otras maquinas no se debe cerrar la consola.

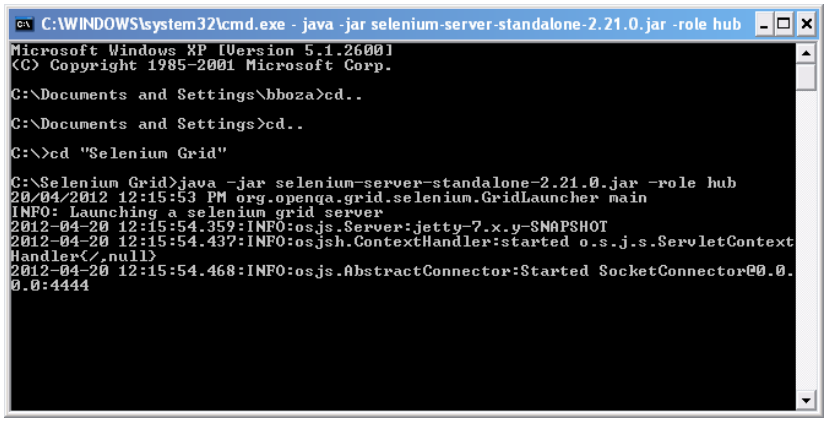


Imagen 41 Terminal Windows ejecutando Selenium Grid en modo hub.

Con el hub en funcionamiento, se procede a la configuración de los nodos para que ejecuten casos de pruebas automatizados, el proceso se realiza siguiendo los siguientes pasos:

a) Copiar el Selenium Server Standalone descargado para la configuración del hub en un directorio de la maquina que será utilizada como nodo.

b) Abrir una terminal y desplazarse hasta el directorio en donde se ubica el archivo.

c) Cada nodo debe tener una configuración en la cual se detallarán los browsers que se podrán ejecutar, la cantidad máxima de instancias a ejecutar, el puerto del nodo, el puerto del nodo principal o hub, en fin, es un conjunto de datos que brindarán los parámetros de ejecución. Esta configuración se realiza en un archivo con extensión json el cual se ubicará dentro del mismo directorio que contiene el archivo jar (Selenium Server Stadalone). El siguiente ejemplo representa la configuración básica de nodo en el archivo:

```
{
            "capabilities":
                    [
                            {
                                    "browserName":"firefox",
                                    "maxInstances":5
                            },
                            {
                                    "browserName":"chrome",
                                    "maxInstances":5
                            },
                            {
                                    "browserName":"internet explorer",
                                    "version":"8",
                                    "platform":"WINDOWS",
                                    "maxInstances":2
                            }
                    ],
            "configuration":
                    {
                            "maxSession": 5,
                        "port": 5555,
                        "host": ip,
                        "register": true,
                        "registerCycle": 5000,
                        "hubPort": 4444

                    }
            }
```

Cuadro 2. Archivo de configuración para nodo de Selenium Grid

d) Una vez creado el archivo json y almacenado en el mismo directorio del donde se ubica el jar, se procede a configurarlo con el siguiente comando: *java -jar selenium-server-standalone-2.21.0.jar -role webdriver -hub http://localhost:4444/grid/register -nodeConfig config.json -webdriver.chrome.driver=chromedriver.exe.*

La instrucción anterior habilita el nodo para que pueda ser utilizado para la ejecución de casos de prueba, en este ejemplo se utiliza *localhost*, pero puede ser reemplazado por la dirección IP de la máquina que cumple la función de hub. Si se va a utilizar el navegador Google Chrome es necesario indicar en donde se ubica en controlador que permitirá manipularlo, en este caso se encuentra en el mismo directorio.

El Framework desarrollado al utilizar las características presentes tanto en Selenium WebDriver, como en Selenium Grid puede ser ejecutado en una gran variedad de navegadores [6], los cuales son :

a) Firefox, en las versiones 11, 10, 9, 8, 7, 6, 5, 4 y 3.6
b) Internet Explorer en las versiones: 9, 8 y 7.
c) Opera en las versiones: 9 y 8
d) Google Chrome, en todas sus versiones mediante la utilización del controlador.

Conclusiones y Comentarios

El proyecto de práctica de especialidad, el cual recibe el nombre de *Estrategia operativa para pruebas de Automatización y Rendimiento* fue concluido con éxito, lo que permitió obtener un conjunto de productos entre los cuales están:

a) *Diseño y estándares*: Documento cuyo objetivo principal es definir el diseño y los estándares que seguirá el proyecto.

b) *Instalación y configuración de Selenium Web Driver con Eclipse y Visual Estudio utilizando Java y C#:* Documento cuyo objetivo principal es brindar una guía práctica para la instalación y configuración de entornos de desarrollo enfocados en la automatización de casos de prueba utilizando Eclipse IDE, Visual Studio 2010 y Selenium Web Driver.

c) *AvanticaWebDriver:* Es un framework de automatización de pruebas desarrollado en Java, el cual pone a disposición una estructura y un conjunto de métodos genéricos que facilitan dicho proceso.

d) *AvanticaWebDriverGrid:* Es un framework de desarrollo que incluye todas las funcionalidades de *AvanticaWebDriver* pero que incluye soporte para la ejecución de casos de prueba de manera distribuida utilizando una red de computadoras.

e) *Instalación y configuración de Selenium Web Driver usando AvanticaWebDriverGrid:* Documento cuyo objetivo principal es brindar una guía práctica para la instalación y configuración de entornos de desarrollo enfocados en la automatización de casos de prueba utilizando Eclipse IDE, Visual Studio 2010 y Selenium Grid para su ejecución de manera distribuida en una red de computadoras.

Mediante la utilización de *AvanticaWebDriver* y *AvanticaWebDriverGrid* se llevaron a cabo ejecuciones automatizadas de pruebas utilizando entornos de producción reales, los

cuales retornaron resultados que permiten solventar las necesidades establecidas en los objetivos propuestos del proyecto.

Una de las pruebas realizadas en un entorno de producción real se dio con la automatización de aproximadamente doscientos casos de prueba, los cuales fueron ejecutados utilizando tanto Selenium Web Driver como Selenium Grid. Dicha prueba generó resultados que demuestran la utilidad del Framework ya que se utilizaron todas las características descritas en los apartados anteriores.

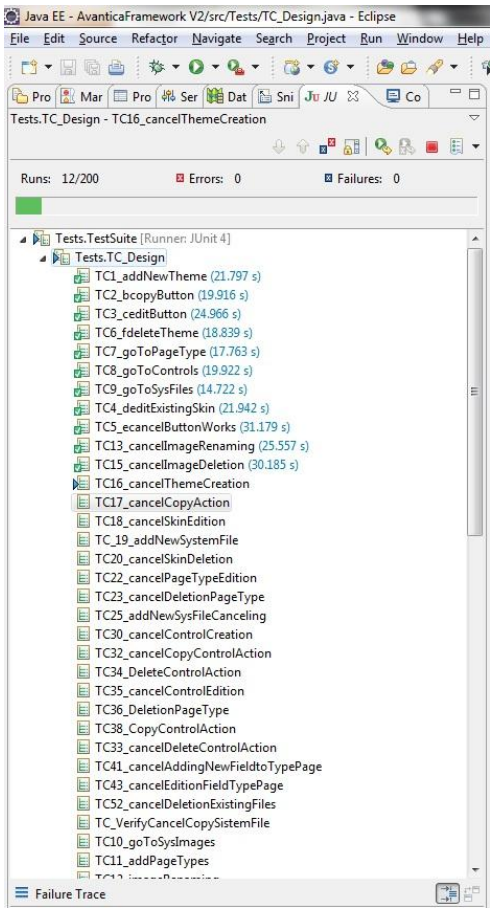


Imagen 5 Progreso de ejecución de doscientos casos de prueba.

Gran parte del éxito del proyecto se dio gracias al gran ambiente laboral que predomina en la empresa, en el cual la generación de conocimiento figura como prioridad y gracias a ello, muchos de los problemas técnicos que surgieron durante el desarrollo se resolvieron con ayuda de miembros del equipo, lo cual fue de gran importancia para la construcción de un producto de alta calidad.

Referencias Bibliográficas:

1. Selenium Project. (2012). Selenium Documentation. Recuperado 8 de marzo, 2012, de http://seleniumhq.org/docs/.

2. JUnit.org. (2012). JavaDoc. Recuperado 8 de marzo, 2012, de http://kentbeck.github.com/junit/javadoc/latest/.

3. Larman, C. (1998). *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development (3rd Ed.).New Jersey, USA: Prentice Hall PTR.*

4. *Massol, V. (2004). Junit in action .Greenwich, Connecticut: Manning Publications Co.*

5. Hunt, A. y Dave, T. (2003). Pragmatic Unit Testing in Java with JUnit. *Dallas, Texas: The Pragmatic Bookshelf.*

6. Open QA. (2012). Platforms Supported by Selenium. Recuperado el 01 de mayohttp://seleniumhq.org/about/platforms.html