TITANIUM FRAMEWORK PARA AUTOMATIZACIÓN DE PRUEBAS DE SOFTWARE

TITANIUM FRAMEWORK FOR SOFTWARE TESTING AUTOMATION

Gilberto Sánchez Mares
Titanium Solutions
gilberto.sanchez@titaniumsolutions.org

Resumen

La automatización de pruebas de software consiste en utilizar herramientas y estrategias para reducir la intervención o interacción humana en tareas redundantes, repetitivas o complejas. Esta automatización se ve reflejada en scripts de pruebas. Para poder simplificar aún más el tiempo de diseño y ejecución de dichos scripts se han desarrollado los frameworks, que son un conjunto de suposiciones, conceptos y prácticas. Se desarrolló un framework llamado "Titanium", para automatizar pruebas en plataformas web; se programó en Java y la intención principal es que se pueda difundir, utilizar y mejorar. Las principales ventajas son: puede ser utilizado por ingenieros de pruebas que tengan conocimientos técnicos o aquellos que no los tengan, es de fácil implementación, con un tiempo de aprendizaje muy corto y reduce costos en las tareas de pruebas.
Palabras Claves: Automatización de Pruebas, framework, pruebas de software, WebDriver.

Abstract

Automated testing consists of using tools and strategies to reduce human intervention or interaction in redundant, repetitive or complex tasks. This automation is reflected in scripts.

To further simplify the design and execution time of these scripts, frameworks have been developed, which are a set of assumptions, concepts and practices.

A framework called "Titanium" was developed to automate tests on web platforms; It was programmed in Java and the main intention is that it can be disseminated,

used and improved. The main advantages are: it can be used by test engineers who have technical knowledge or those who do not have them, it is easy to implement, very short learning time and it reduces costs in testing tasks.
Keywords: *Automation testing, framework, software testing, WebDriver.*

1. Introducción

El avance de las herramientas de desarrollo de software ha ayudado a que los desarrolladores diseñen aplicaciones más complejas (añadiendo o modificando procesos) reduciendo así el tiempo de creación de productos, sin embargo, la inseguridad que produce la calidad final del software incrementa, esto debido a que los ingenieros de pruebas tienen también un tiempo reducido para poder implementar de la mejor forma las pruebas necesarias.

Para asegurar un nivel de calidad antes de lanzar un prototipo de un software, es recomendable recurrir a la automatización de ciertas pruebas funcionales que aporten mayor confiabilidad sobre las principales funcionalidades del producto [J. Hui, 2008].

Según P. Laukkanen, la automatización de pruebas de software consiste en utilizar herramientas y estrategias para reducir la intervención o interacción humana en tareas redundantes, repetitivas o complejas. Esta automatización se ve reflejada en scripts (o por su traducción al español “secuencias de comandos”) con los que se puede aumentar de forma drástica la capacidad de probar el software, esto en lo que respecta a pruebas de regresión antes y después de la publicación de un prototipo [Guru99, 2018], [Laukkanen, 2006].

Frameworks en automatización de pruebas

Un framework (o marco de trabajo por su traducción al español) de automatización de pruebas es un conjunto de suposiciones, conceptos y prácticas que proporcionan apoyo a las pruebas de software automatizado [Guru99, 2018]. Estos framework integran las bibliotecas de funciones, fuentes de datos de prueba, detalles de objetos y diversos módulos reutilizables. Los componentes actúan como pequeños bloques de construcción que deben ser ensamblados para

representar un proceso de negocio. El framework proporciona la base de la automatización de pruebas y simplifica el esfuerzo [Fewster, 2001], [Laukkanen, 2006].

La principal ventaja de los frameworks que proporcionan apoyo a las pruebas de software automatizado es el bajo costo de mantenimiento. Por ejemplo, si hay cambio en los datos de un caso de prueba, entonces sólo el archivo donde se encuentran los datos debe ser actualizado y los scripts seguirán siendo los mismos.

Selenium

Para poder entender que es Selenium, se debe tomar la definición traducida, tomada de la página principal de SeleniumHQ [Kent, 2007], [Garg, 2014]: "*Selenium automatiza los navegadores. ¡Eso es! Lo que hagas con ese poder depende de ti. Principalmente, es para la automatización de aplicaciones web con fines de pruebas, pero ciertamente no se limita a eso. Las tareas aburridas de administración basadas en web pueden (¡y deben!) ser automatizadas.*" Una definición muy concreta y directa, pero vamos a ampliar un poco más la definición. Selenium es un set de herramientas de código abierto, que ayuda a automatizar acciones que un usuario puede realizar sobre aplicaciones web. Cada herramienta dentro del set tiene un enfoque diferente para apoyar el proceso de automatización de pruebas.

Los cuatro componentes de Selenium (figura 1) son:

- *IDE*: Se implementa como una extensión de Firefox, y permite grabar, editar y depurar pruebas. Se recomienda su uso para prototipos de pruebas, debido a que no es capaz de generar ciclos ni condiciones.
- *RC*: Esta es la primera herramienta de prueba web automatizada que permite a los usuarios utilizar un lenguaje de programación que prefieran.
- *WebDriver*: Implementa un enfoque más moderno y estable en la automatización de las acciones del navegador. WebDriver, a diferencia de RC, no se basa en JavaScript para Automatización. Controla el navegador comunicándose directamente.

- *Grid*: Se especializan en ejecutar múltiples pruebas a través de diferentes navegadores, sistemas operativos y máquinas. Puede conectarse con RC especificando el navegador, la versión del navegador y el sistema operativo. Hay dos elementos principales: hub y nodos.

Figura 1 Selenium suite.

Entre todas las herramientas Open Source, Selenium se considera un framework de pruebas de software altamente portátil y una de las mejores herramientas disponibles en el mercado actual para la automatización de aplicaciones web.

Selenium WebDriver

WebDriver es un framework de automatización web que permite ejecutar casos de prueba sobre diferentes navegadores (figura 2).

Figura 2 Ejemplos de navegadores soportados.

Debido a que es posible utilizar lenguajes de programación para la creación de scripts de automatización de pruebas, se pueden utilizar operaciones condicionales y bucles. Algunos de los lenguajes soportados son:

- Java
- C#
- Python
- Ruby
- PHP
- JavaScript

Arquitectura de WebDriver

A primera vista puede parecer que Selenium está manejando el navegador directamente desde el código, sin embargo, este proceso es un poco más complejo de lo que pareciera. La arquitectura de WebDriver está dividida en tres partes principales: **Lenguaje de vinculación**, **WebDriver Application Programming Interface (API)** y **Drivers** (figura 3).

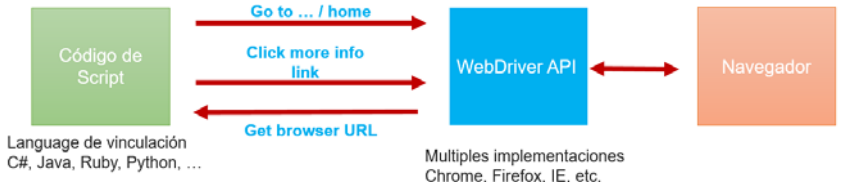


Figura 3 Arquitectura WebDriver.

Para ver cómo es que interactúan las partes entre sí, se puede basar en que se ha escrito un script de prueba usando Java (lenguaje de vinculación) para comunicarse con la API de Selenium, el código generado va a emitir comandos a través del *WebDriver wire protocol*, el cual es un servicio *Representational State Transfer* (Rest) capaz de interpretar dichos comandos. El driver es un ejecutable que básicamente escucha en un puerto de la máquina local cuando se ejecutan las pruebas y espera que los comandos entren. Una vez que comandos son captados por driver, son interpretados y ejecutados sobre navegador [Garg, 2014].

Ventajas y desventajas de WebDriver sobre otras herramientas

En tabla 1 se pueden encontrar ventajas y desventajas sobre otras herramientas automatizadas muy populares en el mercado [Garg, 2014], [IBM, 2017].

Tabla 1 Selenium WebDriver vs otras herramientas.

Característica	Selenium	IBM RFT	UFT
Lenguaje de scripting	Java, C#, Ruby, Python, Perl, JavaScript, PHP	Java y C#	VB Scripting
Tecnologías soportadas	Todas las tecnologías web	HTML, Java, AWT, SWT, Dojo	HTML, Java, .Net, WPF, SAP, Oracle
Soporte a navegadores	Internet Explorer, Chrome, Firefox, Safari, Opera	Internet Explorer, Chrome, Firefox	Internet Explorer, Chrome, Firefox
Soporte de Ambientes	Windows, Linux, OS X	Windows y Linux	Sólo Windows
Soporte Móvil	Sí	Sí	Sí
Manejo de Repositorio de objeto	Bueno	Muy bueno	Excelente
¿Su uso tiene un costo?	No	Sí	Sí
Opción de Record & Play	Disponible	Disponible	Disponible
Depuración	Depende del IDE	Muy bueno	Excelente
Soporte de Frameworks	Data Driven, Keyword Driven, Modularity, Test Library Architecture e Híbrido	Data Driven, Keyword Driven, Modularity e Híbrido	Data Driven, Keyword Driven, Modularity, Test Library Architecture e Híbrido
Continuous Integration	Posible usando Jenkins/ Hudson/ Cruise Control/ etc	Posible con Jenkins	Posible con ALM o Jenkins
Mecanismo de Reporte	No tiene un mecanismo integrado	Mecanismo integrado	Mecanismo integrado
Soporte	Comunidad Open Source	Soporte de IBM	Soporte de HP

2. Métodos

Titanium es un framework híbrido, que nace de la combinación de los frameworks: *Data-Driven Testing* (o pruebas dirigidas por datos, por su traducción al español) y *Keyword-Driven Testing* (pruebas dirigidas por palabras clave por su traducción al español). Además, se utilizó el patrón de diseño *Page Object Model* (o modelo de objetos de página), los cuales se explican a continuación:

Data-driven testing

Mientras que la lógica del caso de prueba reside en los scripts de prueba, los datos de prueba son separados y se mantienen fuera de dichos scripts. Los datos son leídos desde archivos externos (Excel, texto, CVS, bases de datos, objetos de

accesos de datos, etc.) y son cargados en variables [Guru99, 2018], [Laukkanen, 2006]. Dichas variables son usadas tanto como datos de entrada como datos de salida (verificación de valores). En Data-Driven Testing, se propone una mejor organización de los scripts de prueba y de esta manera reducir los costos de mantenimiento de éstos. Permite que tanto los datos de entrada y los resultados esperados puedan ser almacenados juntos y por separado de la secuencia de comandos en sí [Bahaggan, 2009]. La figura 4 ilustra el marco de trabajo.

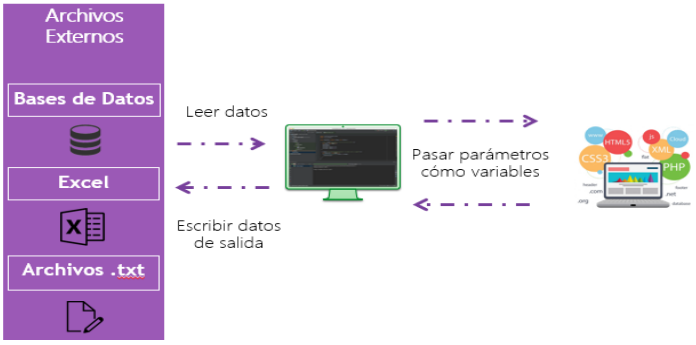


Figura 4 Funcionamiento de Data-Driven Testing.

Keyword-driven testing

Este framewrok requiere del desarrollo de tablas de datos y palabras clave, independientemente de la herramienta de automatización de pruebas utilizada para ejecutarlos [Guru99, 2018]. Las pruebas pueden ser diseñadas con o sin la aplicación. En una prueba dirigida por palabra clave, la funcionalidad de la aplicación bajo prueba se documenta en una tabla, así como en el paso a paso las instrucciones para cada prueba. También es desarrollado para ofrecer un enfoque más fácil de pruebas funcionales y de Pruebas de Procesos de Negocio pasando los parámetros adecuados para particulares palabras de acción en el marco. Este marco permite pruebas razonablemente intuitivas para ser desarrolladas y ejecutadas sin necesidad de modificar los scripts de prueba [Laukkanen, 2006], [Bahaggan, 2009].

Hay 3 componentes básicos en este framework: palabra clave, mapa de aplicaciones, función de componentes. La figura 5 ilustra el proceso para crear este framework.

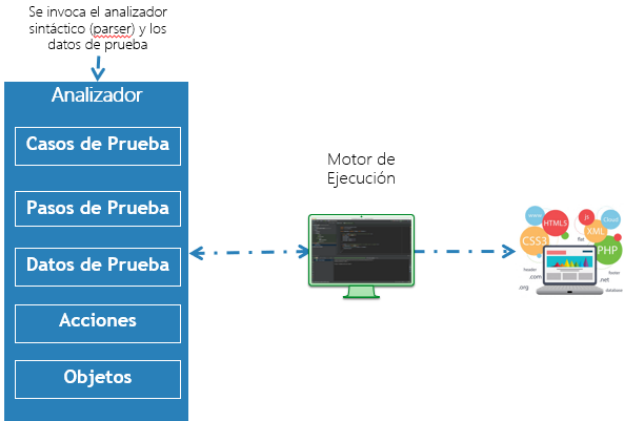


Figura 5 Funcionamiento de Keyword-Driven Testing.

Page object model

En esencia, Page Object modela la aplicación web y sirve como interfaz para las acciones que se pueden realizar y los datos que se pueden recuperar de ella. En su forma más simple y tradicional, un objeto de página es solo una colección de funciones comunes a una página web para evitar que se repita.

Si bien esto suele ocurrir cuando la comprensión del patrón de diseño del objeto de página finaliza para muchas implementaciones, es donde simplemente comienza para el objeto de página [Zylberman, 2010]. En la figura 6 se muestra la estructura de este patrón de diseño.

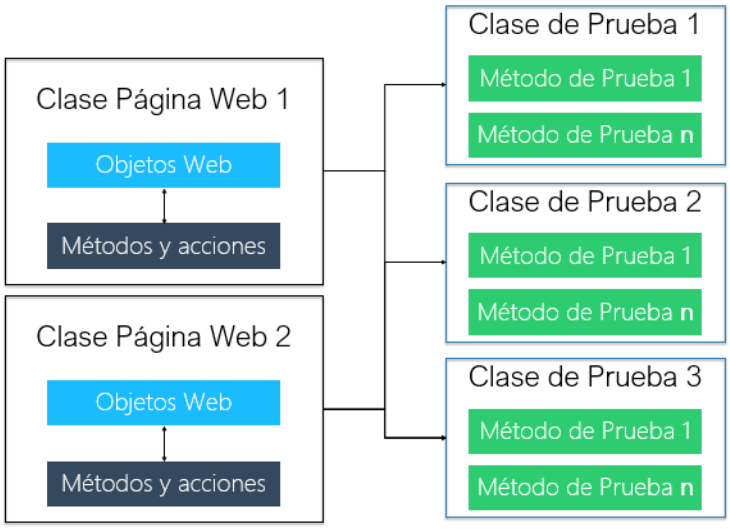


Figura 6 Estructura de Page Object Model.

Arquitectura de titanium

El flujo inicia dentro de la clase DriverScript.java, donde se encuentra el motor que leerá archivos y clases externas, además se cuenta con un archivo en Excel nombrado *DataEngine.xlsx*, el cual sirve como analizador sintáctico que contiene los casos de prueba y sus respectivos pasos, así como los objetos web obtenidos de la aplicación.

La clase ActionKeyword.java, contiene todos los métodos para interactuar con los objetos web, cómo lo es hacer un clic, escribir sobre una caja de texto, seleccionar un elemento en una lista desplegable, etc. En total, hasta el momento se han diseñado 73 métodos. El framework es capaz de generar video de la ejecución, capturas de pantalla en errores (pero no limitado sólo a estos, se puede capturar pantalla en cualquier momento), registros de los pasos ejecutados (en consola y archivo) y un reporte HTML con los resultados de la ejecución.

La ejecución de los casos de prueba se puede ejecutar en cualquier entorno de desarrollo integrado (IntelliJ, Eclipse, NetBeans, etc.), y puede ser ejecutado utilizando Maven. La arquitectura se muestra en la figura 7.

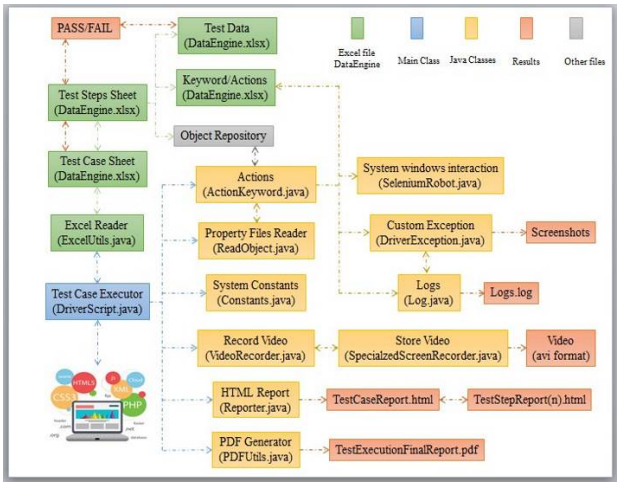


Figura 7 Arquitectura del framework titanium.

Estructura del proyecto

El proyecto principal contiene cuatro directorios:

- Config: Contiene las clases donde se almacenan las constantes, así como clases que ayudan a leer archivos y generar video.

- Engine: Cuenta con la clase que ejecuta los casos de prueba que se encuentran en el analizador sintáctico.
- Properties: En este directorio se almacenan todos los modelos.
- Utils: Se encuentran las clases que ejecutan las acciones que se realizarán sobre los objetos web, captura de excepciones, toma de capturas de pantalla y clase que se encarga de abrir los navegadores dependiendo del navegador y el sistema operativo.

En la figura 8 se puede apreciar la estructura completa del proyecto.

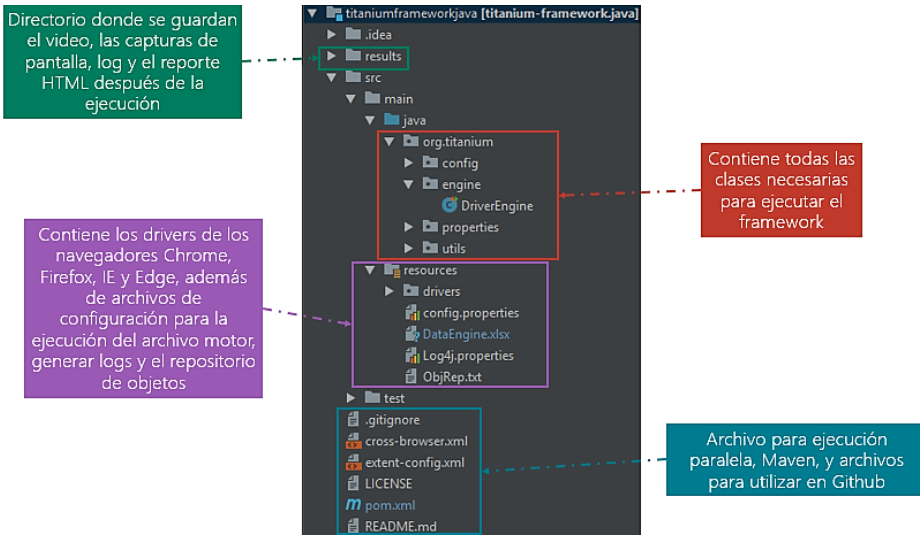


Figura 8 Estructura del proyecto titanium.

Archivo de configuración

En este archivo se puede indicar si se quiere abrir el video, los logs, el reporte HTML y el archivo *DataEngine.xlsx* inmediatamente después de la ejecución de los casos de prueba, así como indicar a que correo se quiere enviar algún error que haya surgido de la ejecución (figura 9).

Data-engine

En este archivo contiene cuatro hojas de alta relevancia para el proyecto, la hoja de los *casos de prueba*, los *pasos de prueba*, las *páginas* y las *configuraciones*, además de que se puede agregar una hoja con datos de prueba.

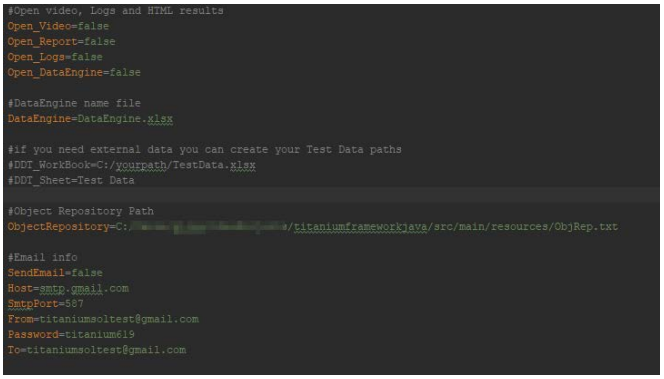


Figura 9 Ejemplo de archivo de configuración.

La hoja de casos de prueba contiene seis columnas (figura 10):

- ID: Número único de identificación del caso de prueba.
- Test Case ID: Código alfanumérico de identificación del caso de prueba.
- Description: Breve descripción de lo que hace el caso de prueba.
- Run: Se indica si se quiere ejecutar el caso de prueba o no.
- DDT Sheet: Se le debe indicar el nombre de la hoja de datos en la cual se encuentran los datos de prueba.
- Result: Se imprime el resultado de la ejecución.

En la hoja de pasos de prueba, existen ocho columnas (figura 11):

- Test Case ID: Código alfanumérico de identificación del caso de prueba.
- TS_ID: Código alfanumérico identificador de cada paso.
- Description: Descripción de cada paso de prueba.
- Page Name: Nombre de la página donde se encuentran nuestros objetos.
- Page Object: Alias del objeto, con el cual se identificará para realizar una acción sobre éste.
- Action Keyword: Acción a realizar sobre el objeto web.
- Data Set: Datos de prueba necesarios para el paso de prueba.
- Result: Se imprime el resultado de la ejecución.

En la hoja de páginas, se mapea cada objeto web con su respectiva página, tal y como se muestra en la figura 12 Hoja para almacenar páginas usando Page Object

Model. En configuración se encuentran todas las acciones, ya que el framework se puede aumentar y/o mejorar, se pueden agregar más acciones necesarias para cada proyecto. Por último, es posible iterar utilizando diferentes datos, para ello se puede crear una hoja de datos de prueba, sólo hay que recordar utilizar el mismo nombre de parámetro dentro de cada paso de prueba y la hoja de datos de prueba (figura 13).

	A	B	C	D	E
1	Test Case ID	Description	Run	DDT Sheet	Result
2	LogInStoreApp_01	LogIn in to the Online Store Application	Yes		
3	LogInStoreApp_02	Invalid LogIn in to the Online Store Application	Yes	Login Data	
4	LogInStoreApp_03	Invalid URL	No		

Figura 10 Hoja de casos.

A	B	C	D	E	F	G	H
Test Case ID	TS_ID	Description	Page Name	Page Object	Action Keyword	Data Set	Result
LogInStoreApp_01	1	Navigate to website			navigate	http://www.store.demoqa.com	
LogInStoreApp_01	2	Move to Element My Account button on the Top Right location	Store_Home_Page	btn_MyAccount	moveToElement		
LogInStoreApp_01	3	Click and hold on My Account button on the Top Right location	Store_Home_Page	btn_MyAccount	clickAndHold		
LogInStoreApp_01	4	Release My Account button on the Top Right location	Store_Home_Page	btn_MyAccount	release		
LogInStoreApp_01	5	Enter the Username in the UserName field	Store_LogIn_Page	txtbx_UserName	input	tiempouser	
LogInStoreApp_01	6	Enter the Password in the Password field	Store_LogIn_Page	txtbx_Password	input	RHJEzDXX#ZShqBxY	
LogInStoreApp_01	7	Click on Login button	Store_LogIn_Page	btn_LogIn	click		
LogInStoreApp_01	8	Wait for Some time			waitFor	5	
LogInStoreApp_01	9	Click on LogOut button	Store_Home_Page	btn_LogOut	click		
LogInStoreApp_01	10	Wait for Some time			waitFor	5	
LogInStoreApp_02	1	Navigate to website			navigate	http://www.store.demoqa.com	
LogInStoreApp_02	2	Click on My Account button on the Top Right location	Store_Home_Page	btn_MyAccount	click		
LogInStoreApp_02	3	Enter the Username in the UserName field	Store_LogIn_Page	txtbx_UserName	input	UserName	
LogInStoreApp_02	4	Enter the Password in the Password field	Store_LogIn_Page	txtbx_Password	input	PassWord	
LogInStoreApp_02	5	Click on Login button	Store_LogIn_Page	btn_LogIn	click		
LogInStoreApp_02	6	Wait for Some time	Store_LogIn_Page	lbl_Error	waitForElementVisible		
LogInStoreApp_02	7	Verify Label Error	Store_LogIn_Page	lbl_Error	getElementText	Expected	
LogInStoreApp_03	1	Navigate to website			navigate	http://www.store.demoqa.coms	
LogInStoreApp_03	2	Move to Element My Account button on the Top Right location	Store_Home_Page	btn_MyAccount	moveToElement		

Figura 11 Hoja de pasos.

	A	B	C
1	Page Name	Gmail_Login_Page	Gmail_Home_Page
2	Gmail_Login_Page	txtbx_EmailAddress	btn_Redactar
3	Gmail_Home_Page	btn_Next	txtbx_To
4		lbl_EmailDisplay	txtbx_Subject
5		txtbx_Password	txtbx_EmailBody
6		btn_SignUp	btn_SendMail
7			lnk_Inbox
8			lvw_InboxTray
9			lbl_InboxFirstItem
10			lbl_AcctOpt
11			btn_SignOut

Figura 12 Hoja para almacenar páginas.

Data Set
http://www.store.demoqa.com
UserName
PassWord
Expected

A	B	C
UserName	PassWord	Expected
testuser_selen_training1		ERROR: The password field is empty.
	QIiFzliqLIRY	ERROR: The username field is empty.
testuser_selen_training12	QIiFzliqLIRY	ERROR: Invalid login credentials.
		Please enter your username and password.

Figura 13 Hoja de datos de prueba.

Repositorio de objetos

En el repositorio de objetos (figura 14) se tiene el mapeo de los objetos que se encuentran en la columna de *Page Object* de la hoja de pasos de prueba, se deben localizar los elementos de la forma que está permitida con Selenium, por medio de: *Id*, *xpath*, *css selector*, *name*, *class*, *texto de link*, *texto parcial del link* o *tag.*

```
#************************** Home Page ***********************************
btn_MyAccount=//a[@title='My Account']
btn_LogOut=//a[@title='Logout']

#************************** Login Page **********************************
txtbx_UserName=log
txtbx_Password=pwd
btn_LogIn=login
lbl_Error=(//p[@class='response'])[1]
```

Figura 14 Ejemplo de repositorio de objetos.

3. Resultados

Una vez ejecutado el framework, se generan video, reporte HTML, registros y se imprimen los resultados en el archivo DataEngine.xlsx. También se puede enviar un correo indicando si algo funcionó mal o si terminó la ejecución de forma correcta.

Registros

Los registros muestran de forma detallada (fecha y hora), cuál fue la acción que se ejecutó, en el caso de que falle, se indica que línea de código indicó ese error

y cuál es la excepción obtenida (figura 15). Los registros se pueden visualizar en el entorno de desarrollo integrado en el que se esté trabajando, así mismo como en un archivo con extensión .log.

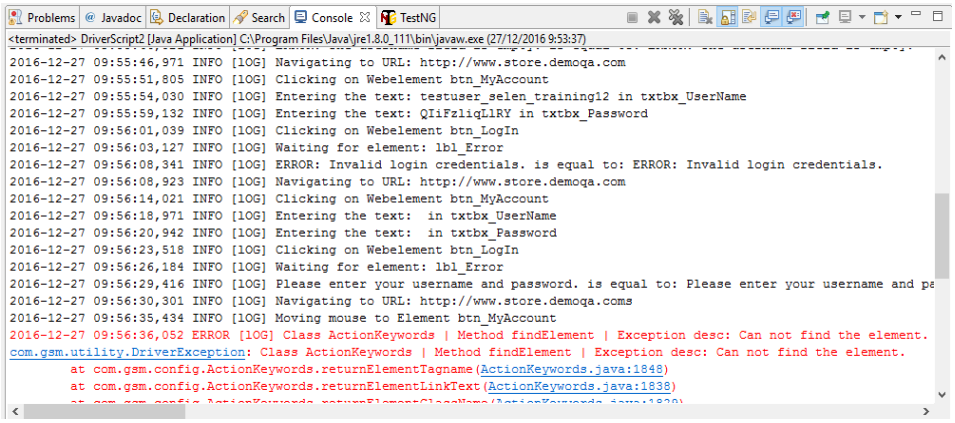
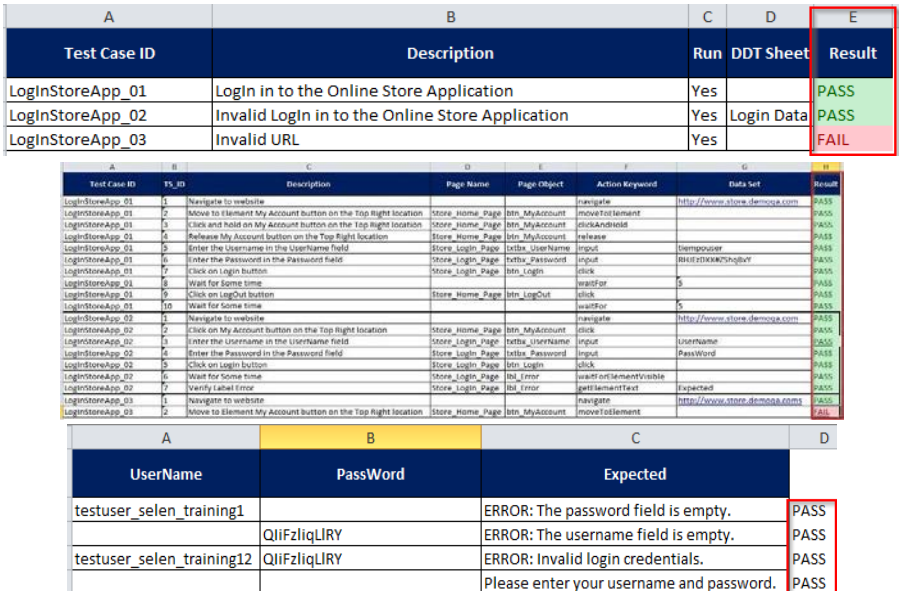


Figura 15 Ejemplo de archivo de registros.

Resultados en el archivo de Excel

Una vez finalizada la ejecución, dentro del archivo *DataEngine.xlsx*, se imprime si la ejecución del caso de prueba fue exitosa o falló (figura 16), si cada paso de prueba se ejecutó correctamente o no. En caso de tener que iterar, se indica si cada iteración se ejecutó de forma correcta o hubo algún error en el proceso.



A	B	C	D	E
Test Case ID	**Description**	**Run**	**DDT Sheet**	**Result**
LogInStoreApp_01	LogIn in to the Online Store Application	Yes		PASS
LogInStoreApp_02	Invalid LogIn in to the Online Store Application	Yes	Login Data	PASS
LogInStoreApp_03	Invalid URL	Yes		FAIL

A	B	C	D
UserName	**PassWord**	**Expected**	
testuser_selen_training1		ERROR: The password field is empty.	PASS
	QIiFzliqLlRY	ERROR: The username field is empty.	PASS
testuser_selen_training12	QIiFzliqLlRY	ERROR: Invalid login credentials.	PASS
		Please enter your username and password.	PASS

Figura 16 Resultado de cada iteración.

Reporte HTML

El reporte HTML que se genera al finalizar la ejecución contiene información detallada de cada caso de prueba y su resultado, así como cada paso de prueba y su hora de ejecución. Se puede ver de forma gráfica el total de casos de prueba que se ejecutaron de forma exitosa y los que no, así como características del equipo en donde se ejecutaron las pruebas, la figura 17 muestra un ejemplo de reporte de ejecución.

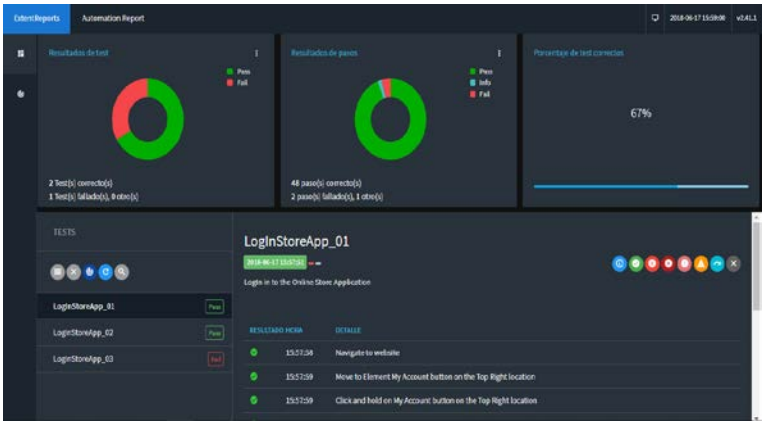


Figura 17 Reporte HTML de ejecución.

Video y correo

Por último, el framework es capaz de crear un video de la ejecución (figura 18) para tener más evidencia de cada ejecución, y en el caso de que algo falle, un correo informando del error, será enviado.

4. Discusión

La elección del framework en la automatización de pruebas será dependiente de la complejidad de cada proyecto y tomando en cuenta las ventajas y desventajas de cada marco. Además de que un factor importante es que la persona o equipos que se encargarán de crear el framework, deberán tener experiencia en desarrollo de software. Pero si hablamos del framework Titanium, nos daremos cuenta de que, desde su estructura, está lo suficientemente modularizado para poder agregar, quitar, o mejorar módulos, ayudando así a ingenieros de pruebas que no tengan experiencia en programación, para poder crear sus propios casos

de prueba automatizados en poco tiempo, ayudando a las empresas a reducir costos de tiempo y dinero en el proceso de calidad de software.

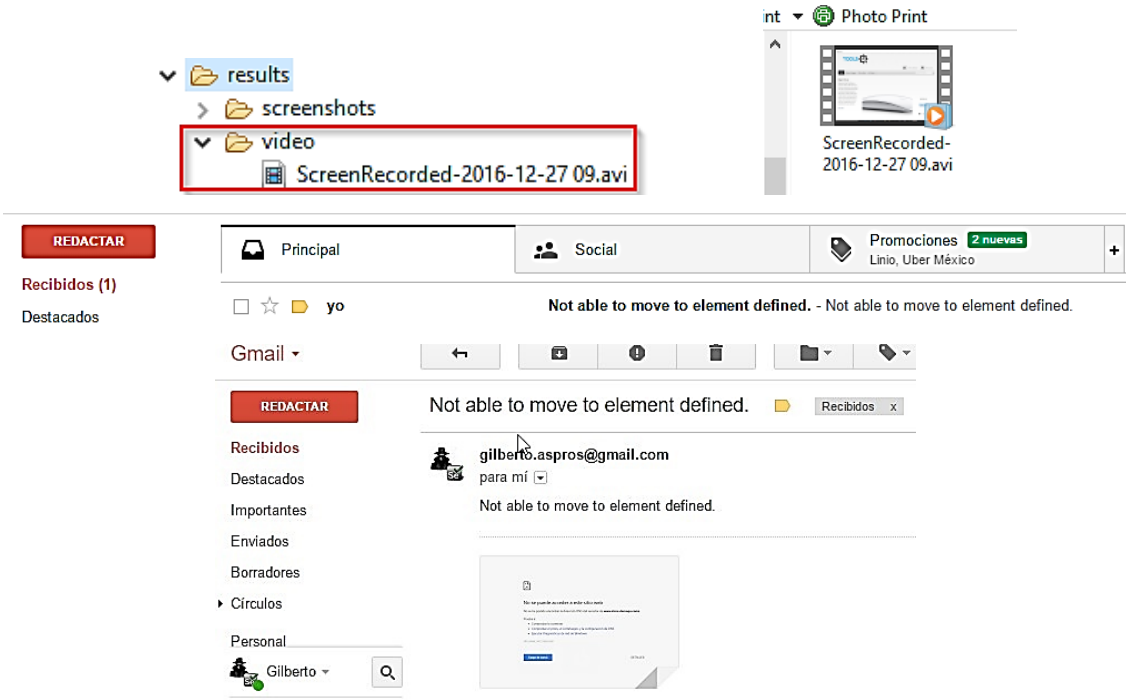


Figura 18 Video de ejecución.

5. Conclusiones

La automatización de pruebas, hablando de scripting o desarrollo, es una actividad realizada mediante la programación de software, es decir que un framework de automatización es un desarrollo en sí que se utiliza para realizar pruebas sobre otro desarrollo (aplicación objeto de prueba). El Titanium framework, ha sido implementado en varias empresas de México y el extranjero, las cuales han tenido una reducción en el tiempo de creación de pruebas y de ejecución, aminorando de gran forma el dinero invertido a estas mismas. El proyecto ésta disponible en https://github.com/gsanchezm/Titanium-Framework-Java para su descarga y puede ser accedido por cualquier persona.

6. Bibliografía y Referencias

[1] Bahaggan, Test Automation in Practice", Delft Universuty of Technology, the Netherlands, 2009.

[2] Fewster, Common Mistakes in Test Automation, Groovy Consultants, 2001.

[3] Garg, Test Automation Using Selenium WebDriver with Java, Adactln Group Pty Ltd, 2014.

[4] Guru99, All About Automated Testing", http://www.guru99.com/automation testing.html. Mayo 2018.

[5] Hui, L. Yuqing, L. Pei, G. Jing and G. Shuhan, LKDT: A Keyword-Driven Based Distributed Test Frameworks, International Conference on Computer Science and Software Engineering, 2008, pp. 719-722.

[6] IBM, Documentación de relational functional tester V9.1.0 https://www.ibm.com /support/knowledgecenter/es/SSJMXE_9.1.0/com.ibm.rational.test.ft.doc/rft_welcome.html, 2017.

[7] Kent, Test Automation from record/playback to frameworks, http://www.simply testing.com/, 2007.

[8] Laukkanen Data-Driven and Keyword-Driven Test Automation Frameworks M.S. thesis, Helsinki University of Technology, Helsinki, Finlandia, 2006.

[9] Zylberman Shotten, Test Language: Introduction to Keyword Driven Testing, http://www.softwaretestinghelp.com, pp. 1-7, 2010.